工程实践与科技创新课程：超并行机器学习与海量数据挖掘

第四讲（上）：深度学习及应用

吕宝粮
计算机科学与工程系
电信楼群3号楼431
Tel: 021-3420-5422
bllu@sjtu.edu.cn
http://bcmi.sjtu.edu.cn/~blu

Introduction to Deep Learning

- Background
- Convolutional Neural Networks
- Applications to EEG-based Emotion Recognition

Background

MIT Technology Review

10 BREAKTHROUGH TECHNOLOGIES 2013

Deep Learning

With massive amounts of computational power, machines can now recognize objects and translate speech in real time. Artificial intelligence is finally getting smart. →

Temporary Social Media

Messages that quickly self-destruct could enhance the privacy of online communications and make people freer to be spontaneous. →

Prenatal DNA Sequencing

Reading the DNA of fetuses will be the next frontier of the genomic revolution. But do you really want to know about the genetic problems or musical aptitude of your unborn child? →

Additive Manufacturing

Skeptical about 3-D printing? GE, the world's largest manufacturer, is on the verge of using the technology to make jet parts. →

Baxter: The Blue-Collar Robot

Rodney Brooks's newest creation is easy to interact with, but the complex innovations behind the robot show just how hard it is to get along with people. →

Memory Implants

A maverick neuroscientist believes he has deciphered the code by which the brain forms long-term memories. Next: testing a prosthetic implant for people suffering from long-term memory loss. →

Smart Watches

The designers of the Pebble watch realized that a mobile phone is more useful if you don't have to take it out of your pocket. →

Ultra-Efficient Solar Power

Doubling the efficiency of a solar cell would completely change the economics of renewable energy. Nanotechnology just might make it possible. →

Big Data from Cheap Phones

Collecting and analyzing information from simple cell phones can provide surprising insights into how people move about and behave – and even help us understand the spread of diseases. →

Supergrids

A new high-power circuit breaker could finally make highly efficient DC power grids practical. →

Major Breakthrough in 2006

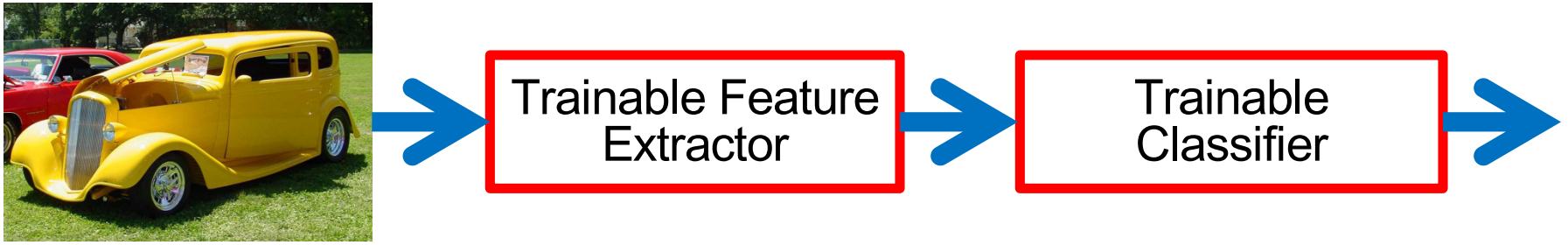
- Biological Ability to train deep architectures by using layer-wise unsupervised learning, whereas previous purely supervised attempts had failed

- Unsupervised feature learners:
  - RBMs
  - Auto-encoder variants
  - Sparse coding variants

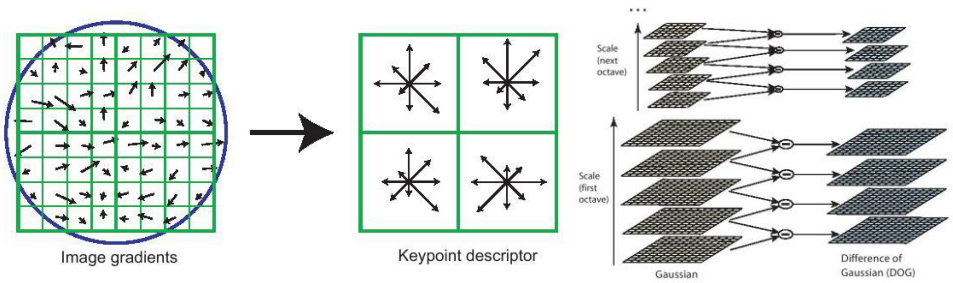
Deep Learning = Learning Features

- Traditional model of pattern recognition
  - Fixed features + trainable classifier

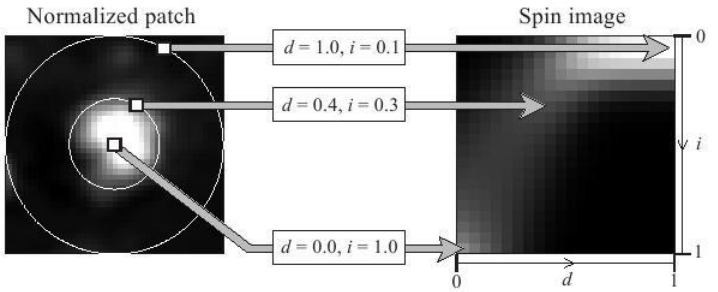
- Deep Learning
  - Trainable features + trainable classifier



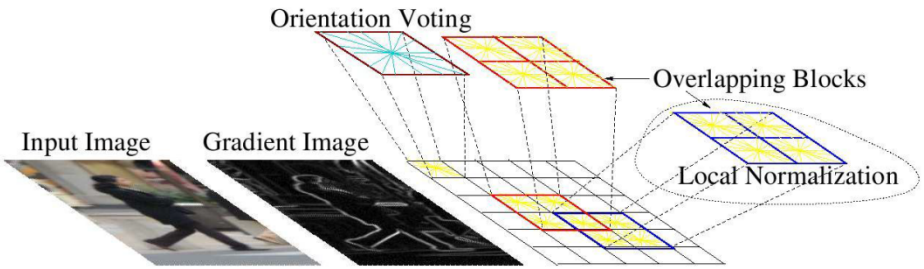
Computer vision features



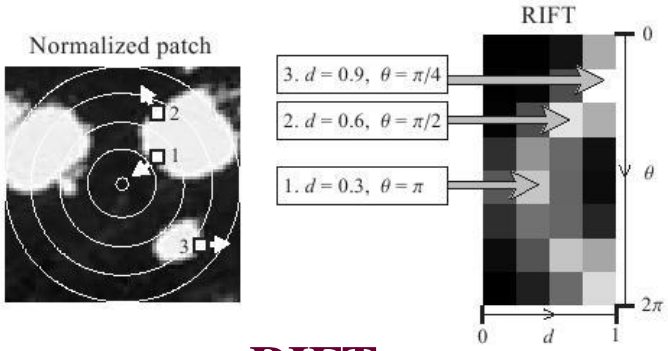
SIFT



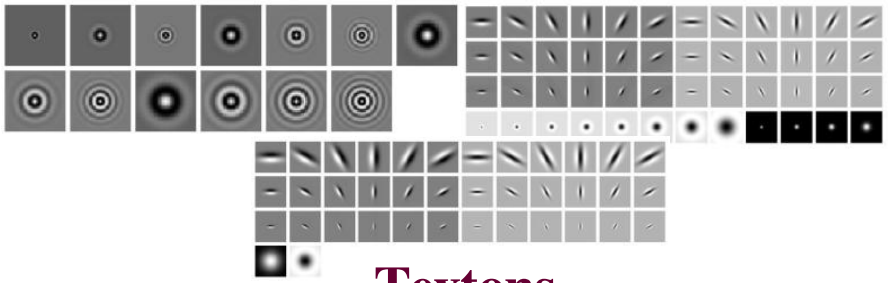
Spin image



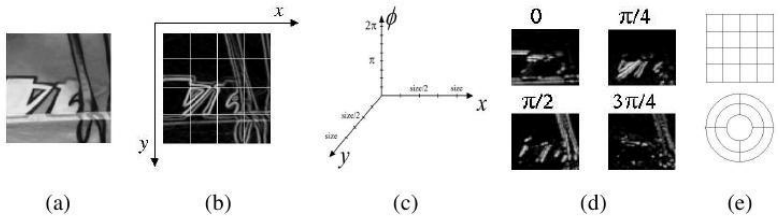
HoG



RIFT

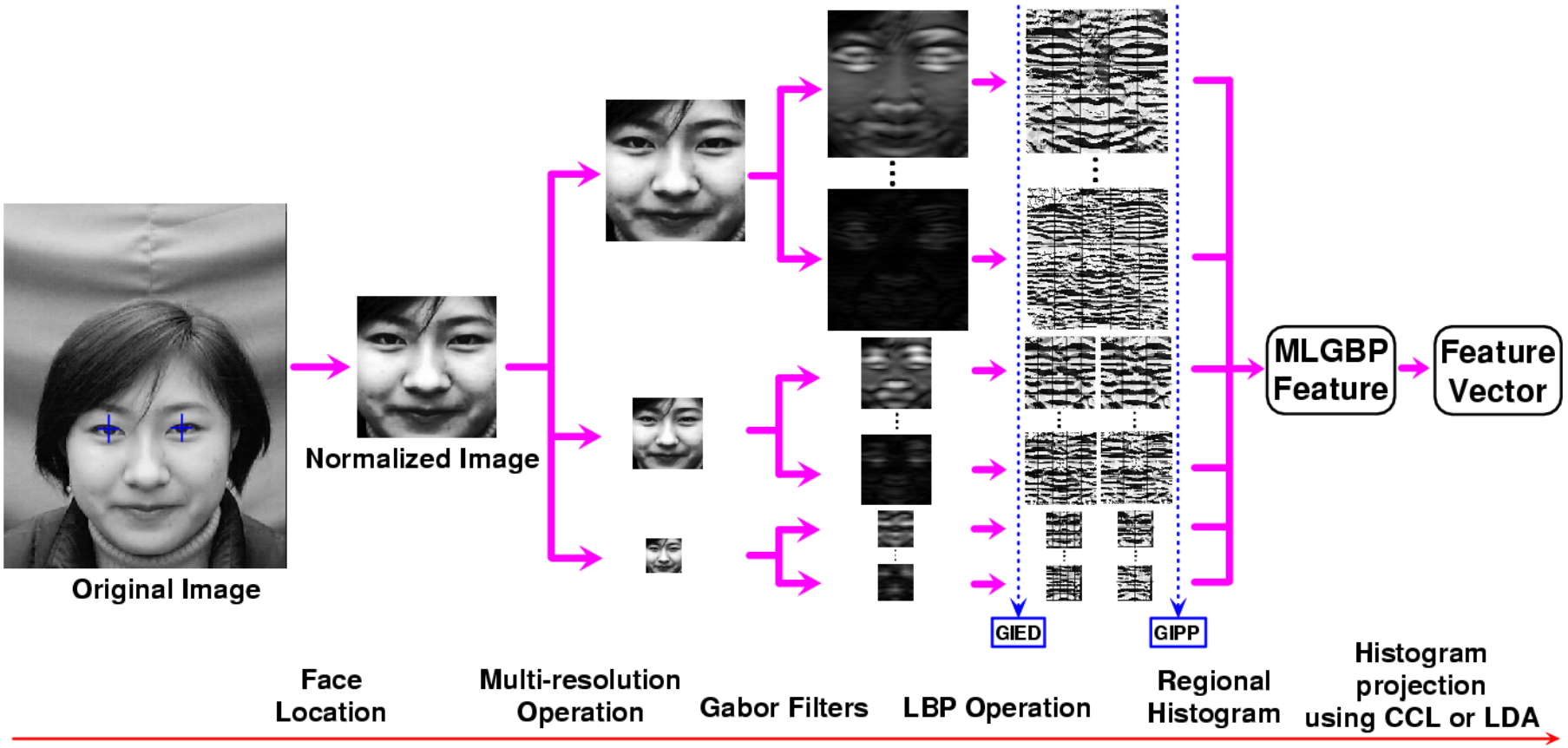


Textons

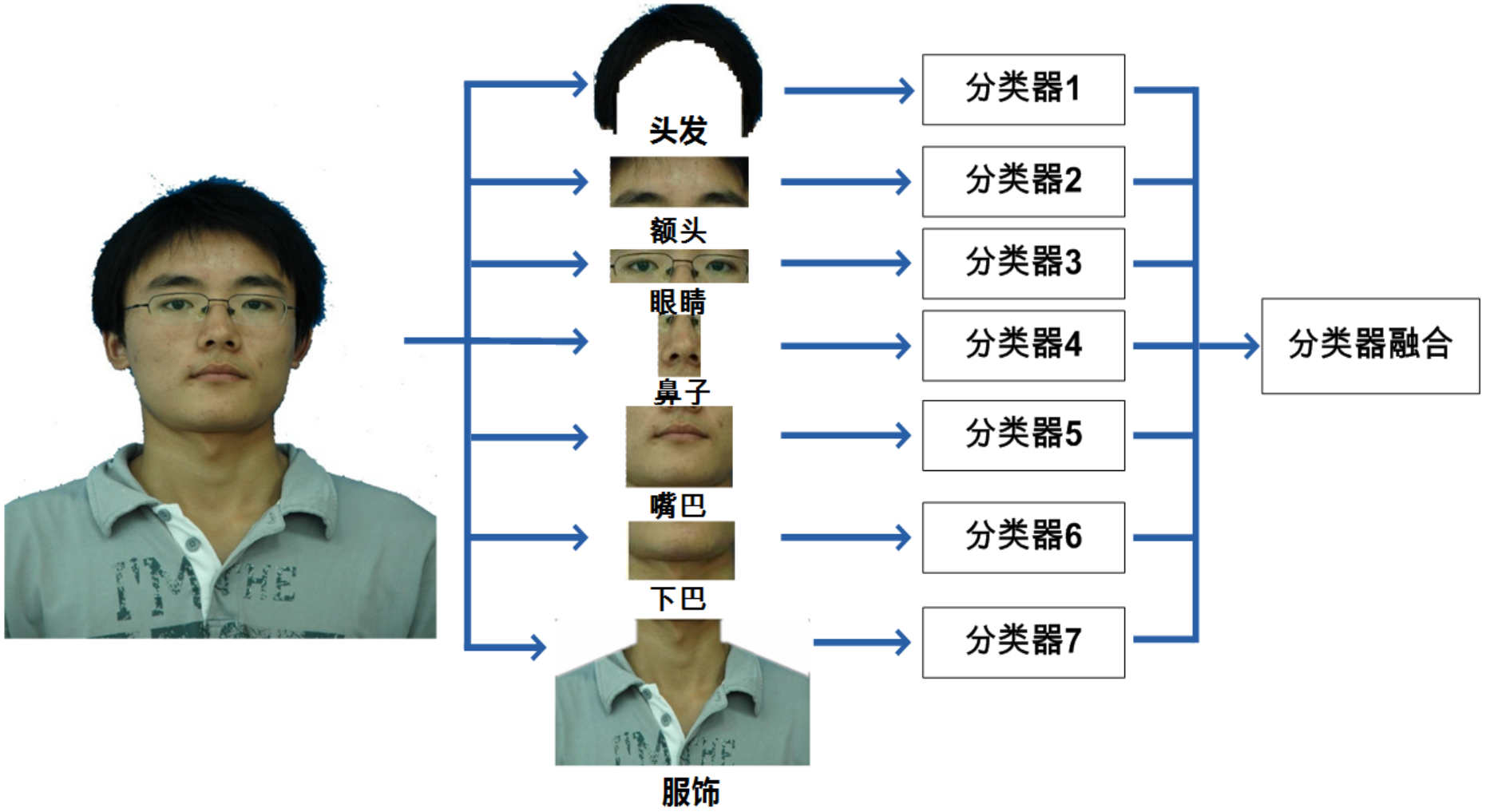


GLOH

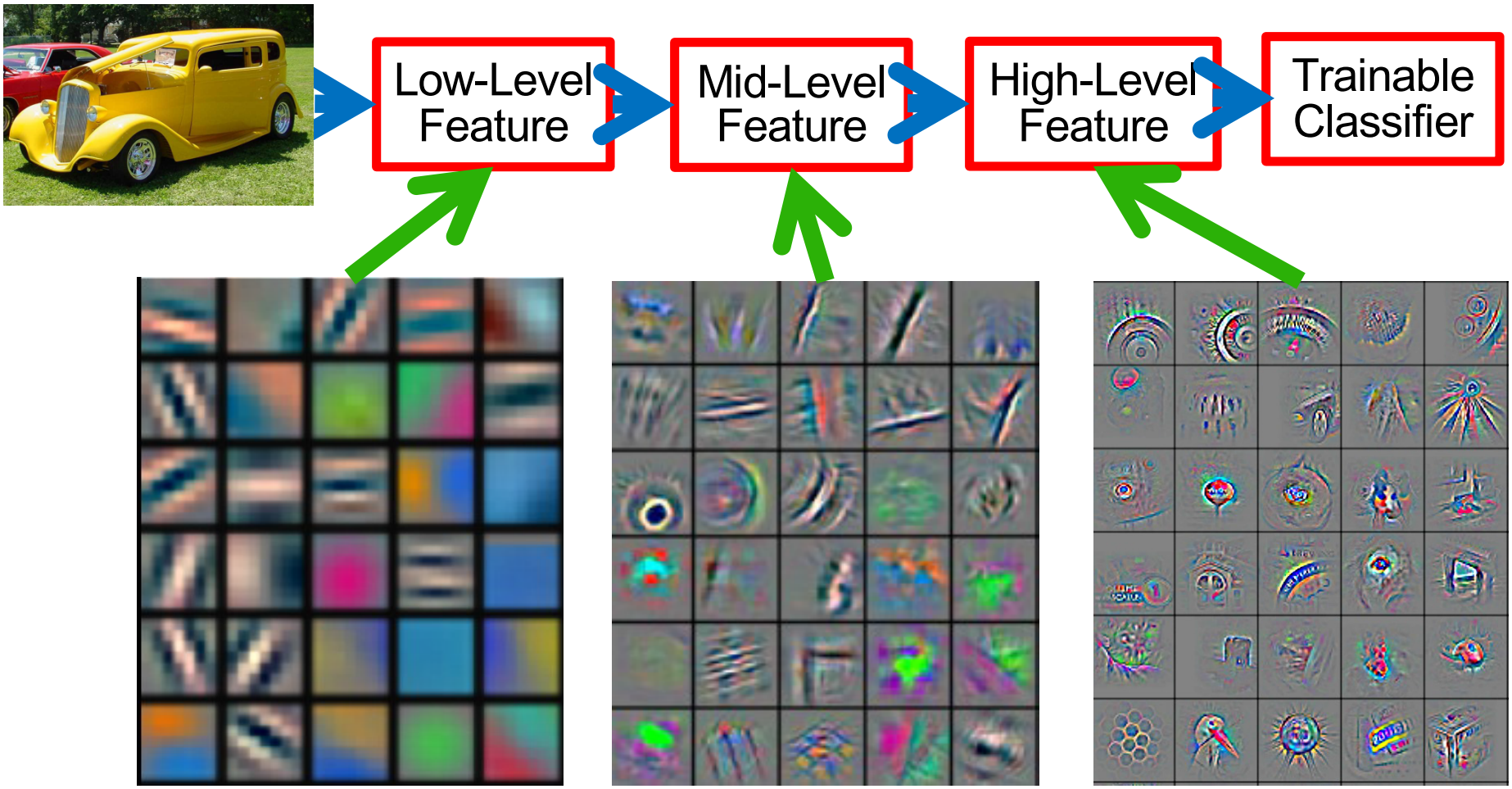
人脸特征（ MLGBP）


Original Image
Normalized Image
GIED
GIPP
MLGBP Feature
Feature Vector
Face Location
Multi-resolution Operation
Gabor Filters
LBP Operation
Regional Histogram
Histogram projection using CCL or LDA

多特征性别分类框架



Deep Learning = Learning Hierarchical Representations

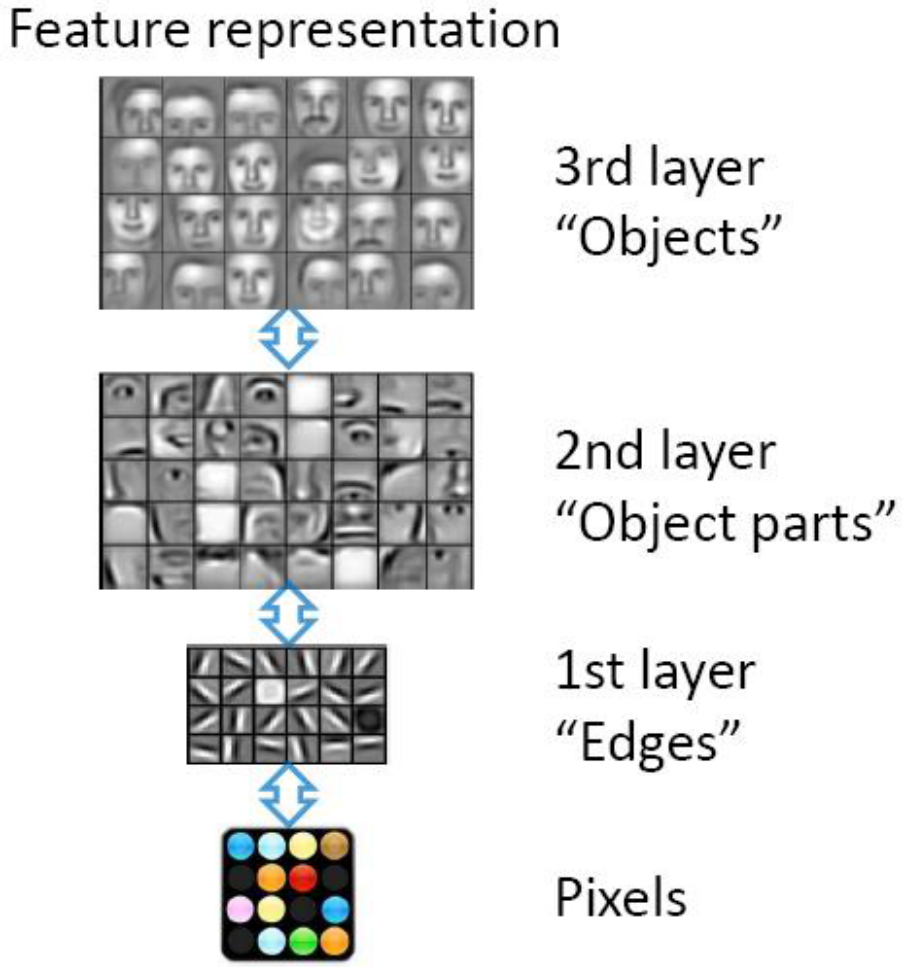


Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

Learning Feature Hierarchy

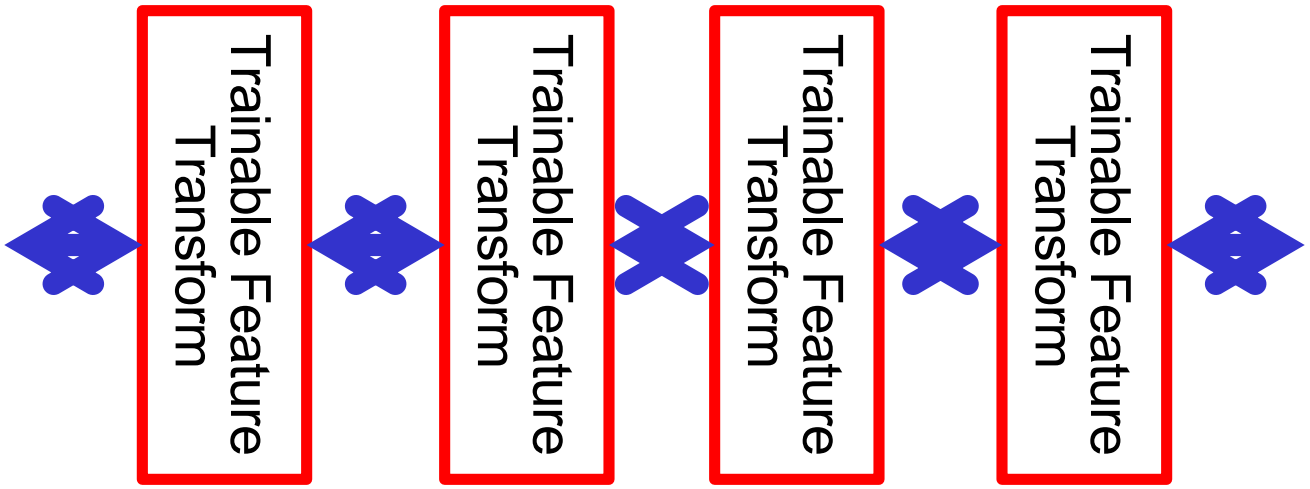
Deep Learning

- Deep architectures can be representationally efficient.
- Natural progression from low level to high level structures.
- Can share the lower-level representations for multiple tasks



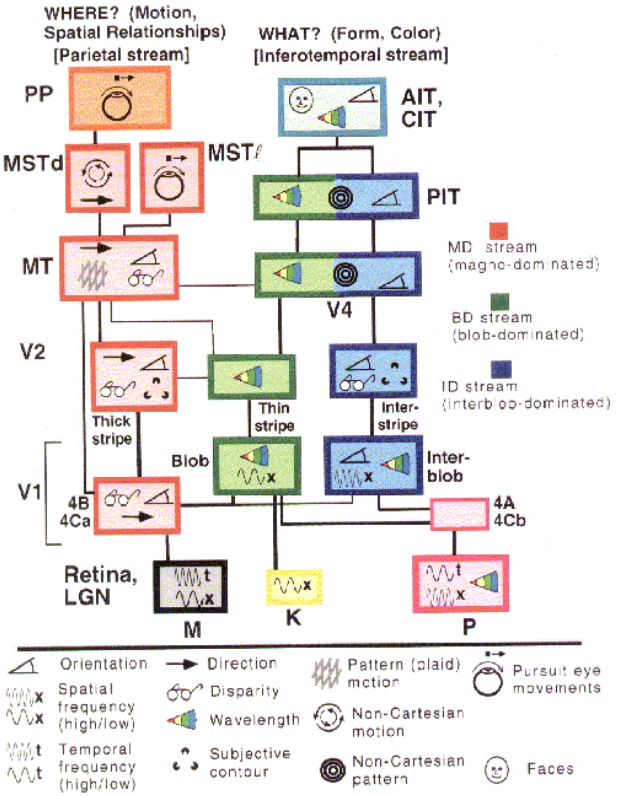
Trainable Feature Hierarchy

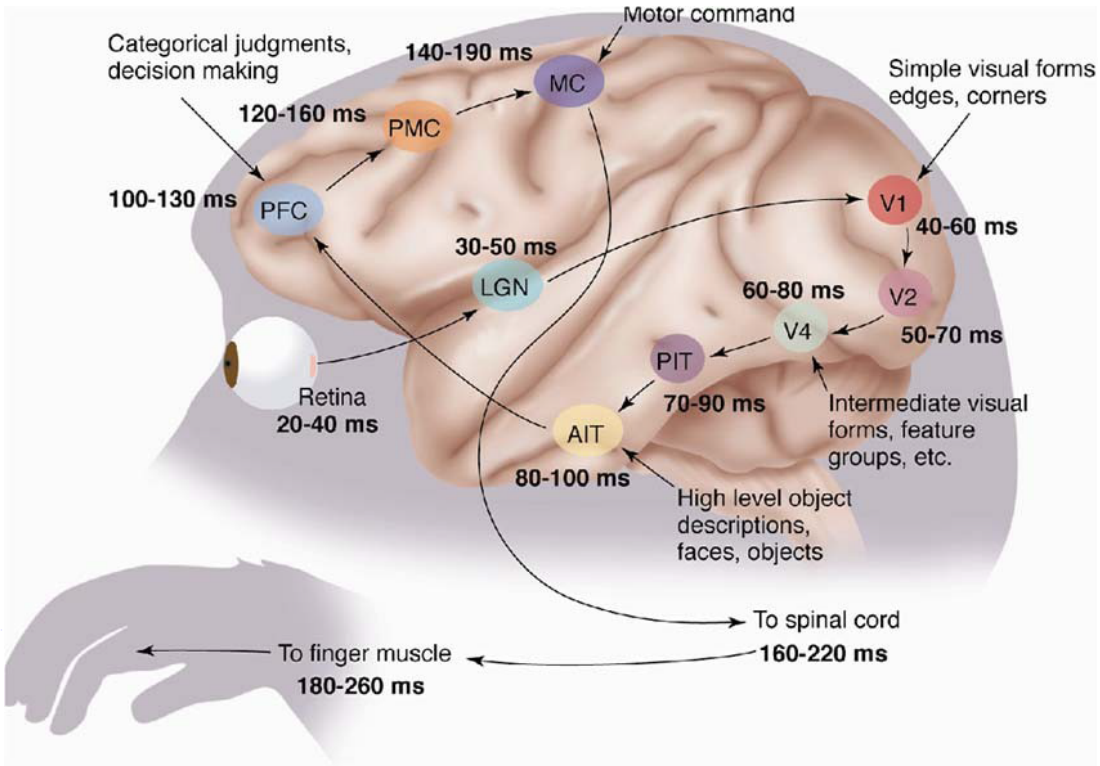
- Hierarchy of representations with increasing level of abstraction
- Each stage is a kind of trainable feature transform
- Image recognition
  - Pixel → edge → texton → motif → part → object
- Text
  - Character → word → word group → clause → sentence → story
- Speech
  - Sample → spectral band → sound → … → phone → phoneme →word



The Mammalian Visual cortex is Hierarchical

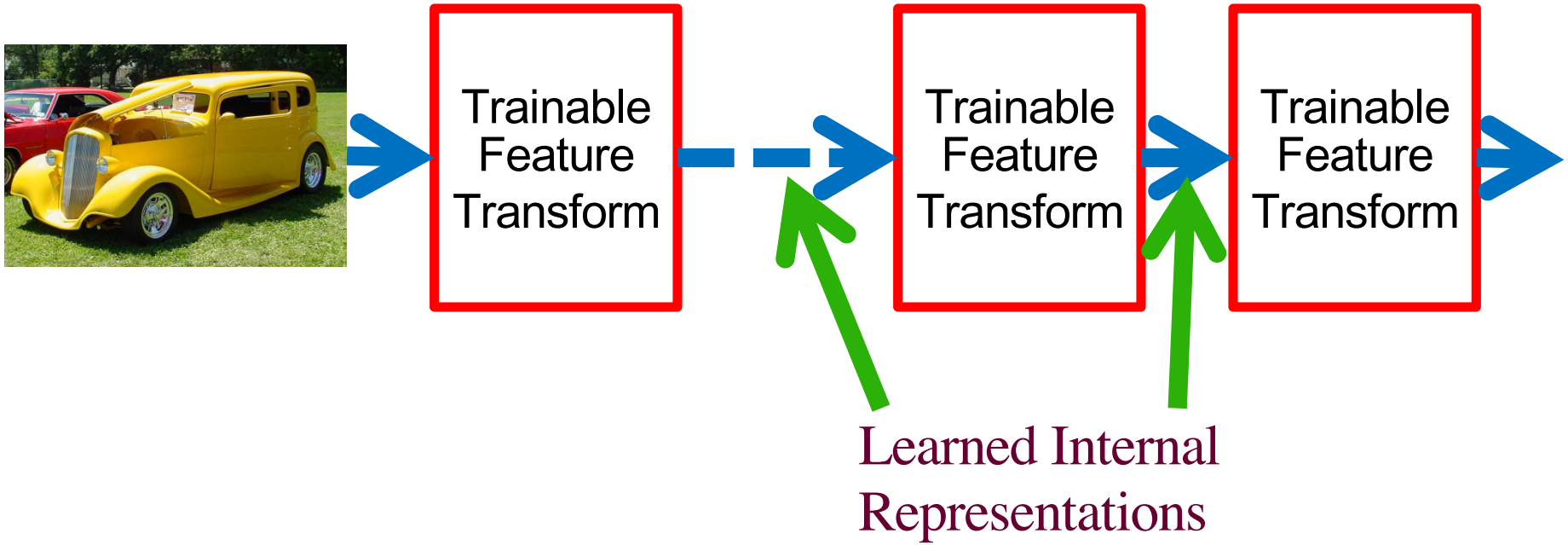
- The ventral (recognition) pathway in the visual cortex has multiple stages
- Retina – LGN – V1 – V2 – V4 – PIT – AIT ....
- Lots of intermediate representations





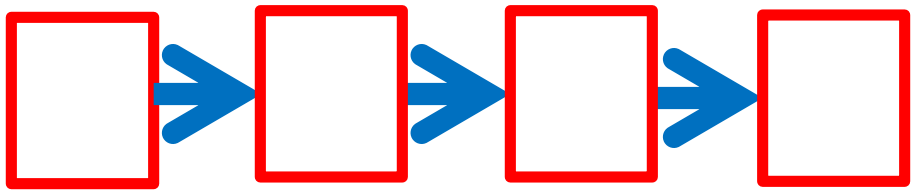
Trainable Feature Hierarchies

- A hierarchy of trainable feature
  - Each module transforms its input representation into a higher-level one.
  - High-level features are more global and more invariant
  - Low-level features are shared among categories

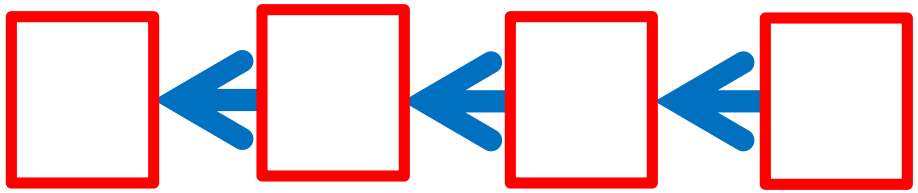


Three Types of Deep Architectures

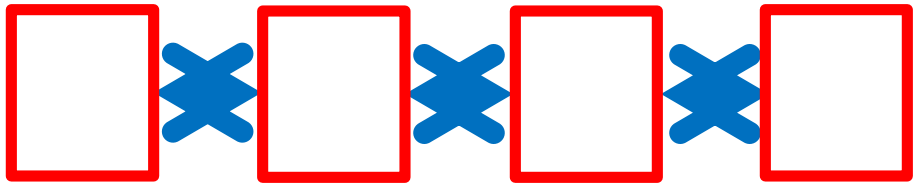
- Feed-Forward: multilayer neural nets, convolutional nets



- Feed-Back: Stacked Sparse Coding, Deconvolutional Nets



- Bi-Drectional: Deep Boltzmann Machines, Stacked Auto-Encoders



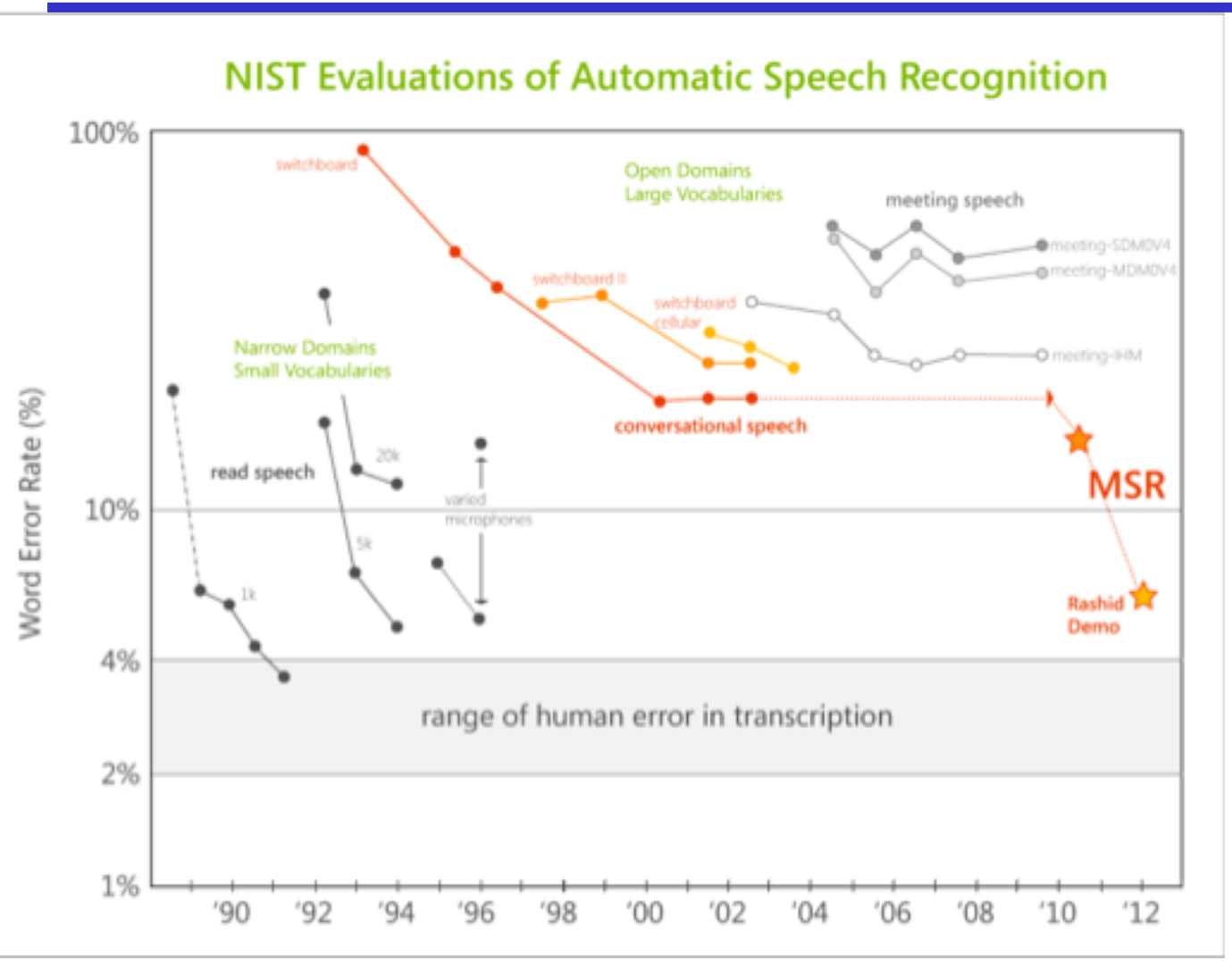
Three Types of Training Protocols

- Purely Supervised
  - Initialize parameters randomly
  - Train in supervised mode
  - Used in most practical systems for speech and image recognition
- Unsupervised, layerwise + supervised classifier on top
  - Train each layer unsupervised, one after the other
  - Train a supervised classifier on top, keeping the other layers fixed
  - Good when very few labeled samples are available
- Unsupervised, layerwise + global supervised fine-tuning
  - Train each layer unsupervised, one after the other
  - Add a classifier layer, and retrain the whole thing supervised
  - Good when label set is poor

Why Deep Learning

- Biological Plausibility – e.g. Visual Cortex
- Learning features, not just handcrafting them
- Unsupervised feature learning
- Learning multiple levels of representation
- Highly varying functions can be efficiently represented with deep architectures
  - Less weights/parameters to update than a less efficient shallow representation
- Sub-features created in deep architecture can potentially be shared between multiple tasks
  - Type of Transfer/Multi-task learning

Speech Recognition Progress: NIST evaluations



After no improvement for 10+ years by the research community…

…MSR used deep learning to reduce error rate from **~23%** to **~13%** on SWBD (and under 7% for Rick Rashid's demo)!

Deep Learning Today

- AlphaGo
- Speech Recognition
  - A few long-standing performance records were broken with deep learning methods
  - Microsoft and Google have both deployed DL-based speech recognition system in their products
  - Microsoft, Google, IBM, Nuance, AT&T, and all the major academic and industrial players in speech recognition have projects on deep learning
- Computer Vision
  - Feature engineering is the bread-and-butter of a large portion of the CV community, which creates some resistance to feature learning
  - But the record holders on ImageNet and Semantic Segmentation are convolutional nets

Convolutional Neural Networks

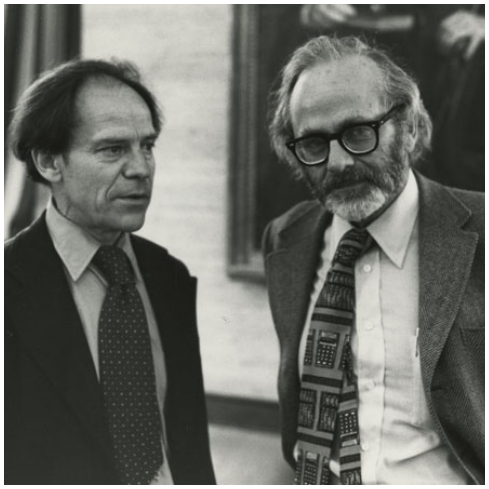
卷积神经网络

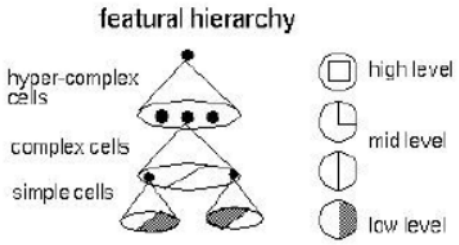
- CNN的生物学基础
- CNN的历史
- CNN的结构元件
- CNN的结构：LeNet-5
- CNN VS 全连接网络

2
2

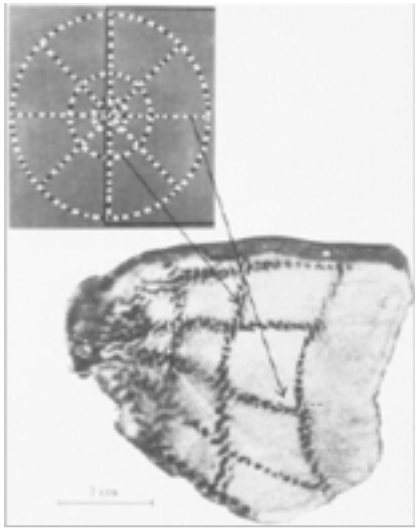
CNN的生物学基础:Hubel & Wiesel

- 猫的大脑中第一层的视觉神经元（V1），只处理局部视觉中的简单基础结构信息（如某一方向的直线、点），且一种神经元只对一种特定的基础结构做出反应。
- 而且V1神经元是会保留拓扑结构信息的，即相邻的神经元的感受野在图像中也相邻。
- 视觉神经元是有层次的。



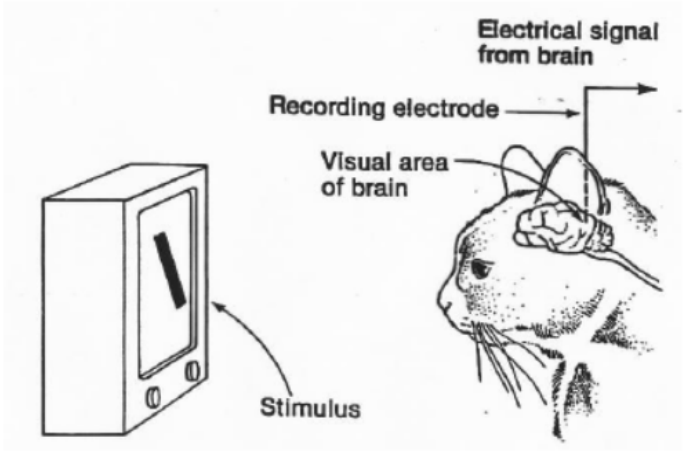


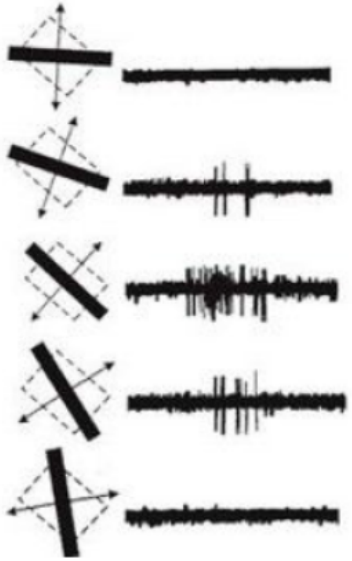
Hubel, David H., and Torsten N. Wiesel. "Receptive fields, binocular interaction and functional architecture in the cat's visual cortex." *The Journal of physiology* 160.1 (1962): 106-154.

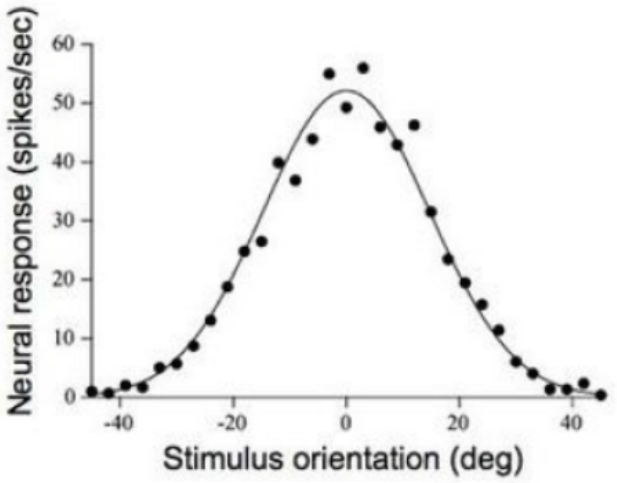


CNN的生物学基础

- 猫的大脑中第一层的视觉神经元（V1），只处理局部视觉中的简单基础结构信息（如某一方向的直线、点），且一种神经元只对一种特定的基础结构做出反应。

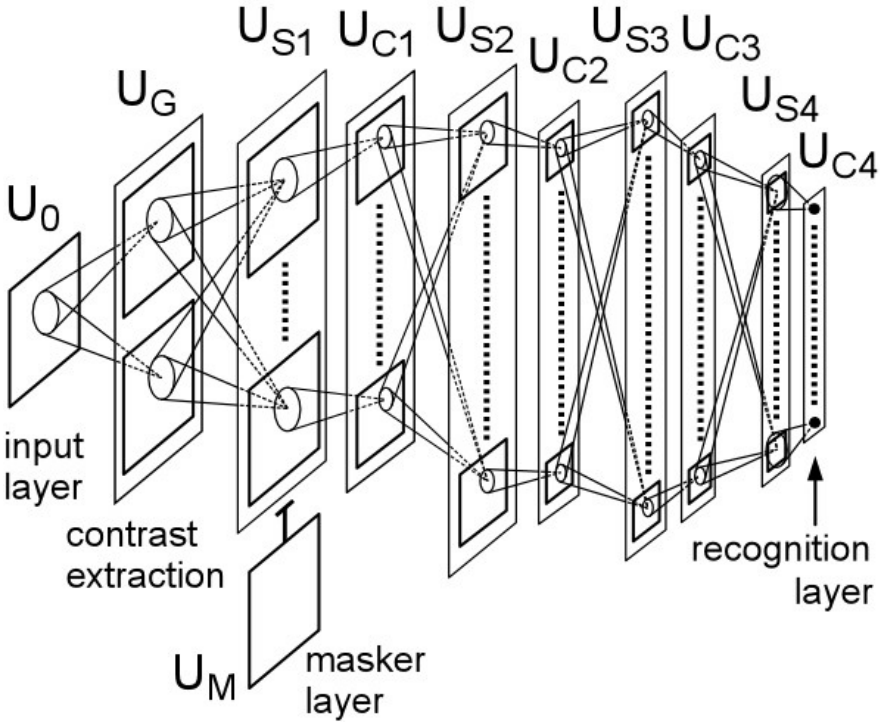






CNN的历史

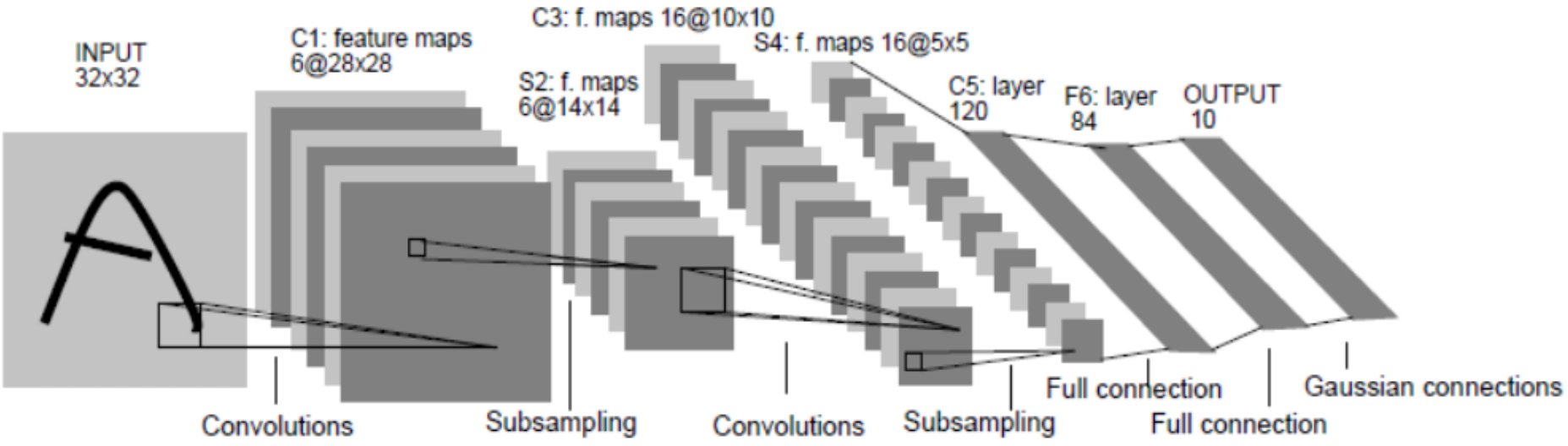
- Fukushima 1980: Neurocognitron
  ("三明治" 结构, 局部感受野, 池化层)



Prof. Fukushima

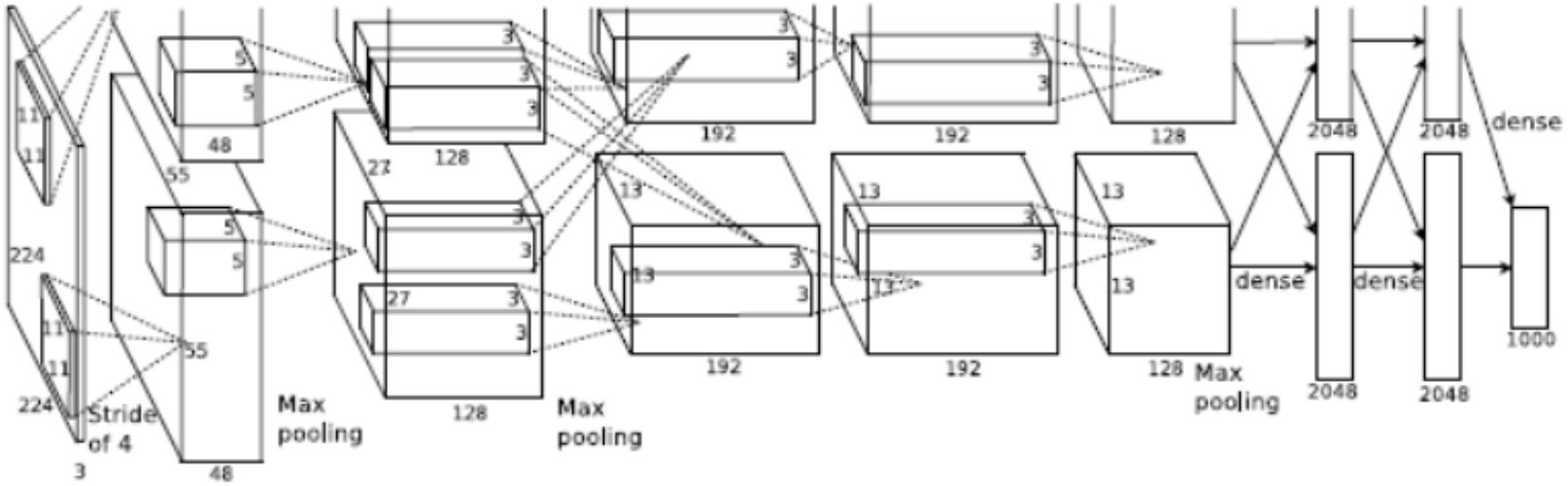
CNN的历史

- Lecun 1998: LeNet-5 (工业界的应用：手写数字识别)



CNN的历史

- Krizhevsky, Sutskever, Hinton, 2012: AlexNet (IMAGENET数据集上分类准确率远高于传统方法)



CNN的结构元件

- 卷积的定义
- 卷积层
- 激活层
- 池化层
- 全连接层
- 损失函数

卷积的定义

- 在泛函分析中，卷积(Convolution)是通过两个函数f 和g生成第三个函数的一种数学算子，表征函数f 与g经过翻转和平移的重叠部分的面积。如果将参加卷积的一个函数看作区间的指示函数，卷积还可以被看作是“滑动平均”的推广。
- 卷积是在信号与线性系统的基础上或背景中出现的，讨论的就是信号经过一个线性系统以后发生的变化（就是输入，输出和所经过的所谓系统，这三者之间的数学关系）。
- 实际上，线性系统的传递函数和输入信号，在数学上的形式就是所谓的卷积关系(一维卷积)。而数字图像处理中的卷积关系是二维卷积。

离散一维卷积

对于两个长度为m和n的序列$f(i)$和$g(j)$，

$$h(i)=f(i)*g(i)=\sum_{j}f(j)g(i-j)$$

给出长度为$N=m+n-1$的输出序列。

其矩阵计算形式为

$$\mathbf{h}=\mathbf{g}\bullet\mathbf{f}=\begin{bmatrix}\mathbf{g}_p(1) & \mathbf{g}_p(\mathrm{N}) & \cdots & \mathbf{g}_p(2)\\ \mathbf{g}_p(2) & \mathbf{g}_p(1) & \cdots & \mathbf{g}_p(3)\\ \vdots & \vdots & \vdots & \vdots\\ \mathbf{g}_p(\mathrm{N}) & \mathbf{g}_p(\mathrm{N}-1) & \cdots & \mathbf{g}_p(1)\end{bmatrix}\begin{bmatrix}\mathbf{f}_p(1)\\ \mathbf{f}_p(2)\\ \vdots\\ \mathbf{f}_p(\mathrm{N})\end{bmatrix}$$

离散二维卷积

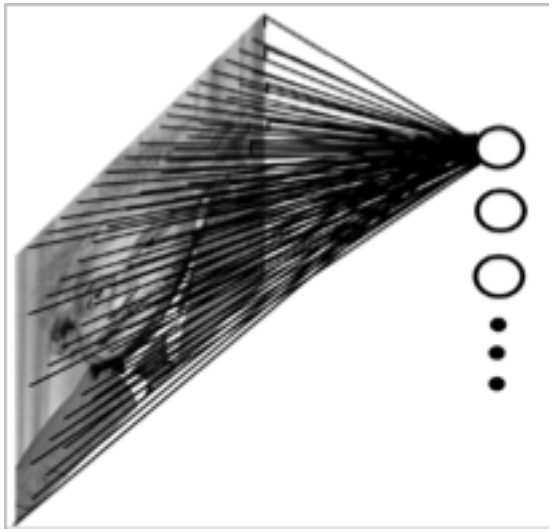
- 将卷积形式从一维推广到二维，我们得到离散二维卷积的形式如下。二维卷积一般被应用在图像预处理之中(平滑，滤波等)。此时，F为卷积函数(又称算子，卷积核)，G为原始图像，H为经过卷积后的图像。

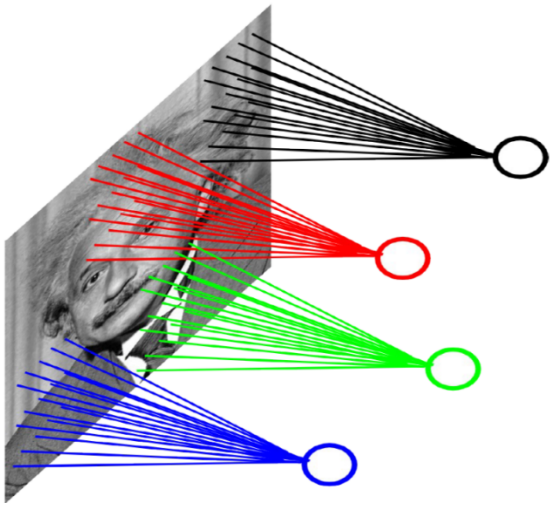
$$H = F * G$$

$$H(i,j) = \sum_m \sum_n F(m,n) G(i-m, j-n)$$

Locally connected and Share Weights

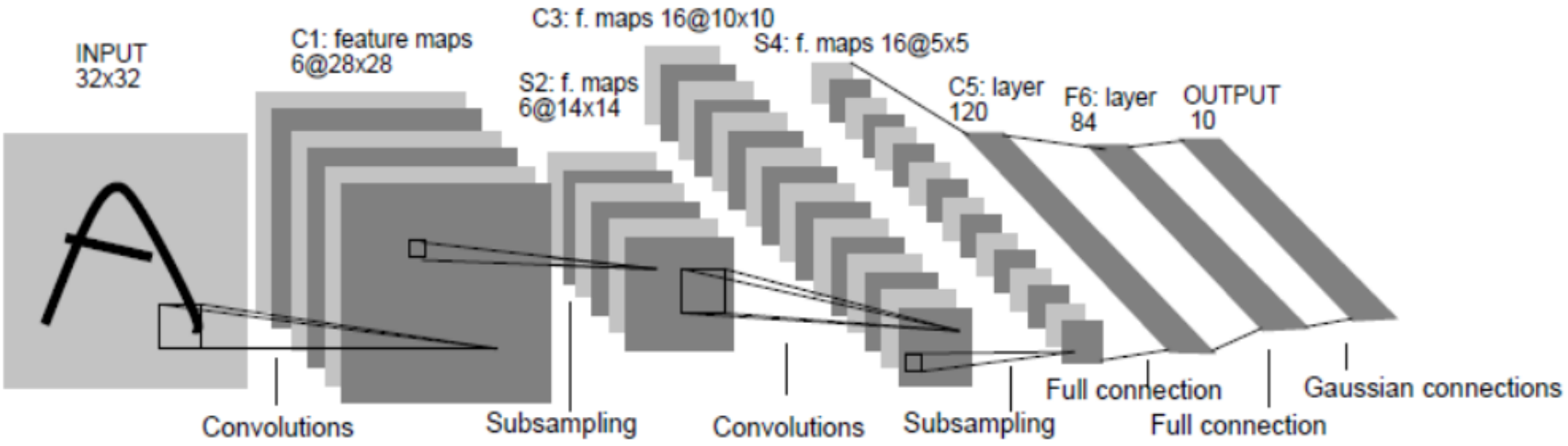
- Example: 1000*1000 image
  - Fully-connected, 1000,000 hidden units
  - 1000 *1000*1000,000= 10^{12} parameters
  - Locally-connected, 1000,000 hidden units 10x10 fields = 10^8 parameters
  - Local connections capture local dependencies
  - 1000,000 hidden units share the same weights
  - The number of parameters are decreased from 10^8 to 100!





CNN的典型结构

- 每层卷积层之后都接激活层
- 若干层卷积层之后接池化层
- 最后使用全连接层+损失函数
- LeNet, AlexNet, VGGNet都符合这一范式

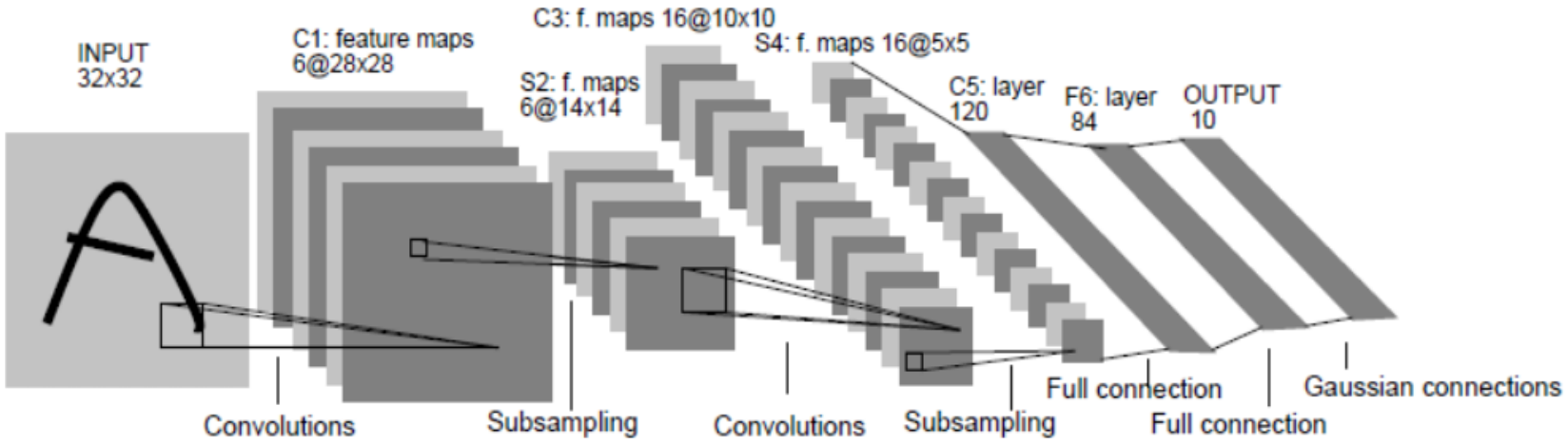


LeNet-5

- 早在1989年，Yann LeCun (纽约大学;Facebook) 和他的同事们就发表了卷积神经网络（Convolution Neural Networks， 简称CNN）的工作。
- CNN是一种带有卷积结构的深度神经网络，通常至少有两个非线性可训练的卷积层，两个非线性的固定卷积层（又叫Pooling Laye）和一个全连接层，一共至少5个隐含层。
- LeNet-5就是一个典型的CNN结构。

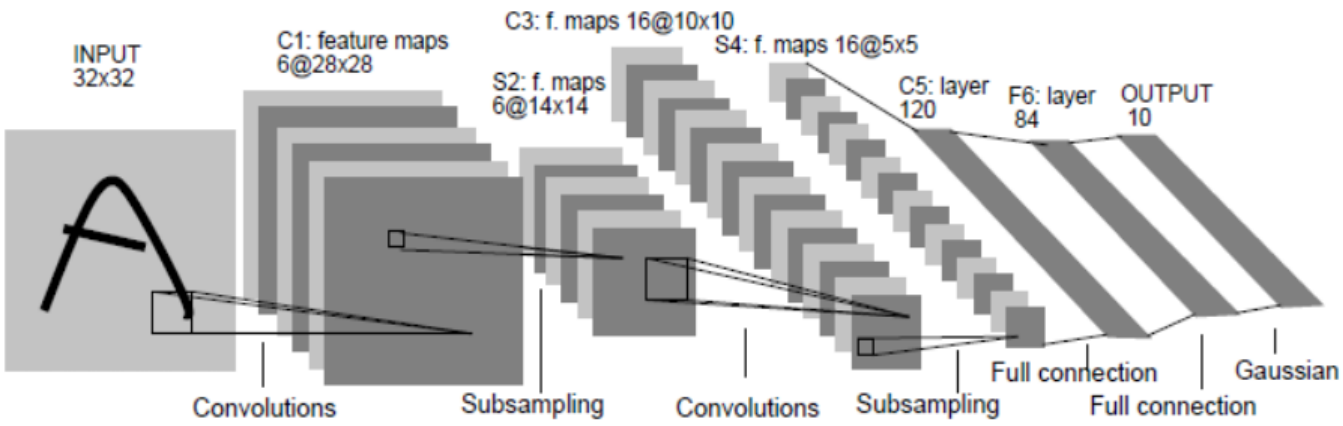
LeNet-5的功能

- LeNet-5是应用在手写数字识别任务中的。
- **输入：**32*32的手写字体图片，这些手写字体包含0~9数字，也就是相当于10个类别的图片。
- **输出：**分类结果，0~9之间的一个数。
- 因此我们可以知道，这是一个多分类问题，总共有十个类，因此神经网络的最后输出层必然是SoftMax问题，然后神经元的个数是10个。



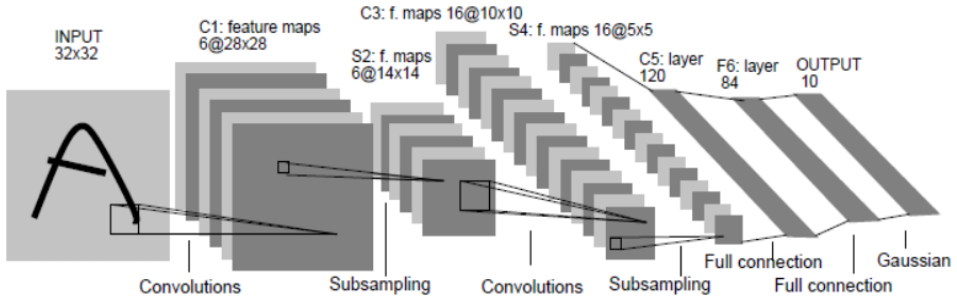
LeNet-5: C1层

- 输入图片大小： 32*32
- 卷积窗大小： 5*5
- 卷积窗种类： 6
- 输出特征图数量： 6
- 输出特征图大小： 28*28 (32-5+1)
- 神经元数量： 4707 [(28*28)*6)]
- 连接数： 12304 [(5*5+1)*6]*(28*28)
- 可训练参数： 156 [(5*5+1)*6]



LeNet-5: S2层

- 输入图片大小：　(28*28)*6
- 卷积窗大小：　2*2
- 卷积窗种类：　6



- 输出亚采样图数量：6
- 输出亚采样图大小：(14*14)*6
- 神经元数量：　1176　(14*14)*6
- 连接数：　5880　(4+1)*(14*14)*6
- 可训练参数：　12　(6*2)
- 每个单元的四个输入相加，乘以一个可训练参数，

  再加上一个可训练偏置，结果通过Sigmoid函数计算

LeNet-5：C3层

输入图片大小：　(14*14)*6

卷积窗大小：　5*5

卷积窗种类：　15

输出特征图数量：　16

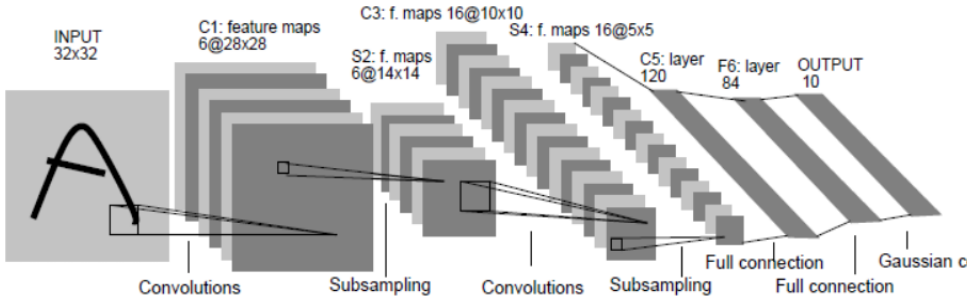
输出特征图大小：　10*10　　(14-5+1)

神经元数量：　1600　[(10*10)*16)]

连接数：　151600 [(60+16)*25]*(10*10)

（部分连接）

可训练参数：　1516　[(60+16)*25]



LeNet-5：S4层

输入图片大小：(10*10)*16

卷积窗大小：2*2

卷积窗种类：16

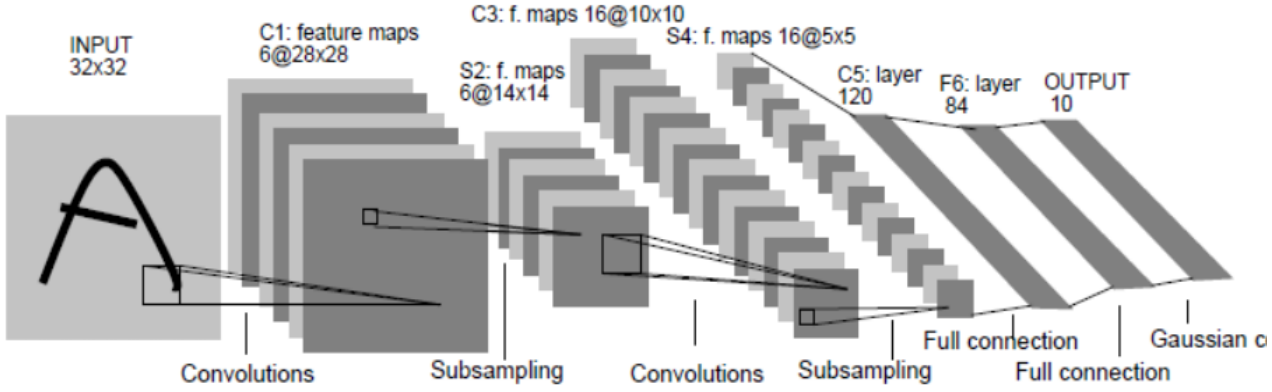
输出下采样图数量：16

输出下采样图大小：(5*5)*16

神经元数量：(5*5)*16=400

连接数：(4+1)*(5*5)*16=2000

可训练参数：(16*2)=32



LeNet-5：C5层（NN）

输入图片大小：　(5*5)*16

卷积窗大小：　5*5

卷积窗种类：　120

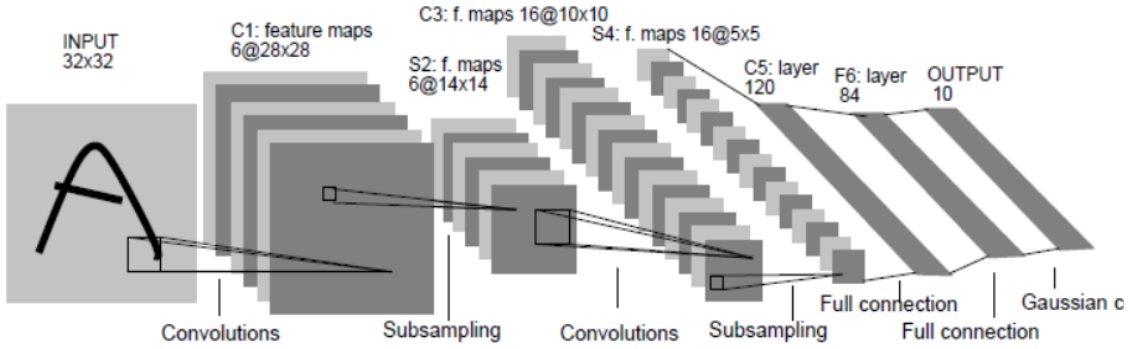
输出特征图数量：　120

输出特征图大小：　1*1　(5-5+1)

神经元数量：　120　(1*120)

连接数：　（16*5*5+1）*120(全连接）=48120

可训练参数：　（16*5*5+1）*120=48120



LeNet-5：F6层（NN）

输入图片大小：　(1*1)*120

卷积窗大小：　1*1

卷积窗种类：　84

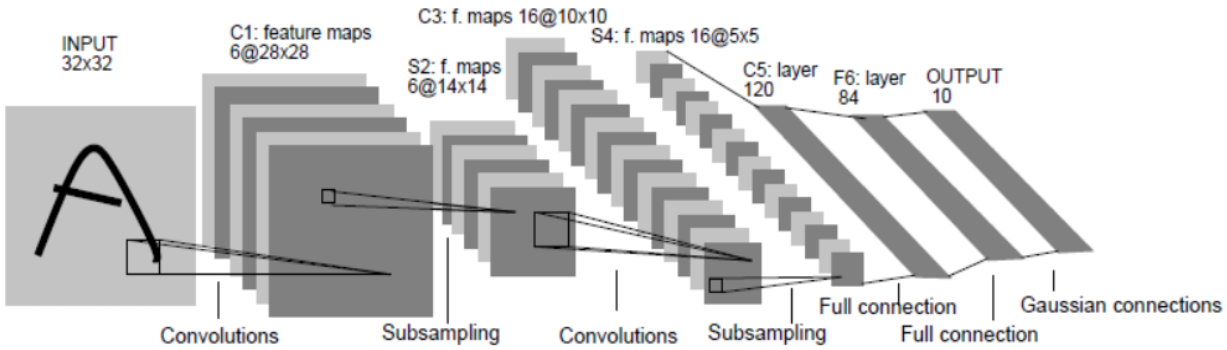
输出特征图数量：　84

输出特征图大小：　1

神经元数量：　84

连接数：　（120+1）*84=10164（全连接）

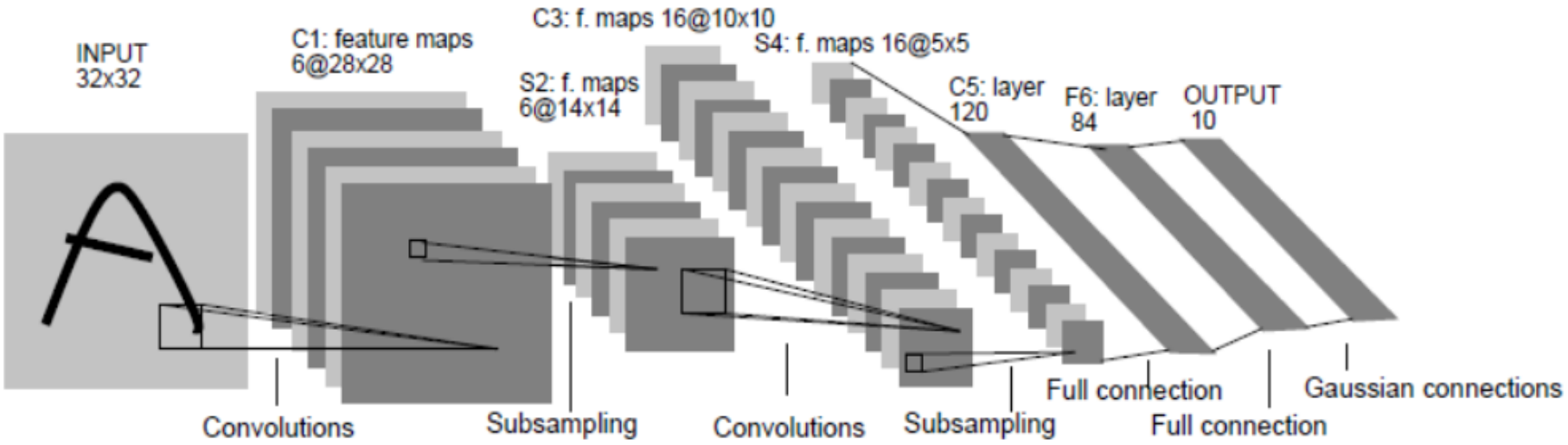
可训练参数：　（120+1）*84=10164



LeNet-5：OUTPUT层（NN）

输入图片大小：　　1*84

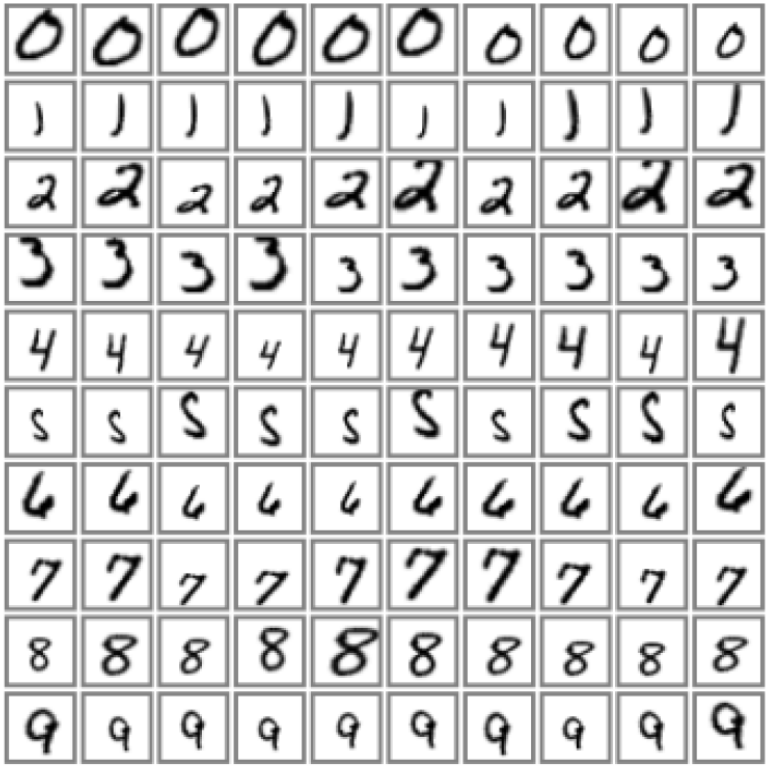
输出特征图数量：　1*10



LeNet-5：MINIST Dataset

60,000 original datasets

Test error: 0.95%



540,000 artificial distortions

+ 60,000 original

Test error: 0.8%

4
3

LeNet-5：Misclassified Examples

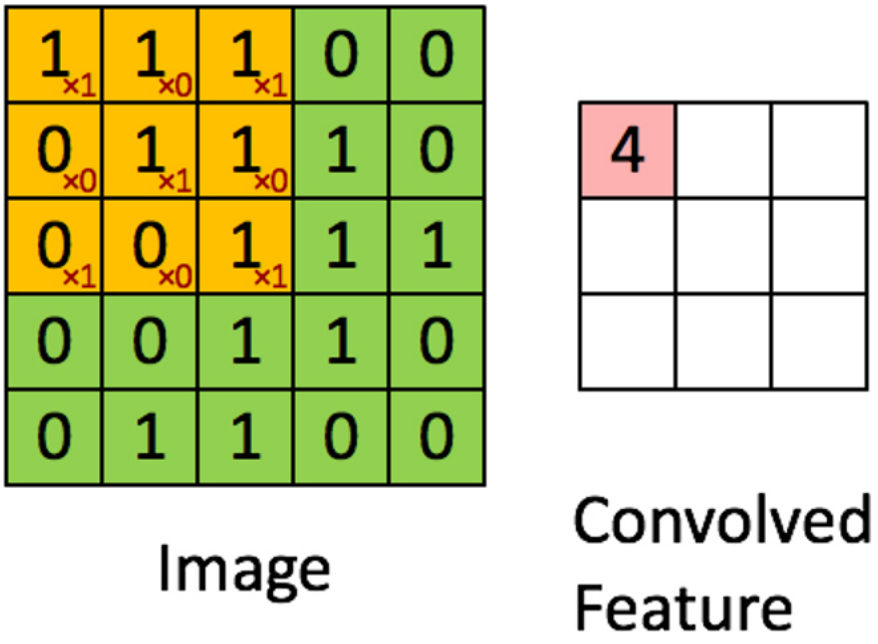
CNN的结构元件：卷积层

- O = convolution(I , filters)
  - O = {convolution(I , filter), for filter in filters}
  - 每个滤波器生成输出的一个通道，滤波器数=输出通道数
- 输入I：H*W*C_1，其中H,W分别是输入的高度和宽度，C_1是输入的通道数（深度）。
- 滤波器组filters：C_2*滤波器，其中C_2是滤波器的数目。
  - 滤波器：F*F*{C}+bias+P+S，其中F是局部感受野的边长， bias是偏移量 (通常为1)，P是填充宽度，S是步长，{C}是输入通道的组合，通常{C}={1,2,…,C_1}
- 输出O：H'*W'*C_2，由输入和滤波器组唯一确定。
  - H'=(H–F+2*P)/S+1，W'=(W–F2*P)/S+1，C_2=C_2

CNN的结构元件：卷积层

- 把滤波器想象成窗口
- 一次卷积操作是窗口内部的输入与滤波器的卷积（乘积的和）
- 窗口按照给定的步长S在输入上滑动，每次都进行一次卷积操作，即可得到输出的一个通道（特征图）

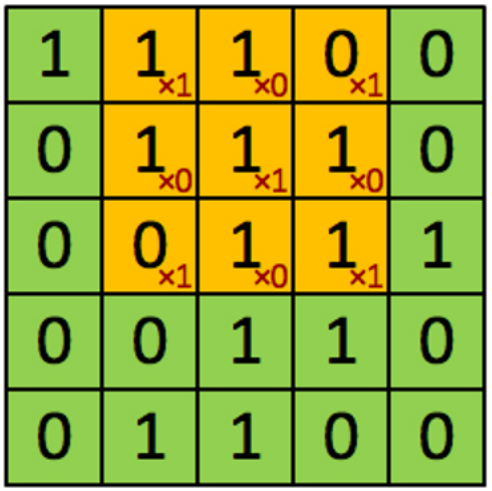
- 右图给出了H=W=5, C_1=1, C_2=1, F=3, {C}={1}, S=1, P=0 情况下的例子。

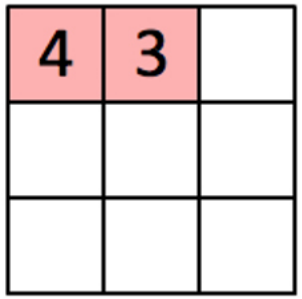


CNN的结构元件：卷积层

- 把滤波器想象成窗口
- 一次卷积操作是窗口内部的输入与滤波器的卷积（乘积的和）
- 窗口按照给定的步长S在输入上滑动，每次都进行一次卷积操作，即可得到输出的一个通道（特征图）

- 右图给出了H=W=5, C_1=1, C_2=1, F=3, {C}={1}, S=1, P=0 情况下的例子。

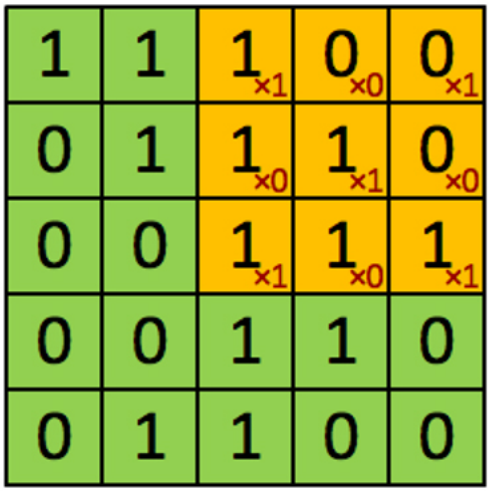




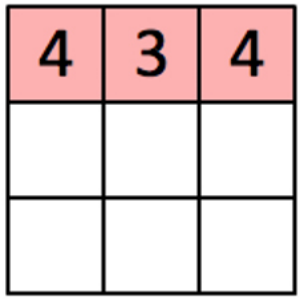
CNN的结构元件：卷积层

- 把滤波器想象成窗口
- 一次卷积操作是窗口内部的输入与滤波器的卷积（乘积的和）
- 窗口按照给定的步长S在输入上滑动，每次都进行一次卷积操作，即可得到输出的一个通道（特征图）

- 右图给出了H=W=5, C_1=1, C_2=1, F=3, {C}={1}, S=1, P=0 情况下的例子。



Image

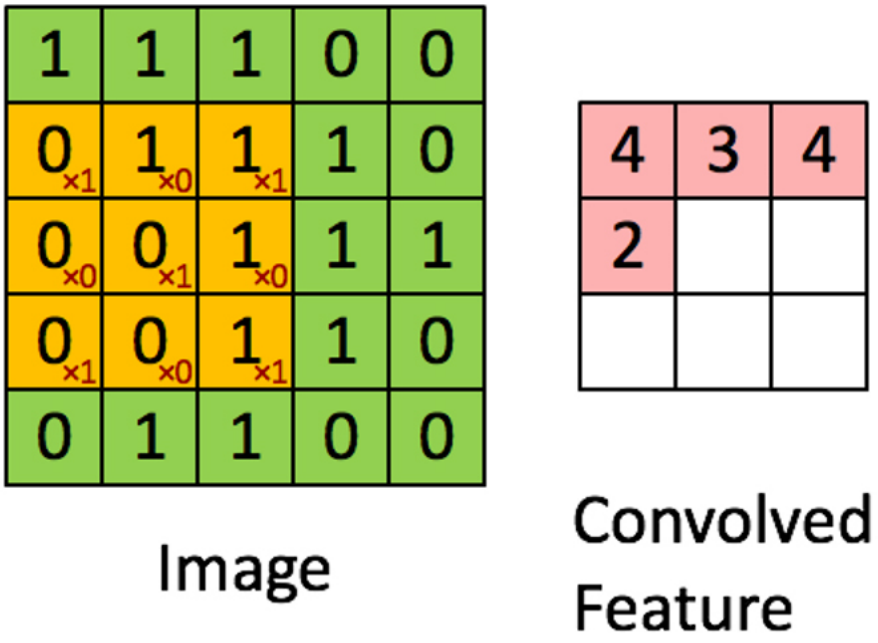


Convolved Feature

CNN的结构元件：卷积层

- 把滤波器想象成窗口
- 一次卷积操作是窗口内部的输入与滤波器的卷积（乘积的和）
- 窗口按照给定的步长S在输入上滑动，每次都进行一次卷积操作，即可得到输出的一个通道（特征图）

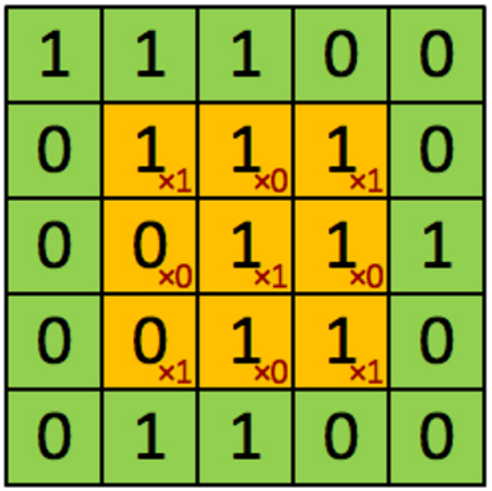
- 右图给出了H=W=5, C_1=1, C_2=1, F=3, {C}={1}, S=1, P=0 情况下的例子。

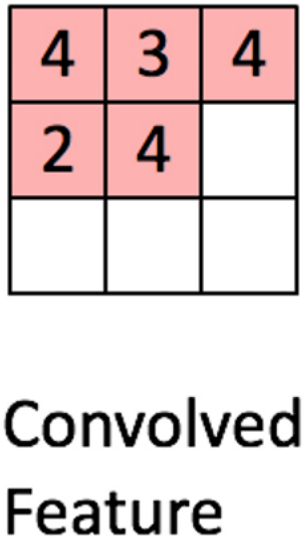


CNN的结构元件：卷积层

- 把滤波器想象成窗口
- 一次卷积操作是窗口内部的输入与滤波器的卷积（乘积的和）
- 窗口按照给定的步长S在输入上滑动，每次都进行一次卷积操作，即可得到输出的一个通道（特征图）

- 右图给出了H=W=5, C_1=1, C_2=1, F=3, {C}={1}, S=1, P=0 情况下的例子。

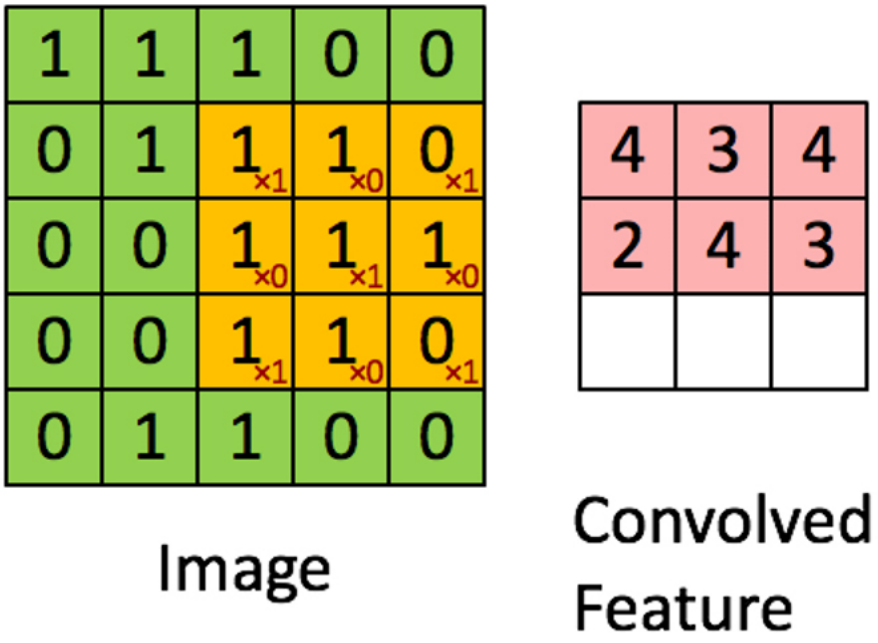




CNN的结构元件：卷积层

- 把滤波器想象成窗口
- 一次卷积操作是窗口内部的输入与滤波器的卷积（乘积的和）
- 窗口按照给定的步长S在输入上滑动，每次都进行一次卷积操作，即可得到输出的一个通道（特征图）

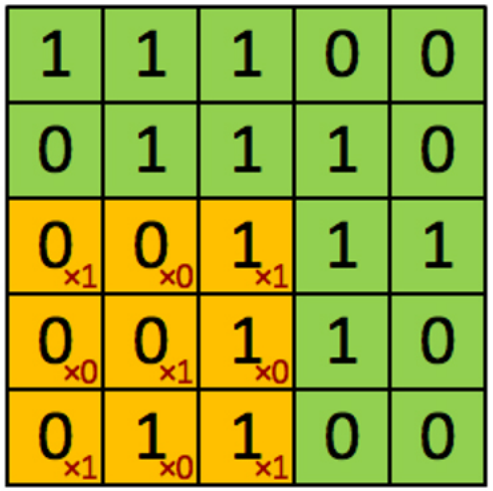
- 右图给出了H=W=5, C_1=1, C_2=1, F=3, {C}={1}, S=1, P=0 情况下的例子。



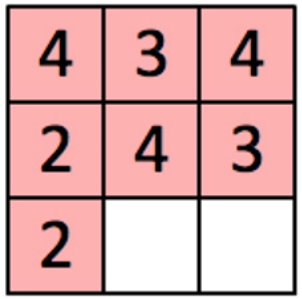
CNN的结构元件：卷积层

- 把滤波器想象成窗口
- 一次卷积操作是窗口内部的输入与滤波器的卷积（乘积的和）
- 窗口按照给定的步长S在输入上滑动，每次都进行一次卷积操作，即可得到输出的一个通道（特征图）

- 右图给出了H=W=5, C_1=1, C_2=1, F=3, {C}={1}, S=1, P=0 情况下的例子。



Image

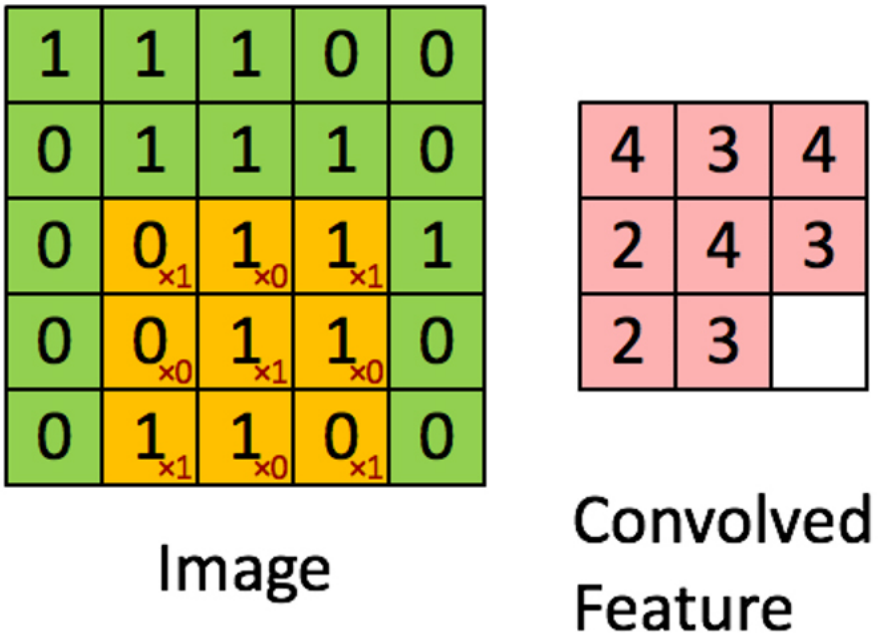


Convolved Feature

CNN的结构元件：卷积层

- 把滤波器想象成窗口
- 一次卷积操作是窗口内部的输入与滤波器的卷积（乘积的和）
- 窗口按照给定的步长S在输入上滑动，每次都进行一次卷积操作，即可得到输出的一个通道（特征图）

- 右图给出了H=W=5, C_1=1, C_2=1, F=3, {C}={1}, S=1, P=0 情况下的例子。

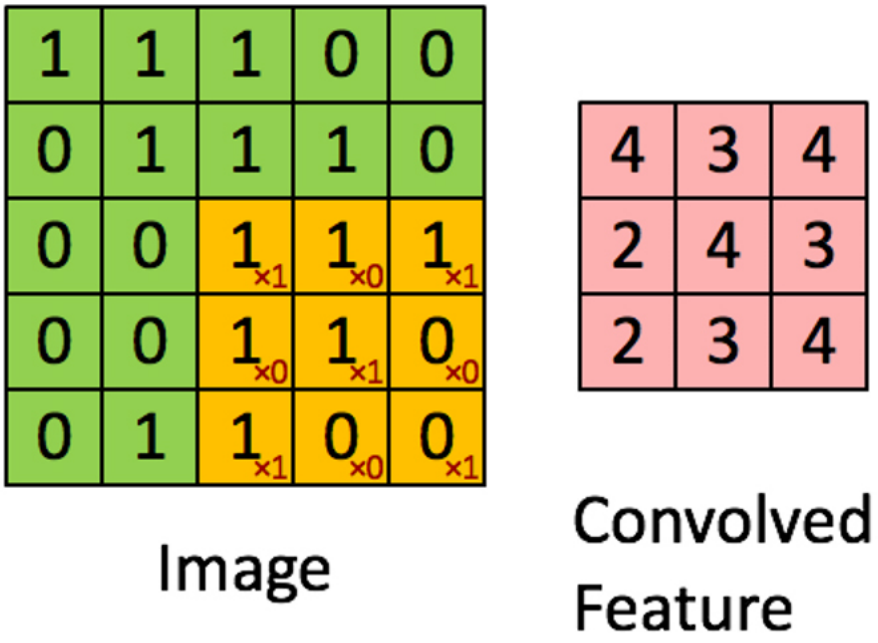


5
3

CNN的结构元件：卷积层

- 把滤波器想象成窗口
- 一次卷积操作是窗口内部的输入与滤波器的卷积（乘积的和）
- 窗口按照给定的步长S在输入上滑动，每次都进行一次卷积操作，即可得到输出的一个通道（特征图）

- 右图给出了H=W=5, C_1=1, C_2=1, F=3, {C}={1}, S=1, P=0 情况下的例子。

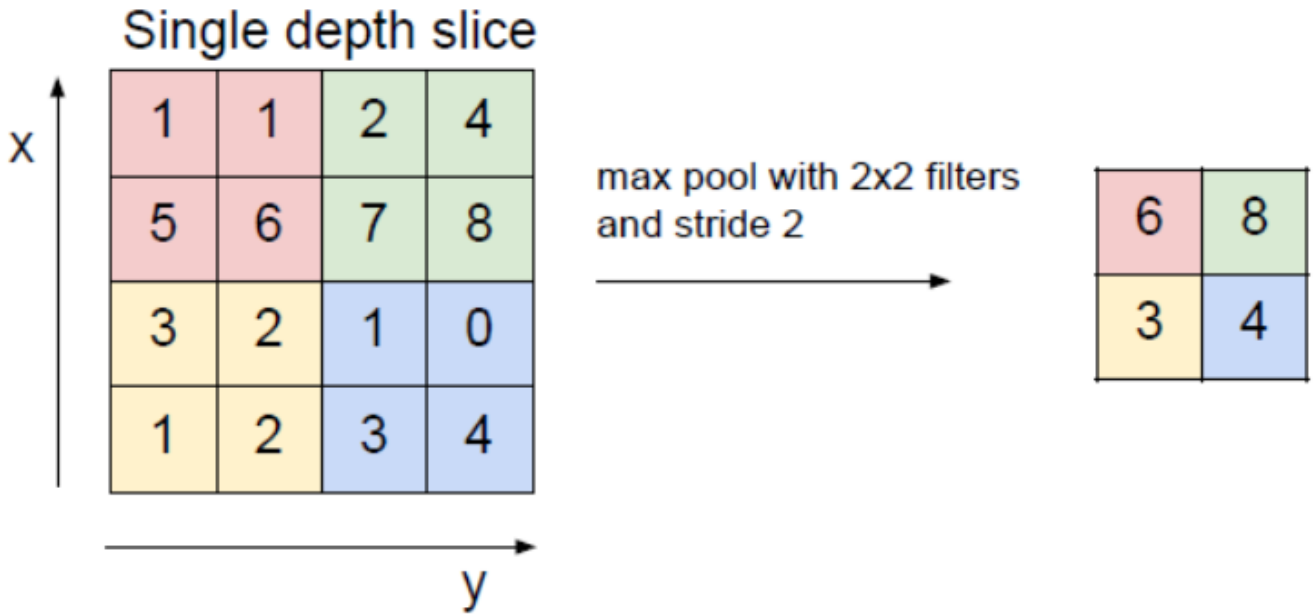


CNN的结构元件：池化的意义

- 在卷积神经网络中，经常会进行池化操作，池化层往往在卷积层后面，通过池化来降低卷积层输出的特征向量，同时改善结果（不易出现过拟合）。
- 图像具有一种“静态性”的属性，这也就意味着在一个图像区域有用的特征极有可能在另一个区域同样适用。因此，为了描述大的图像，一个很自然的想法就是对不同位置的特征进行聚合统计，例如，人们可以计算图像一个区域上的某个特定特征的平均值（或最大值）来代表这个区域的特征。这就是池化操作，通过池化我们可以降低特征的维度。

CNN的结构元件：池化层

- 池化层：F*F+S，其中F是窗口长度，S是移动步长。
- 常见的CNN都是使用最大化池化层，即输出窗口中的最大值。
- 均值池化层(输出窗口内的均值)，最近常用于替代全连接层。
- 下图是F=2,S=2最大化池化的例子：



5
6

CNN的结构元件：全连接层&损失函数

- 全连接层：常见的前向传播网络，每个神经网络节点与前一层的所有节点连接。
  - 全连接层可以看做F=H=W, P=0的卷积层。
  - 大型网络中常用均值池化层替代全连接层(减少参数个数)
- 损失函数：常见的有交叉熵，hinge, 平均平方误差等。

5
7

Softmax: 归一化指数函数

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \qquad S(x) = \frac{1}{1+e^{-x}}$$

	Softmax	Sigmoid
本质	离散概率分布	非线性映射
任务	多分类	二分类
定义域	某个一维向量	单个数值
值域	[0,1]	（0,1）
结果之和	一定为1	为某个正数

5
8

ImageNet Classification with Deep Convolutional Neural Networks

Alex Krizhevsky, Ilya Sutskever, Geoffrey Hinton,

Advances in Neural Information Processing Systems 2012

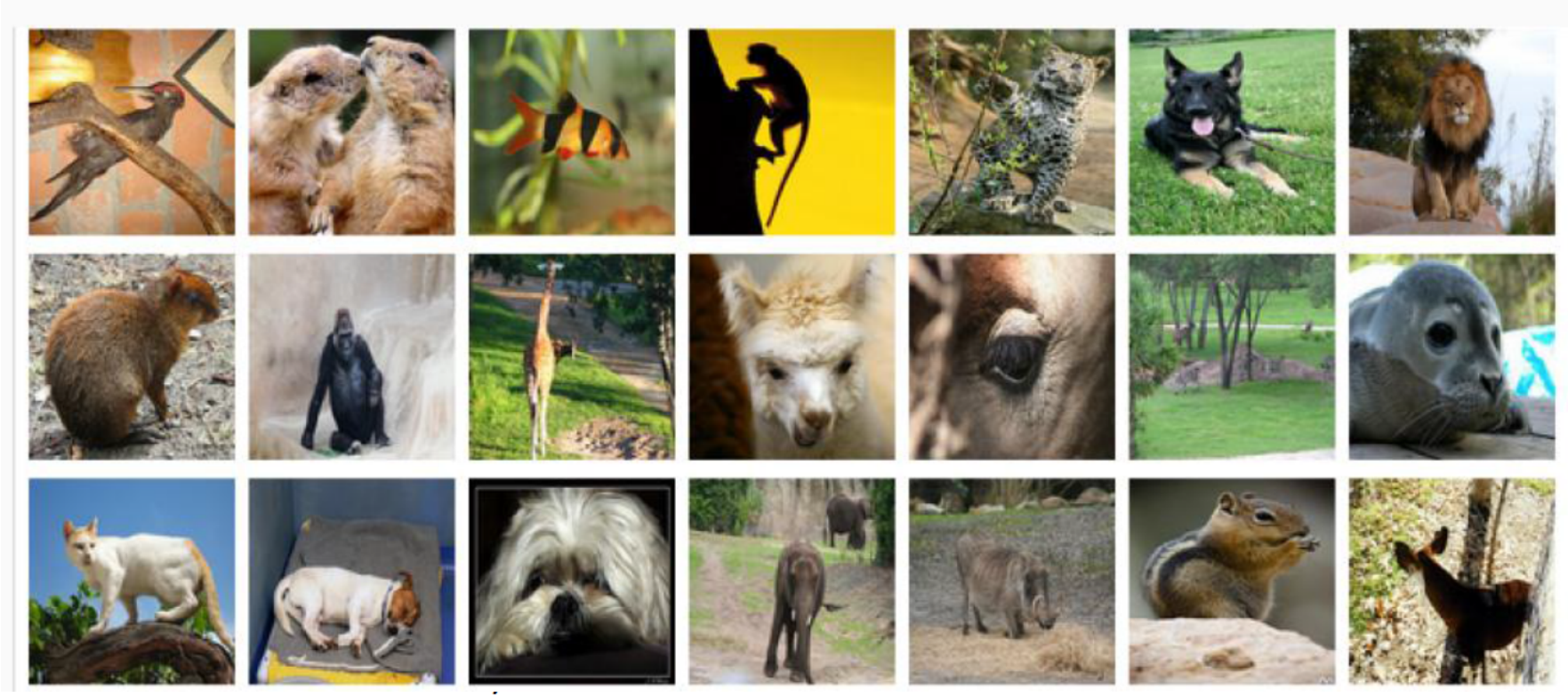
ImageNet

- 15M images
- 22K categories
- Images collected from Web
- Human labelers (Amazon's Mechanical Turk crowd-sourcing)
- ImageNet Large Scale Visual Recognition Challenge (ILSVRC-2010)
  - 1K categories
  - 1.2M training images (~1000 per category)
  - 50,000 validation images
  - 150,000 testing images
- RGB images
- Variable-resolution, but this architecture scales them to 256x256 size

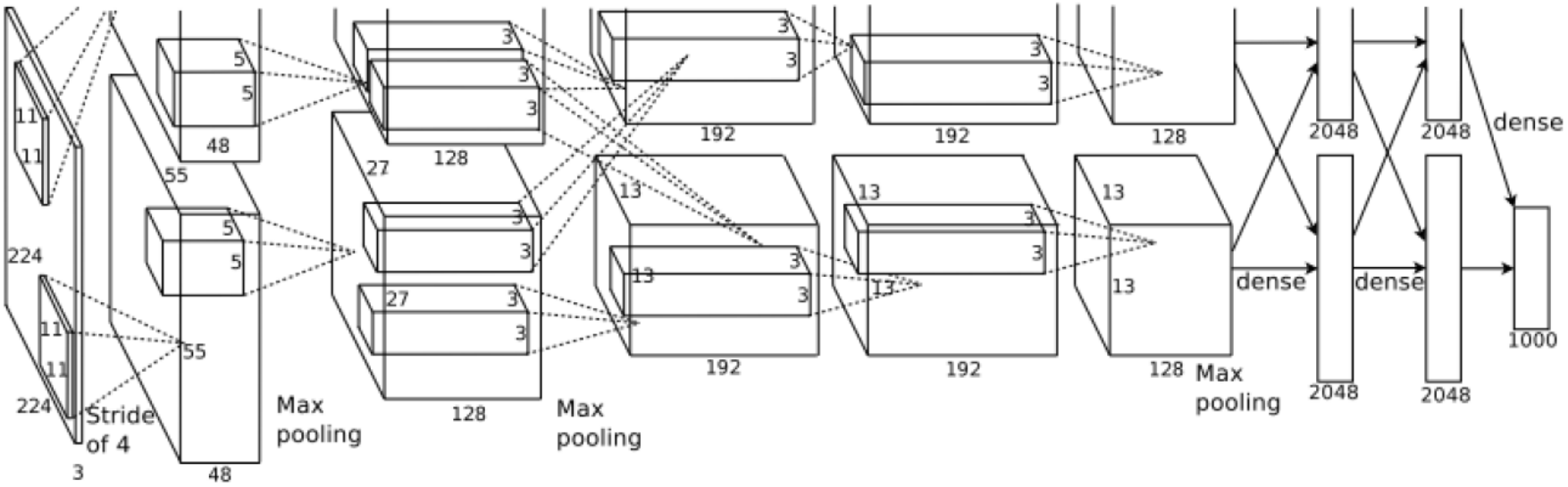
ImageNet–1

Classification goals:

- Make 1 guess about the label (Top-1 error)
- make 5 guesses about the label (Top-5 error)



The CNN Architecture



The first convolutional layer filters the 224×224×3 input image with 96 kernels of size 11×11×3 with a stride of 4 pixels (this is the distance between the receptive field centers of neighboring neurons in the kernel map. 224/4=56

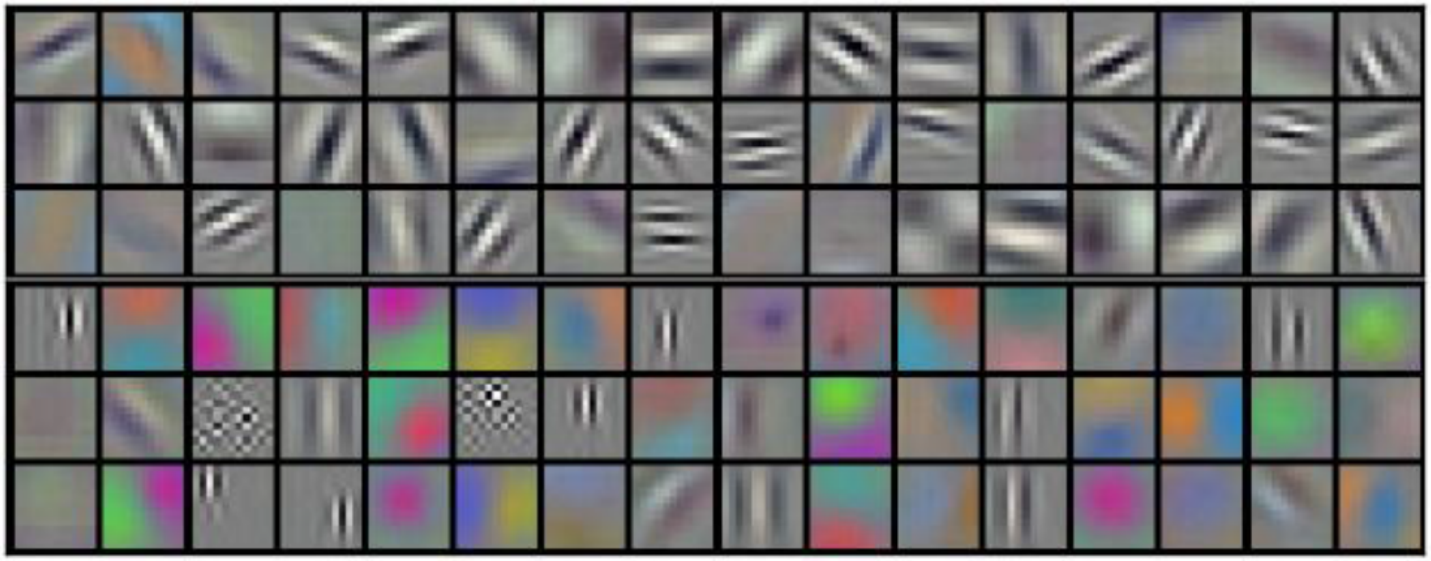
The pooling layer: form of non-linear down-sampling. Max-pooling partitions the input image into a set of rectangles and, for each such sub-region, outputs the maximum value

Experimental Setting

- Trained with stochastic gradient descent
- on two NVIDIA GTX 580 3GB GPUs
- for about a week

- 650,000 neurons
- 60,000,000 parameters
- 630,000,000 connections
- 5 convolutional layer, 3 fully connected layer
- Final feature layer: 4096-dimensional

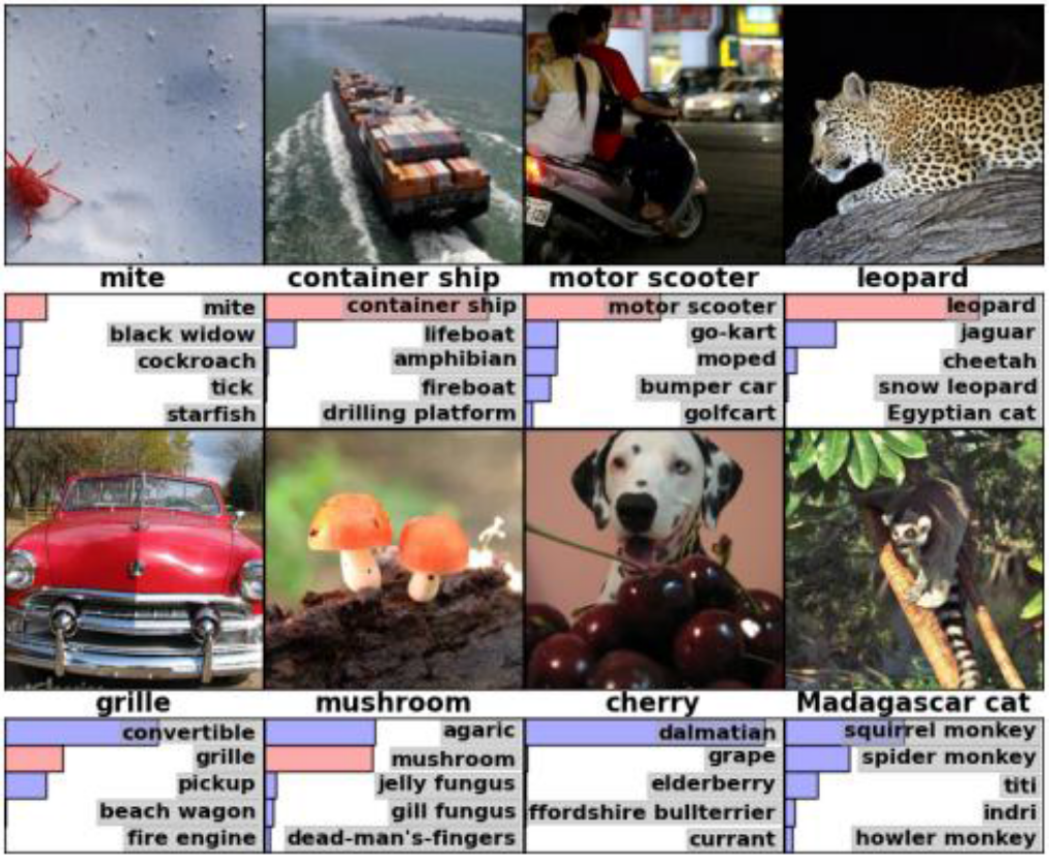
The First Convolutional Layer



96 convolutional kernels of size 11×11×3 learned by the first convolutional layer on the 224×224×3 input images.

The top 48 kernels were learned on GPU1 while the bottom 48 kernels were learned on GPU2

Experimental Results



ILSVRC-2012 competition:

15.3% accuracy

2nd best team: 26.2% accuracy

CNN VS 全连接网络

- 局部感受野
  - 保留图像的拓扑结构
  - 提取图像的基础特征
- 权值共享
  - 减少可训练参数→防止过拟合
  - 基础特征适用于图像的所有位置→平移、扭曲不变性
- 降采样
  - 提取层次化特征
  - 减少具体位置对于最终结果的影响→平移、扭曲不变性

参考资料

- Hubel, David H., and Torsten N. Wiesel. "Receptive fields, binocular interaction and functional architecture in the cat's visual cortex." *The Journal of physiology* 160.1 (1962): 106-154.
- Hubel, David H., and Torsten N. Wiesel. "Receptive fields of single neurones in the cat's striate cortex." The Journal of physiology 148.3 (1959): 574-591.
- Fukushima, Kunihiko, and Sei Miyake. "Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition." Competition and cooperation in neural nets. Springer Berlin Heidelberg, 1982. 267-285.
- LeCun, Yann, et al. "Gradient-based learning applied to document recognition." Proceedings of the IEEE 86.11 (1998): 2278-2324.
- Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." Advances in neural information processing systems. 2012.

参考资料-1

- UNDERSTANDING CONVOLUTIONAL NEURAL NETWORKS FOR NLP [http://www.wildml.com/2015/11/understanding-convolutional-neural-networks-for-nlp/]
- CS231n: Convolutional Neural Networks for Visual Recognition [http://cs231n.stanford.edu/]
- Szegedy, Christian, et al. "Going deeper with convolutions." Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2015.
- He, Kaiming, et al. "Deep residual learning for image recognition." Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2016.

Deep Learning Reading List

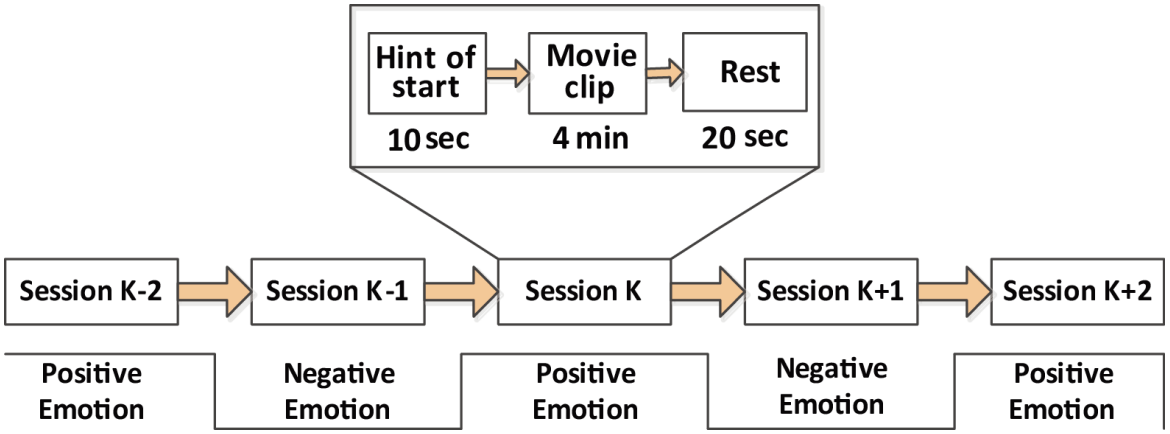
1. **Deep learning**(2015), Y Lecun，Y Bengio，G Hinton.（review）
2. **Teaching machines to read and comprehend** (2015), K. Hermann et al. (Natural Language Processing)
3. **Batch normalization: Accelerating deep network training by reducing internal covariate shift** (2015), S. Loffe and C. Szegedy. (Optimization / Training Techniques)
4. **Inception-v4, inception-resnet and the impact of residual connections on learning** (2016), C. Szegedy et al. (Convolutional Neural Network Models)
5. **You only look once: Unified, real-time object detection** (2016), J. Redmon et al. (Image: Segmentation / Object Detection)

Deap Learning Reading List-1

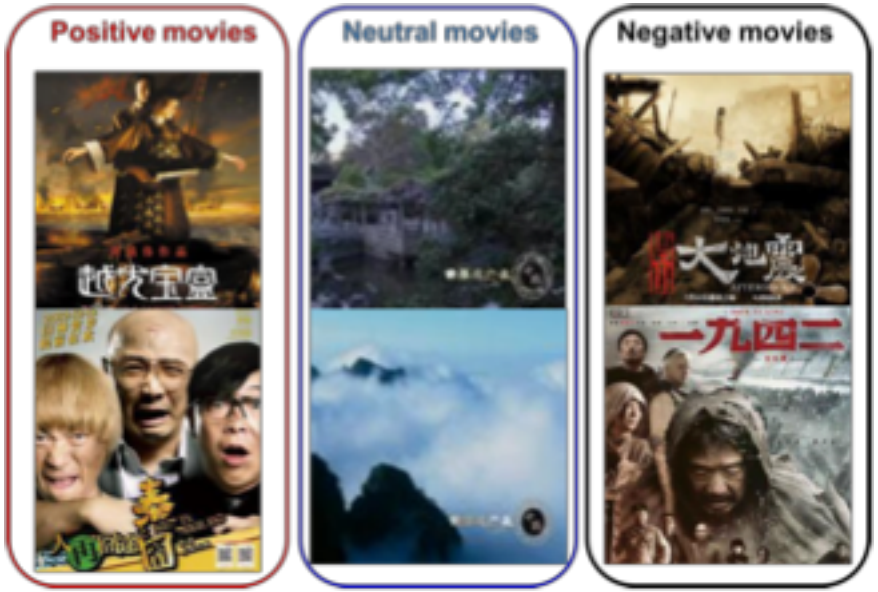
6. **Fully convolutional networks for semantic segmentation** (2015), J. Long et al.. (Image: Segmentation / Object Detection)
7. **Deep residual learning for image recognition** (2016), K. He et al. (Convolutional Neural Network Models)
8. **Fast R-CNN** (2015), R. Girshick.(Image: Segmentation / Object Detection)
9. **Image Super-Resolution Using Deep Convolutional Networks** (2016), C. Dong et al. (Image / Video / Etc)
10. **Neural Architectures for Named Entity Recognition** (2016), G. Lample et al (Natural Language Processing)
11. **End-to-end attention-based large vocabulary speech recognition** (2016), D. Bahdanau et al. (Speech / Other Domain)
12. Learning deep physiological models of affect. HP Martinez, Y Bengio, GN Yannakakis

Applications to EEG-based Emotion Recognition

Emotion Experiments



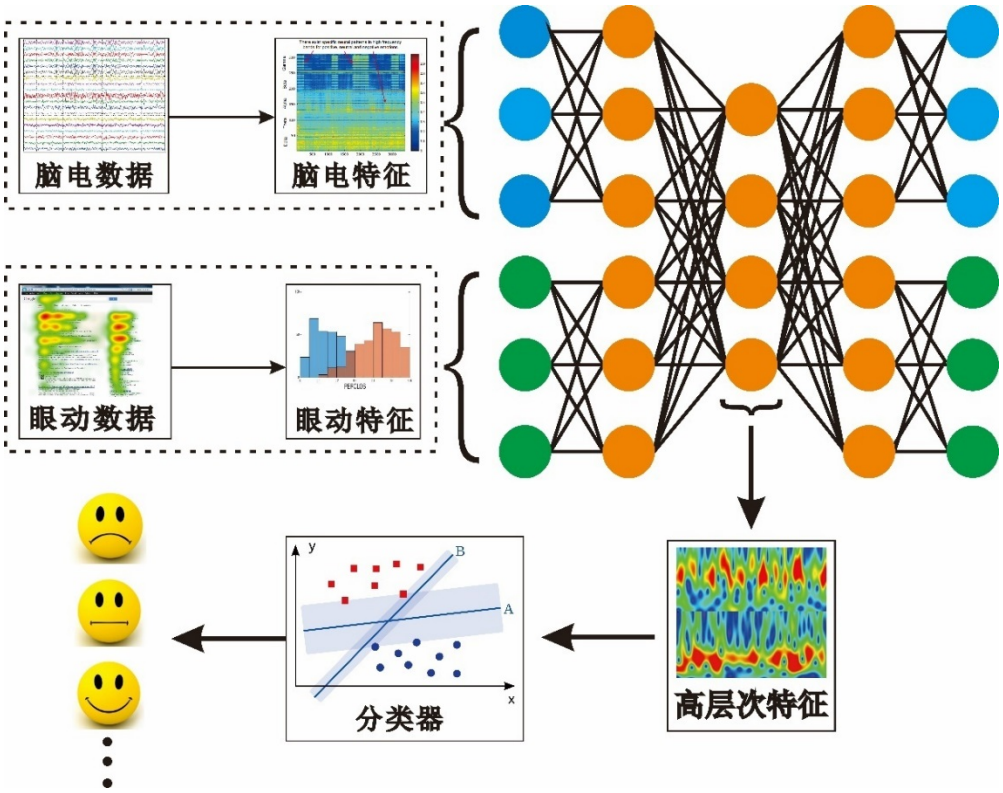
About 15 movie clips were used in each experiment, approximately 5 clips for each emotion (positive, neutral and negative).

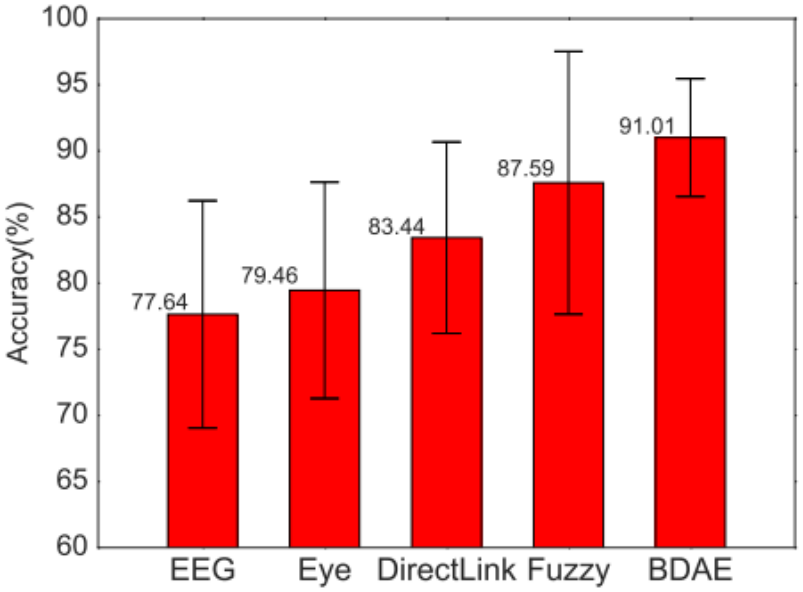


Multimodal Emotion Recognition With Multimodal Deep Learning

Multimodal Facilitation

同时输入脑电和眼动数据，重建脑电和眼动数据。网络提取融合后的高层次特征进行识别。





Wei Liu, Wei-Long Zheng, **Bao-Liang Lu***, Emotion Recognition using Multimodal Deep Learning, Proc. of ICONIP2016, Kyoto, 2016

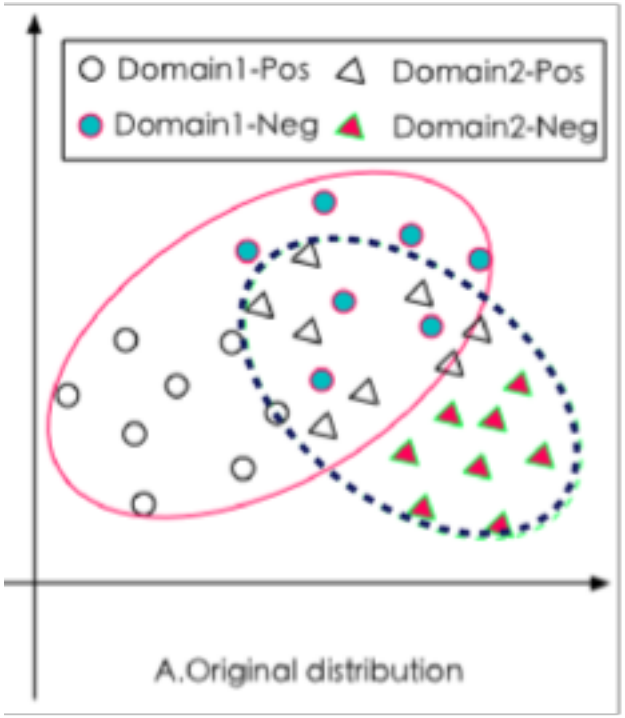
Personalizing EEG-based Affective Models with Transfer Learning

Wei-Long Zheng and Bao-Liang Lu, Personalizing EEG-based Affective Models with Transfer Learning, Proc. of the 25th International Joint Conference on Artificial Intelligence (IJCAI-16).

Challeges of Subject Transfer

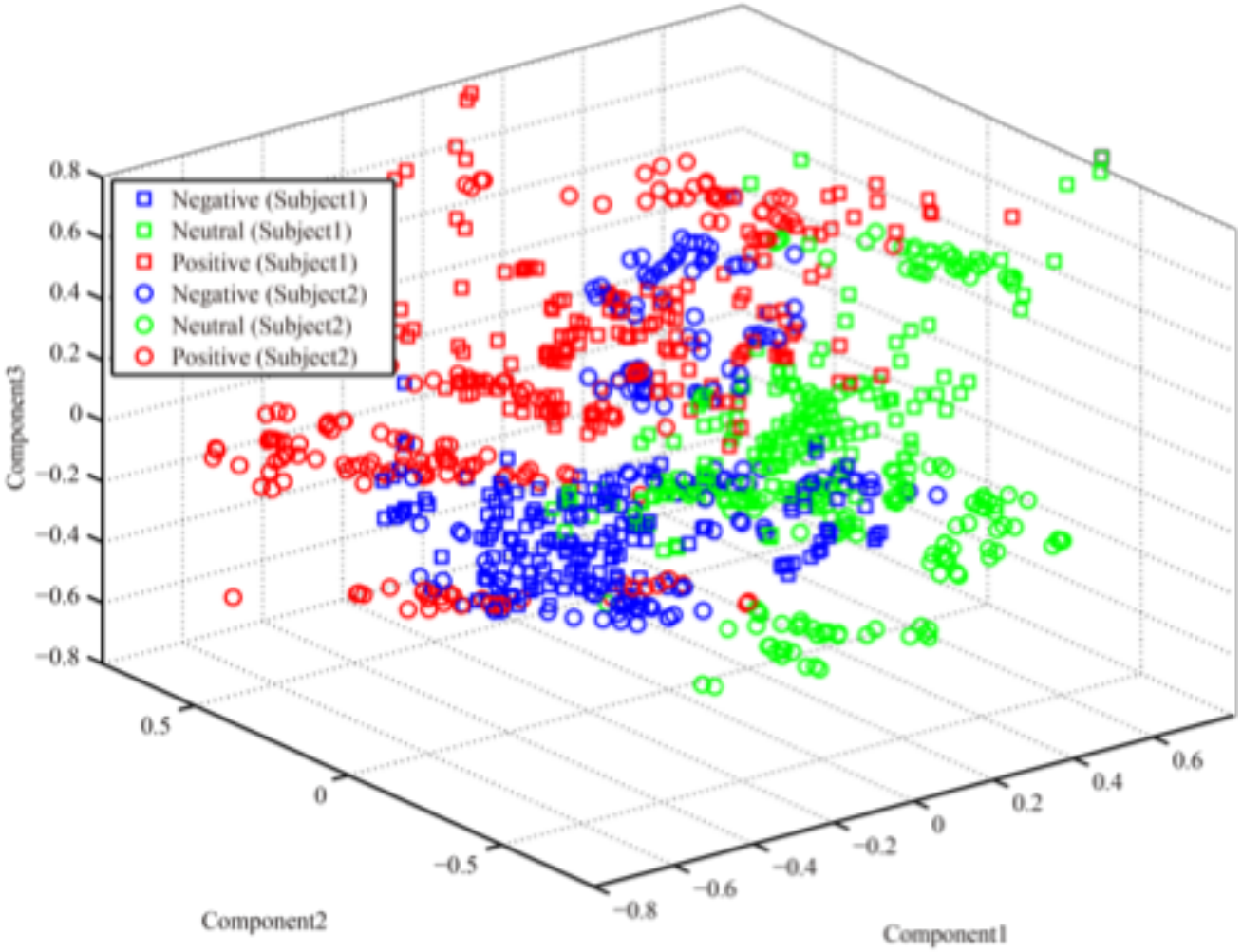
The feature distributions of EEG data of source subjects and target subjects are not independently and identically distributed (i.i.d.) due to:

- The structural and functional variability between subjects
- The non-stationary nature of EEG signals
- The inherent changes of environmental variables

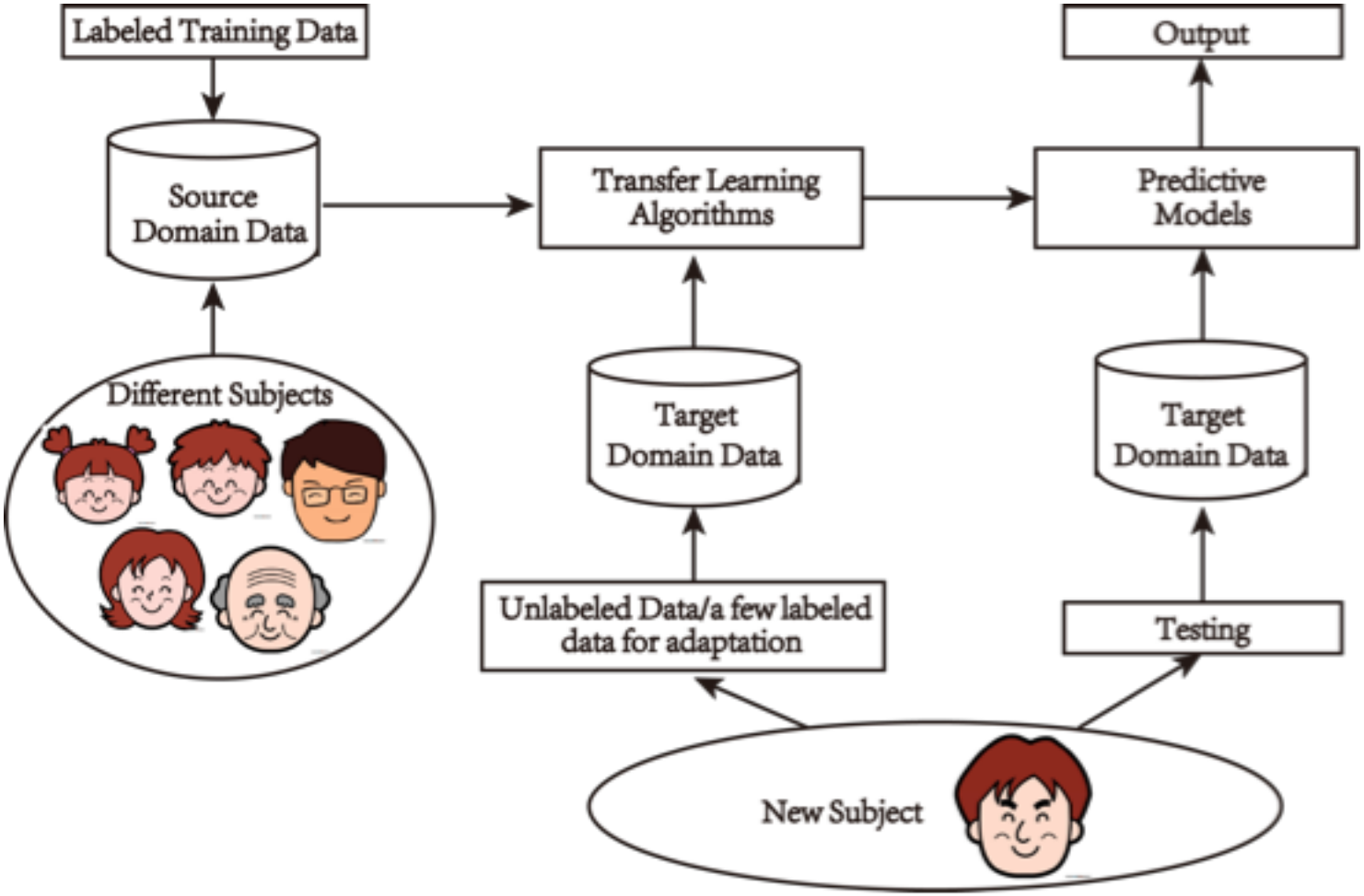


Covariate Shift Challenges

- Individual Differences Across Subjects and Sessions
- Non-stationary Characteristics of EEG

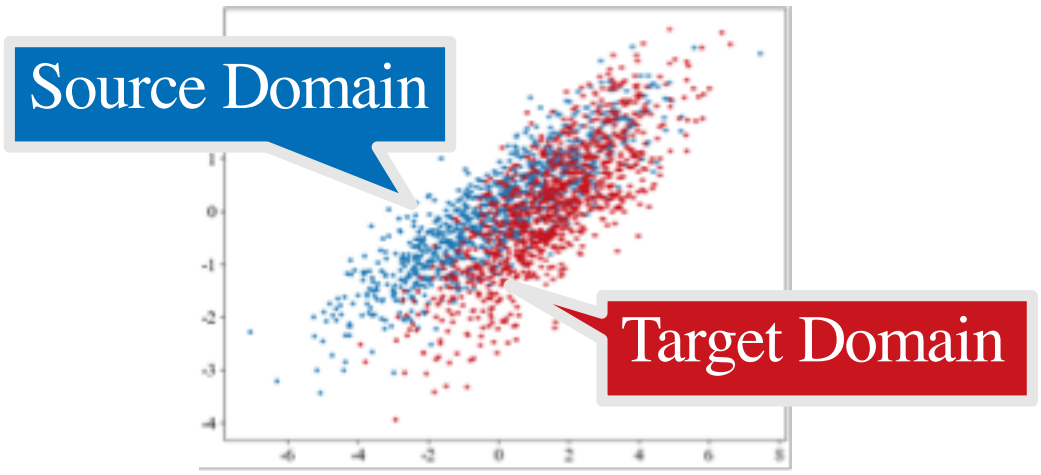


Transfer learning framework



Domain Shift and Domain Adaptation

- Training data are drawn from source domain $\{X_s, \varphi_s\}$, and test data are drawn from target domain $\{X_t, \varphi_t\}$.
- Both domain have the same set of features $(X_s, X_t \in R^m)$.
- Domain shift $(P(X_s) \neq P(X_t))$ makes ordinary learning methods degenerate.
- We can use domain adaptation methods to eliminate or reduce the domain shift.



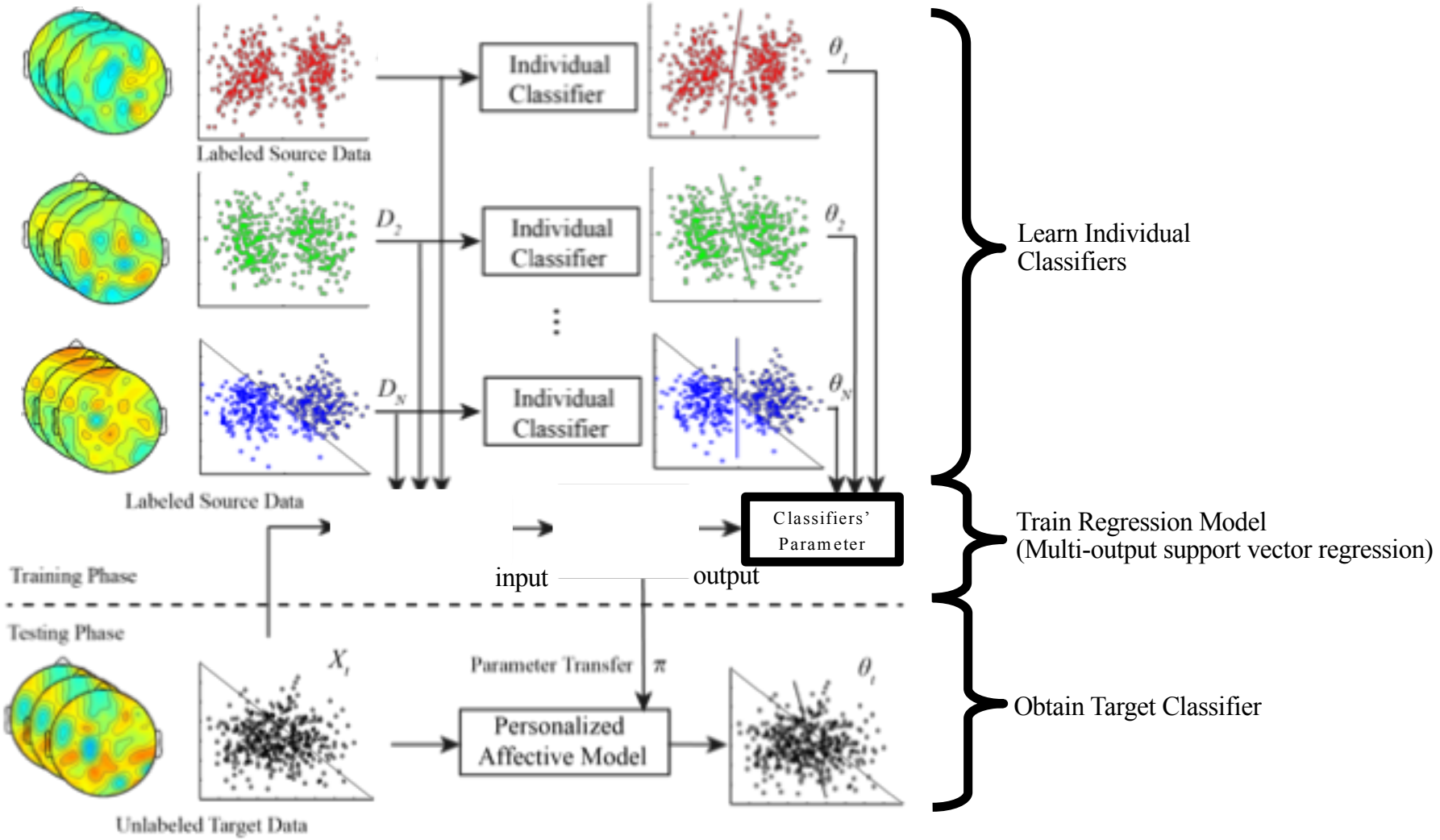
Feature Reduction based Subject Transfer

Although the distributions of source domain and target domain in high dimensional space are different, we may find a low dimensional manifold space where the distributions of both domains are similar.

TCA and KPCA try to learn a set of common transfer components underlying both the source domain and the target domain. When projected to this subspace, the difference of feature distributions of both domains can be reduced.

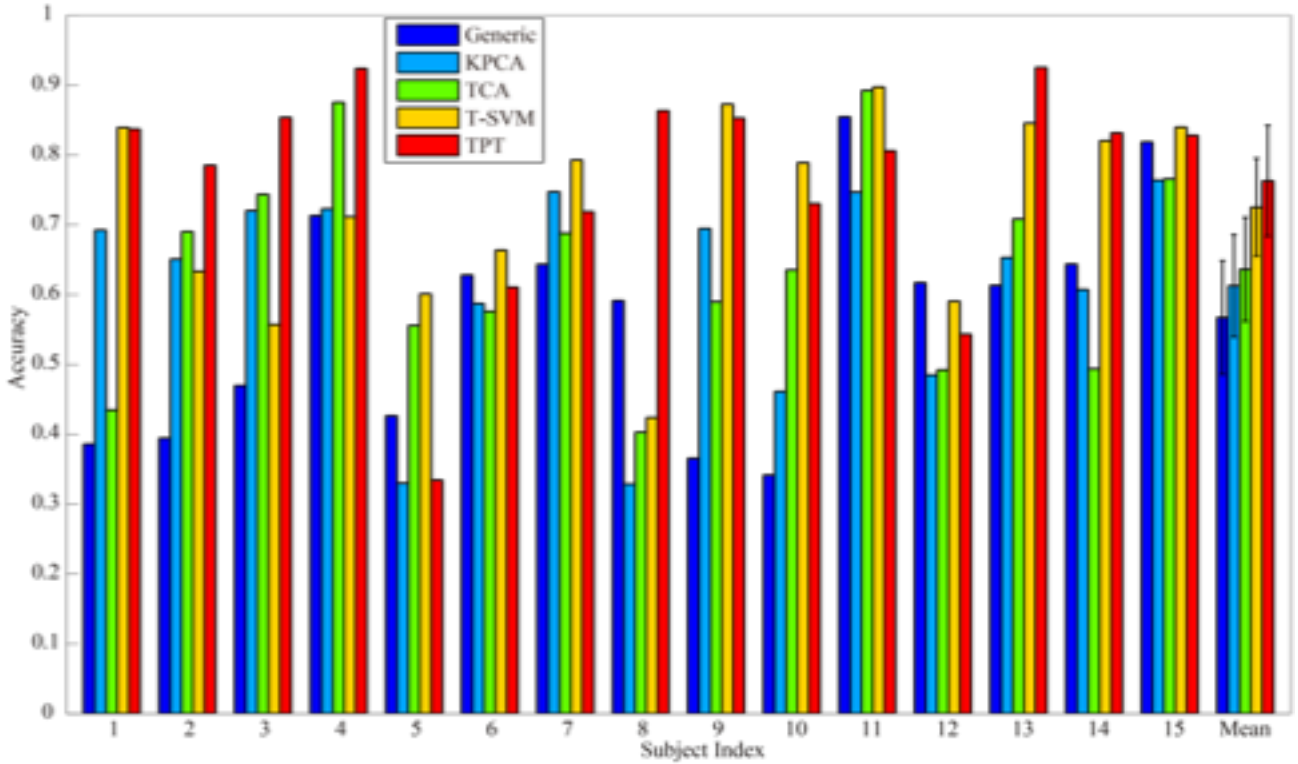
$$P(\phi(X_S)) \approx P(\phi(X_T))$$
$$P(Y_S|\phi(X_S)) \approx P(Y_T|\phi(X_T))$$

Transductive Parameter Transfer



Wei-Long Zheng and Bao-Liang Lu*, Personalizing EEG-based Affective Models with Transfer Learning, Proc. of the 25th International Joint Conference on Artificial Intelligence (IJCAI-16), New York

Experimental Results



Stats.	Generic	KPCA	TCA	T-SVM	TPT
Mean	0.5673	0.6128	0.6364	0.7253	**0.7631**
Std.	0.1629	0.1462	0.1488	**0.1400**	0.1589

(Wei-Long Zhen,, Bao-Liang Lu, Personalizing EEG-based Affective Models with Transfer Learning. IJCAI 2016)

Experimental Results

- Our framework significantly outperforms the state-of-the-art methods with mean accuracy of 87.07% and standard deviation of 0.0714 on SEED dataset.
- On DEAP dataset, our framework achieves mean accuracies of 66.85% and 67.99% and standard deviations of 0.0552 and 0.0656 on arousal and valence classifications, respectively

Methods	SEED		DEAP-Arousal		DEAP-Valence	
	Mean	Std.	Mean	Std.	Mean	Std.
SVM	0.5673	0.1629	0.4922	0.1571	0.5036	0.1125
KPCA	0.6128	0.1462	0.5891	0.1521	0.5658	0.0980
TCA	0.6364	0.1488	0.5193	0.1539	0.5516	0.1069
TPT	0.7631	0.1589	0.5577	0.1496	0.5564	0.1221
WGANDA-Bas.	0.5260	0.1831	0.5183	0.1406	0.5164	0.0929
WGANDA-Adv.	**0.8707**	**0.0714**	**0.6685**	**0.0552**	**0.6799**	**0.0656**

Useful Sites & Software on Deep Learning

- http://deeplearning.net/reading-list/ (Bengio's group)
- http://ufldl.stanford.edu/wiki/index.php/UFLDL_Recommended_Readings
- http://www.cs.toronto.edu/~hinton/
- http://deeplearning.net/tutorial/

- **TensorFlow: A Open-Source software library for machine intelligence**

  https://www.tensorflow.org/

Other Open Source Software : Caffe; Torch; MXNet

Related Tutorials

- Deep Learning tutorials (python): http://deeplearning.net/tutorials
- Stanford deep learning tutorials with simple programming assignments and reading list

  http://deeplearning.stanford.edu/wiki/
- ACL 2012 Deep Learning for NLP tutorial http://www.socher.org/index.php/DeepLearningTutorial/
- ICML 2012 Representation Learning tutorial http://www.iro.umontreal.ca/~bengioy/talks/deep-learning-tutorial-2012.html
- IPAM 2012 Summer school on Deep Learning

  http://www.iro.umontreal.ca/~bengioy/talks/deep-learning-tutorial-aaai2013.html

机器学习小结

- 感知器
- K-近邻算法
- 多层感知器
- BP算法
- 多级筛选网络
- 最小最大模块化网络
- 支持向量机
- 深度学习

工程实践与科技创新课程：超并行机器学习与海量数据挖掘

第四讲（下）：并行程序设计大作业说明

吕宝粮
计算机科学与工程系
电信楼群3号楼431
Tel: 021-3420-5422
bllu@sjtu.edu.cn
http://bcmi.sjtu.edu.cn/~blu

并行程序设计

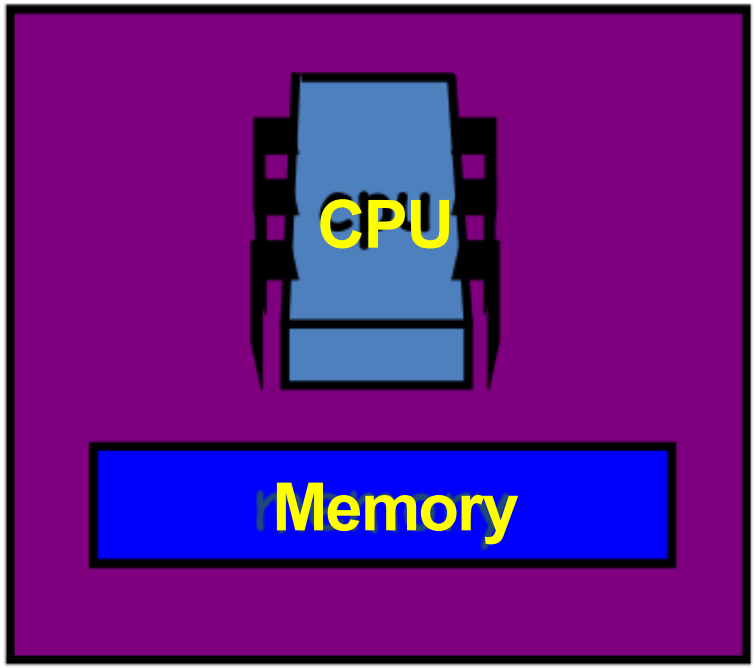
- 串行计算
- 并行计算
- 顺序程序和并行程序
- 使用多指令流实现并行
- 可扩展性和性能可移植性

串行计算

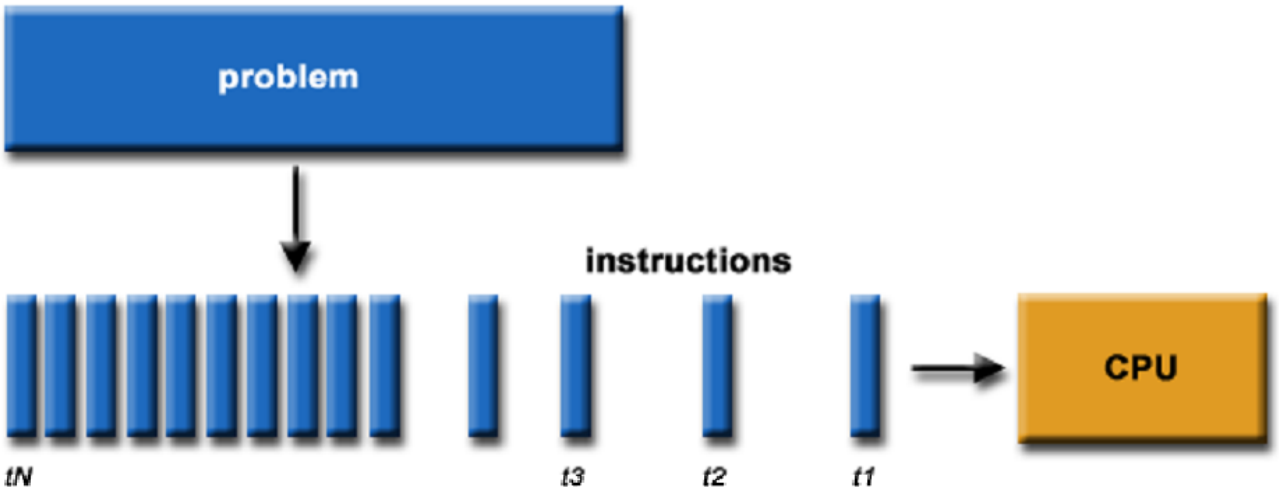
- 传统的软件都是为串行计算编写的：
  - 在只有一个CPU的单个计算机上运行
  - 问题被分解为一系列的指令
  - 指令一个接一个地被执行
  - 任何时刻只有一个指令会被执行

串行计算

- 单处理器的机器正在逐渐被淘汰:



串行计算



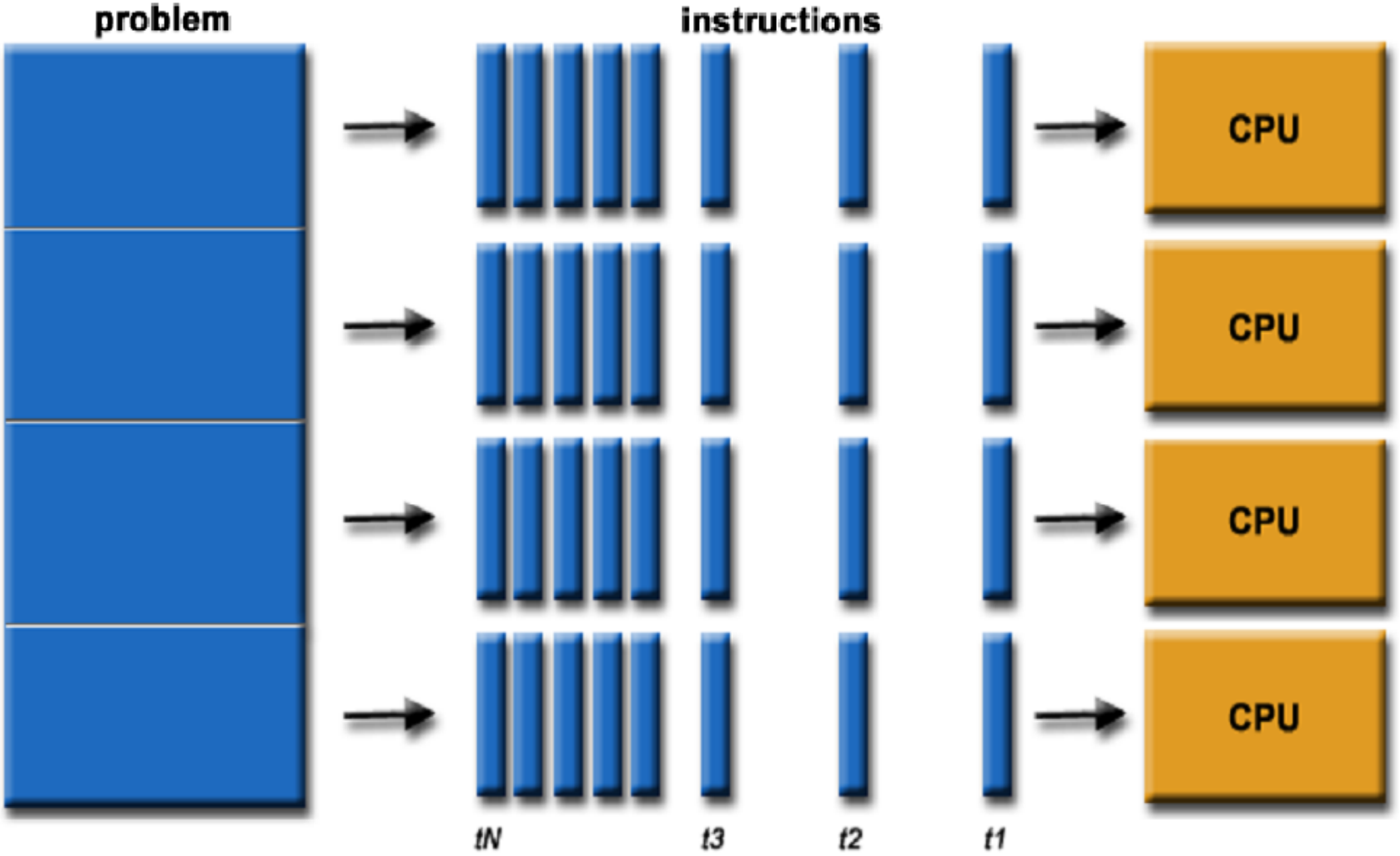
多核处理器

- 现代的机器架构越来越依赖于硬件层的并行化来提高性能：
  - 多核的处理器
  - 流水化的指令执行，即在执行当前指令的同时，对后继的指令进行取址和译码，而前面的指令则正在将结果写回存储器

并行计算

- 简单来说，并行计算是指同时使用多个计算机资源来解决一个计算问题：
  - 使用多个CPU
  - 将一个问题分为多个独立的可以并发执行的部分
  - 每个部分进一步分解为一系列指令
  - 每一部分在不同的CPU上同时执行

并行计算



并行计算

- 为什么要并行计算？
  - 将程序写成并行形式，并且使之可扩展，那么随着硅工艺的进步，将使程序性能持续上升
  - 硅工艺的进展使得在图形处理器上进行通用计算（General Purpose Computing on Graphics Processing Unit, GPGPU)这一概念成为可能

并行计算

- 并行计算环境：
  - 超级计算机
  - 计算机集群
  - 服务器
  - 网格计算
  - 云计算

超级计算机

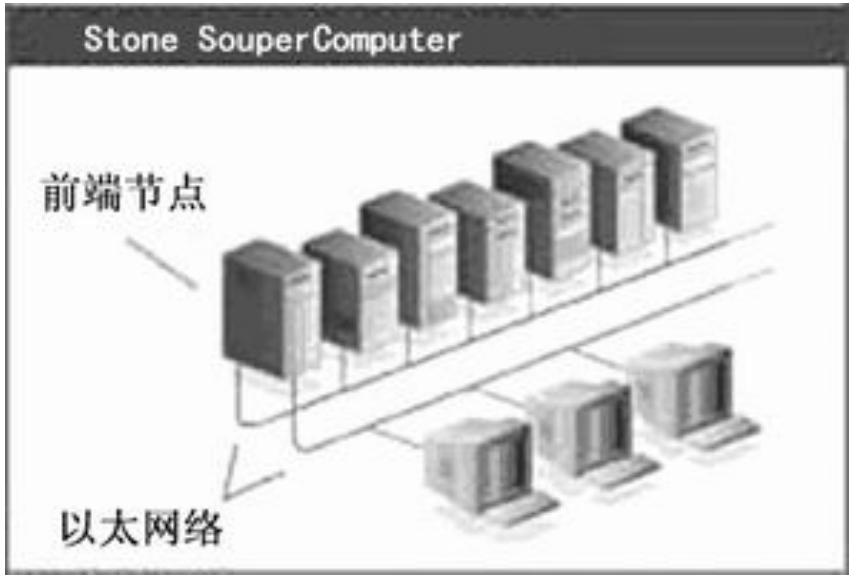
- 超级计算机是指当今世界上最快的计算机，现在的500强全由具有数以百万计的处理器的并行计算机占据

神威·太湖之光超级计算机

- 神威·太湖之光超级计算机安装了40960个中国自主研发的“申威26010”众核处理器，该众核处理器采用64位自主申威指令系统，峰值性能为12.5亿亿次/秒，持续性能为9.3亿亿次/秒。
- 2018年11月12日，全球超级计算机500强榜单公布，“神威·太湖之光”以每秒9.3亿亿次的浮点运算速度名列第3！
- 排名第一的美国“顶点”超级计算机，浮点运算速度从半年前的每秒12.23亿亿次增加到每秒14.35亿亿次

计算机集群

- 计算机集群是指由高速网络连接起来的多个计算机和本地内存

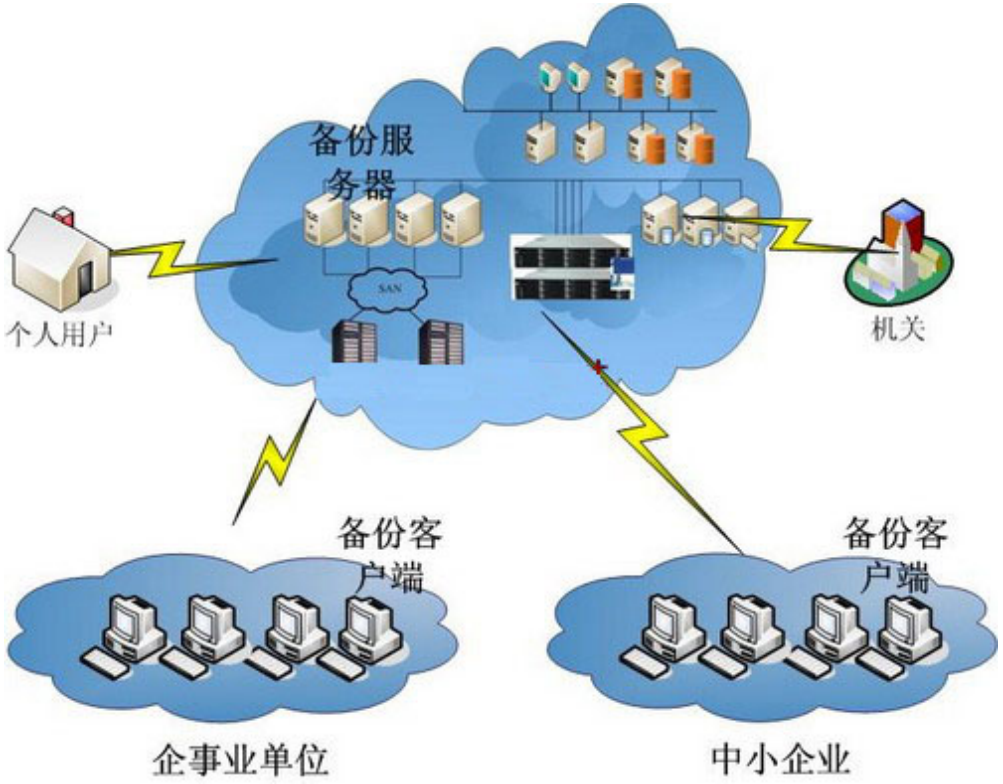


服务器

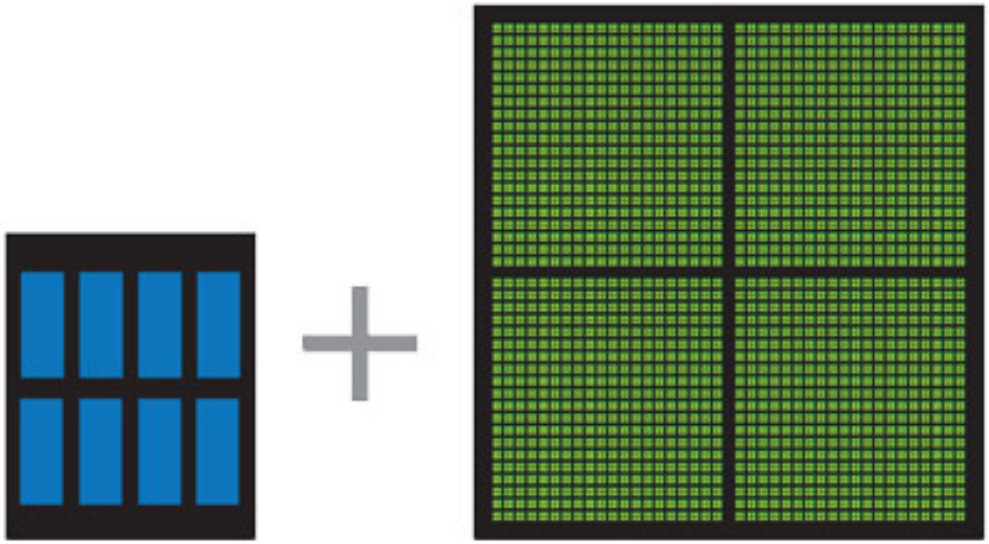
- 由于因特网的扩张和远程服务的普及促成了大量的联网计算机，这些服务需要巨大的计算资源，需要并行程序设计技术来支持

云计算

- 云计算是一种基于互联网的计算方式，通过这种方式，共享的软硬件资源和信息可以按需提供给计算机和其他设备。

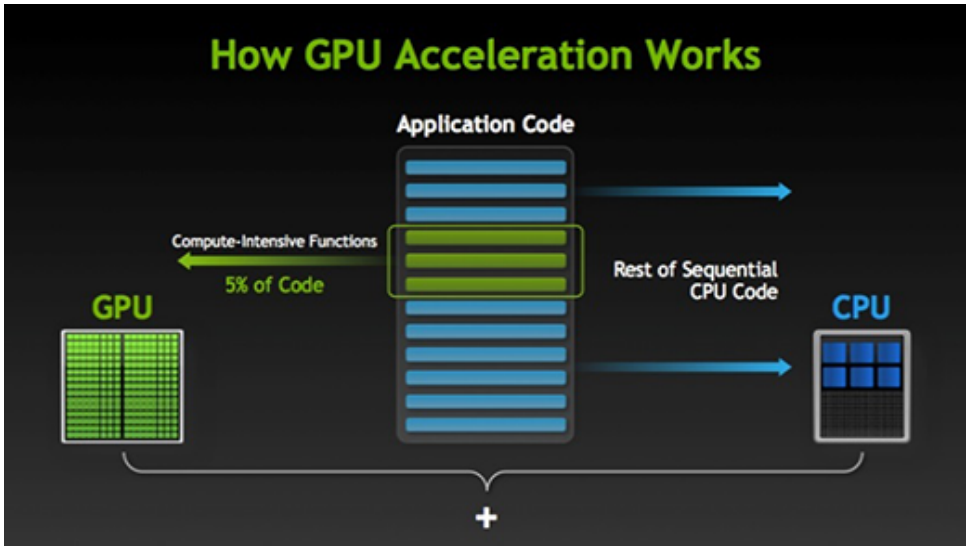


CPU vs GPU



CPU vs GPU小结

- CPU有复杂的逻辑控制单元，并且有大容量的缓存减少访问延迟，擅长复杂的逻辑运算和访问密集型算法
- GPU有数量众多的ALU单元（ 算术逻辑单元 ），适合进行逻辑分支简单和计算密集型高的算法
- 利用CPU+GPU异步并行，让GPU处理数据并行任务，CPU处理复杂逻辑和事物处理任务，可以最大限度地利用计算机资源



各种GPU编程语言的比较

- CUDA
  - 优势: 较成熟，有比较多的学习资料，且容易上手
  - 劣势: 只适用于Nvidia系列的显卡
- OpenCL / DirectCompute
  - 优势：通用标准，可用于所有GPU
  - 劣势：不能发挥出最佳性能。据测对于Nvidia显卡，性能比CUDA速度要慢一倍以上
- CG, Brook等
  - 已淘汰或正在发展中，不推荐

概念和术语

- 并发
  - 并发指的是两个或多个活动在逻辑上同时进行。在数据库领域广泛使用。
- 并行
  - 并行指的是两个或多个活动在物理上同时执行，是通过使用多个计算资源得到的。在系统结构领域广泛使用

顺序程序和并行程序

- 既然并行程序如此重要，为什么不直接写一个将顺序程序翻译成并行执行的编译器？
- 经过计算机科学家30多年的研究，这仍然是个梦想，因为可扩展并行算法与现在的顺序算法在本质上不同，无法实现。

顺序程序和并行程序

- 为了弄清顺序算法和并行算法的不同，我们将比较求一串数总和的不同算法。
- 我们假设有n个数据值，

$$x_0, x_1, x_2, \ldots, x_{n-1}$$

它们存放于数组x中。

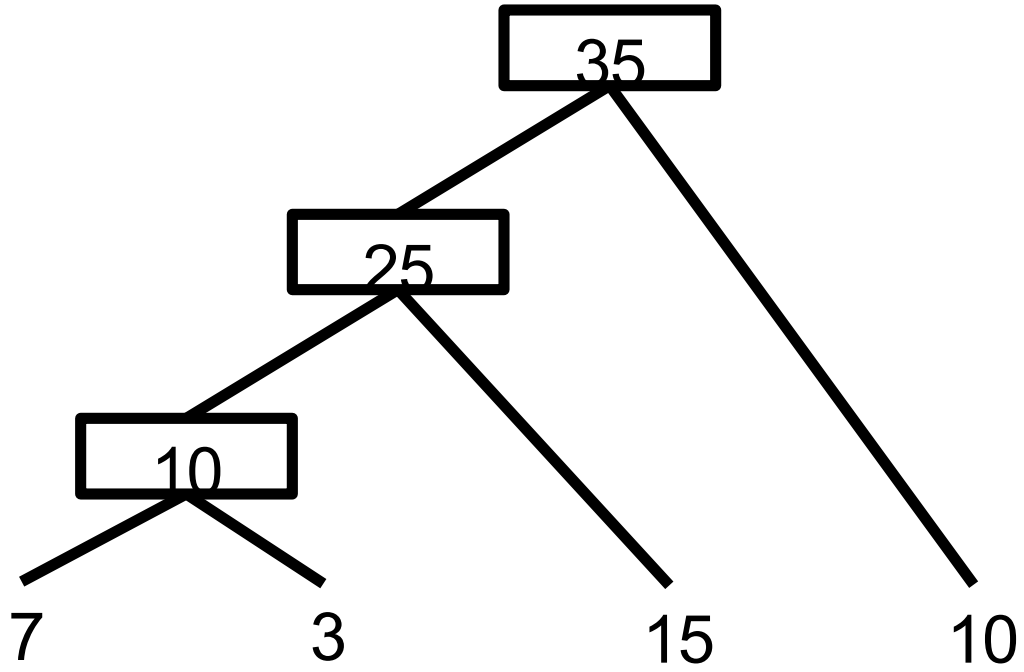
顺序程序和并行程序

- 方法一：迭代求和。这是最直观的方式，先将一个名为sum的变量初始化为0，然后迭代相加数组中的元素。代码如下：

```
sum=0;
for(i=0;i<n;i++)
{
sum+=x[i];
 }
```

顺序程序和并行程序

- 迭代求和，此方法的时间复杂度为0(n)



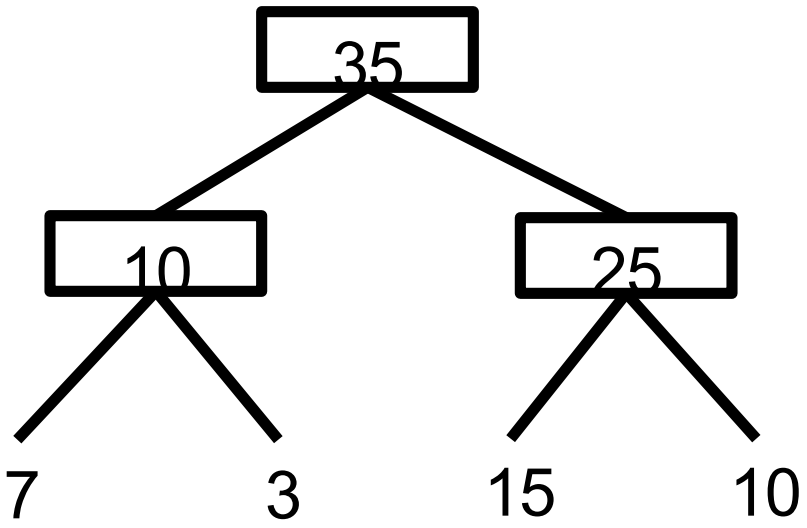
顺序程序和并行程序

- 方法二：成对求和。此方法是将数组中的数据两两相加，然后将得到的结果再两两相加，直到结束。代码如下：

```
t[0]=x[0]+x[1];
t[1]=x[2]+x[3];
t[2]=t[0]+t[1];
```

顺序程序和并行程序

- 成对求和的方法可以看成是生成一棵计算树，示意图如下：



顺序程序和并行程序

- 迭代求和 vs. 成对求和
  - 当只使用一个处理器时，没有区别
  - 当在一个有$n/2$个处理器的机器上时，成对求和的方法就是$O(\log n)$，相比迭代求和的线性复杂度，这是一个明显的改进

使用多指令流实现并行

- 线程，或执行线程，是一个并行的单位。一个线程具备执行一个指令流所需的一切（包括一段私有的程序正文、一个调用堆栈、以及一个程序计数器）。

使用多指令流实现并行

- 为了了解编写正确、高效和可扩展的线程式程序的困难之处，我们将举一个多线程统计一个数组中3的个数的问题。此问题在大多数顺序程序语言中都能很简单地表示，当用多线程来求解时还需要什么？

使用多指令流实现并行

- 第一个求解方案：使用串行方法来求解。代码如下：

```
int *array;
int length;
int count;
int count3s()
{
        int i;
        count = 0;
        for (i = 0; i < length; i++)
        {
                if (array[i] == 3)
                        count++;
        }
        return count;
}
```

使用多指令流实现并行

- 我们用thread_create()来实现该代码的一个并行版本。其中t为线程的个数，id为一个标识线程的整数。

```
int t;
int *array;
int length;
int count;
void count3s()
{
        int i;
        count = 0;
        for (i = 0; i < t; i++)
          thread_create(count3s_thread, i);
}
```

```
void count3s_thread(int id)
{
        int len_per_thd = length / t;
        int start = id * len_per_thd;
        for (i = start; i < start + len_per_thd; i++)
                if (array[i] == 3)
                        count++;
}
```

使用多指令流实现并行

- 遗憾的是，上面的代码不会产生正确的结果，因为递增count的语句中存在竞态条件。此语句在现代机器上通常实现为以下三条指令：
  - 装载count到一个寄存器
  - 递增count
  - 将count写回存储器
- 由于这个递增不是原子的，多个线程的交叉执行将会产生不正确的结果

使用多指令流实现并行

- 第二个求解方案：我们可以使用提供互斥功能的mutex（互斥体）解决上面的问题。
  - 术语：互斥和原子性是一对用来描述不可中断转换的相关术语
  - 互斥：一段代码以互斥方式执行是指在任何时间最多只有一个线程能执行该代码
  - 原子性：如果一组操作要么全执行，要么全不执行，则它就是原子的。

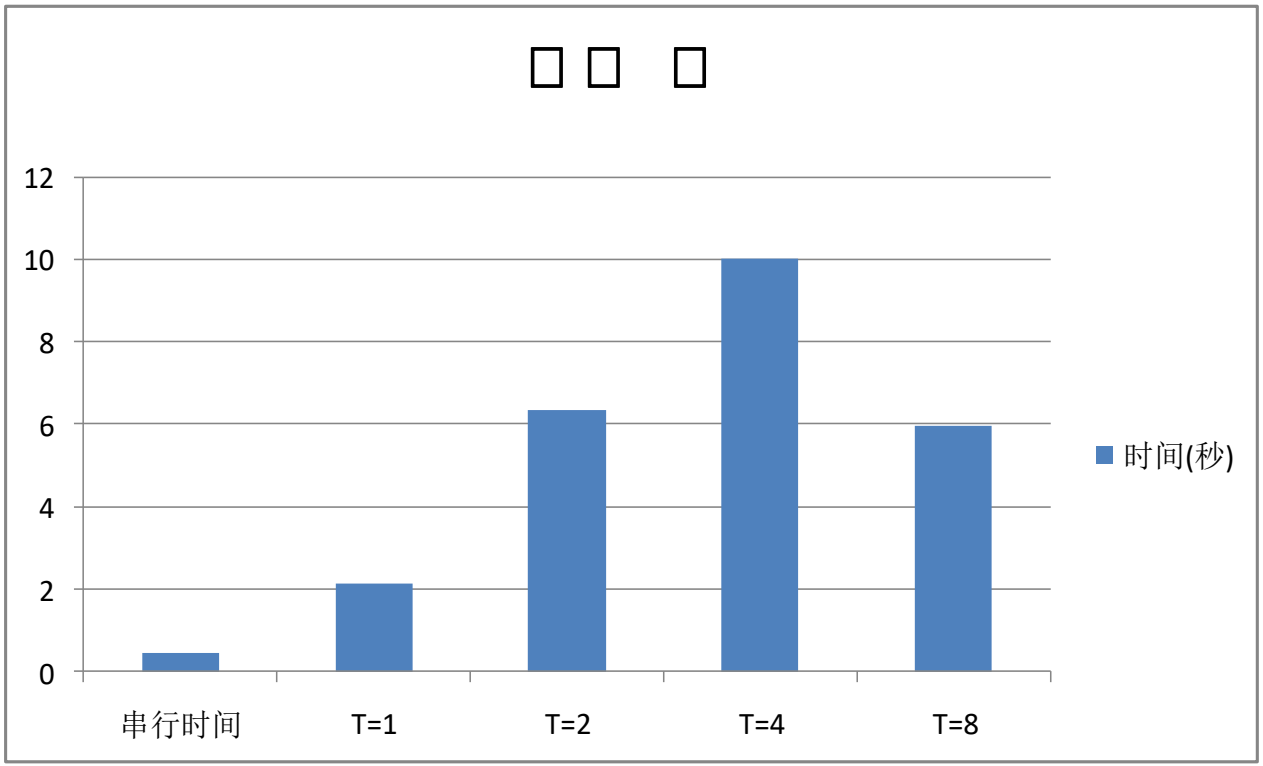
使用多指令流实现并行

- 第二个求解方案的代码实现

```
mutex m;
void count3s_thread(int id)
{
        int len_per_thd = length / t;
        int start = id * len_per_thd;
        for (i = start; i < start + len_per_thd; i++)
        {
                m.lock();
                if (array[i] == 3)
                        count++;
                m.unlock();
        }
}
```

使用多指令流实现并行

- 经过上述修改，这就是一个并行程序。然而，由于使用互斥体的开销使性能受到很大影响，从性能直方图上可以看出其求解速度比最初的顺序代码慢很多。



使用多指令流实现并行

- 第三个方案：这个方案是在了解了开销和竞争后进一步的改进。
- 在此方案中，我们用更大的粒度进行运算。
- 不让线程每次递增count时访问临界区，而是用私有变量private_count累加各自的3统计次数，并且只是在最后进入临界区域与总的结果相加。

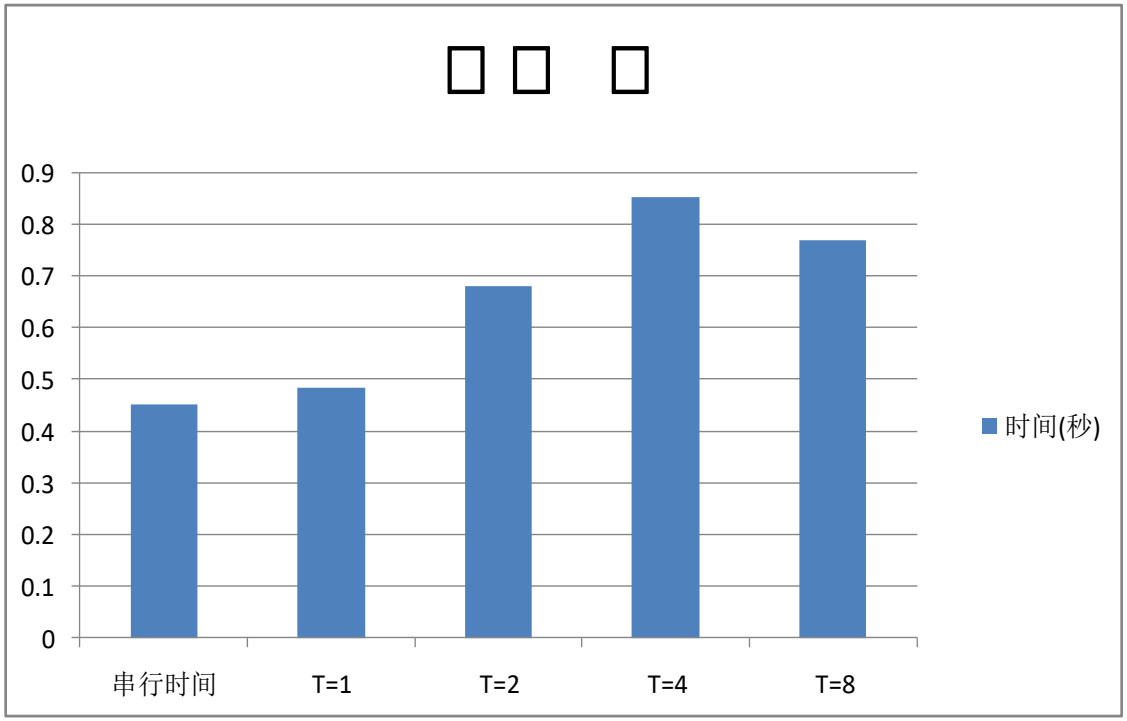
使用多指令流实现并行

- 方案三的代码如下：

```
mutex m;
private_count[MaxThreads];
void count3s_thread(int id)
{
        int len_per_thd = length / t;
        int start = id * len_per_thd;
        for (i = start; i < start + len_per_thd; i++)
        {
                if (array[i] == 3)
                        private_count[id]++;
        }
        m.lock();
        count += private_count[id];
        m.unlock();
}
```

使用多指令流实现并行

- 从下面的性能直方图中可以看出，虽然消除了锁开销，多线程程序与单线程相比仍有显著的下降。



使用多指令流实现并行

- 产生上图中问题的原因，主要是因为硬件使用一种协议以保持一致性的cache（高速缓冲存储器 ），即保证两个处理器所观察到的是相同的存储器映象。
- 处理器0对其L1级cache的存储单元内容的修改将导致处理器1中L1级cache的存储单元的内容变得无效，并且要从处理器0的L1级cache中拷贝数据到处理器1的L1级cahe中
- 这种cache一致性的处理相当昂贵，因为此时数据会在两个cache中跳来跳去。

使用多指令流实现并行

- 第四个方案：在了解了问题出现的原因之后，我们需要更改程序以达到更高的性能。
- 逻辑上不同的数据共享一个物理cache行的现象称为假共享(false sharing)。
- 为了消除假共享，我们通过填充(padding)私有计数器使之处理不同的cache行，使得能够在保持cache一致性的同时避免数据在不同的cache之间跳转。

使用多指令流实现并行

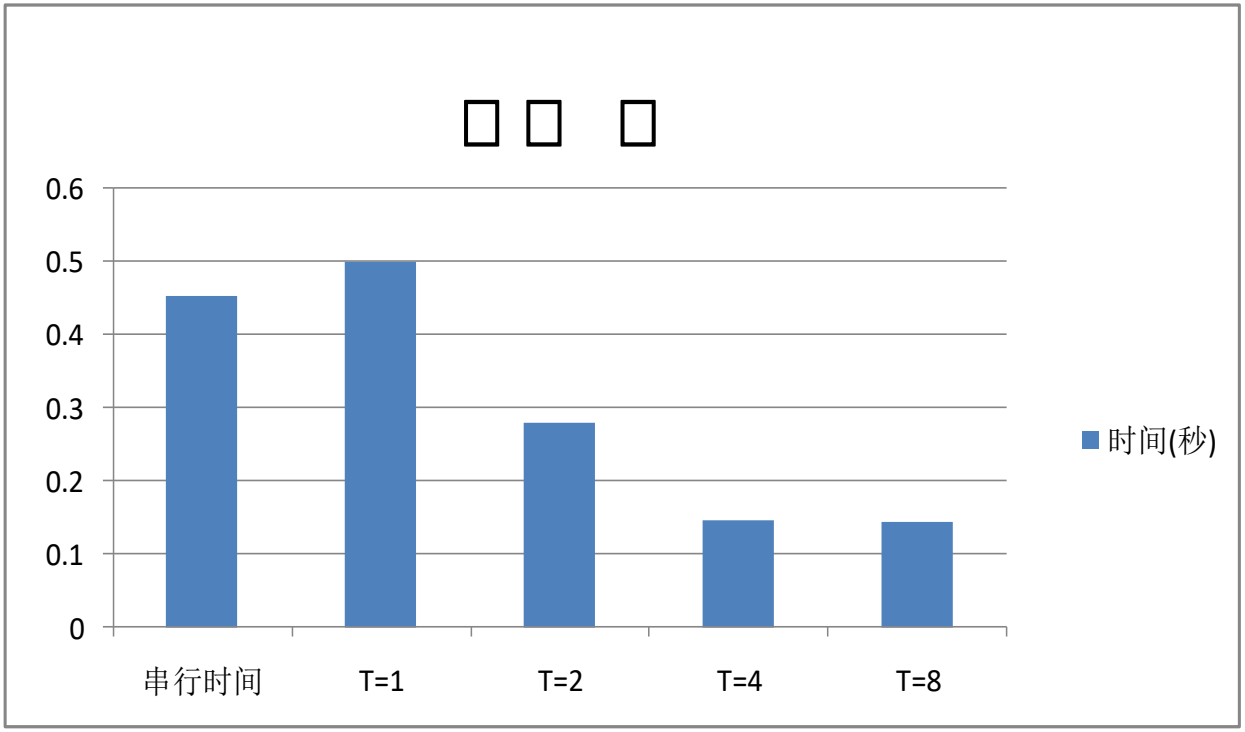
- 第四种方案代码如下：

```
mutex m;
struct padded_int
{
        int value;
        char padding[60];
} private_count[MaxThreads];
void count3s_thread(int id)
{
        int len_per_thd = length / t;
        int start = id * len_per_thd;
        for (i = start; i < start + len_per_thd; i++)
        {
                if (array[i] == 3)

        private_count[id].value++;
        }
        m.lock();
        count += private_count[id].value;
        m.unlock();
}
```

使用多指令流实现并行

- 采用填充措施之后，我们发现消除了互斥的开销，并行程序的优势体现了出来。



可扩展性和性能可移植性

- 良好的并行程序必须具有以下4个特征：
  - 正确
  - 好的性能
  - 可扩展为使用几千个处理器
  - 可移植到各种类型的并行平台上

可扩展性

- 需要考虑代码随着处理器数目的增加 会受到什么影响。
- 当并行处理器数目增加时，物理的极限将迫使设计改变，从而影响程序完成运算。比如通信时延等。
- 高质量的可扩展的程序应当充分利用并行计算中的快速部件，并避免过度使用慢速的部件。

性能可移植性

- 并行计算机通常可被分为以下三类之一：
  - 共享存储器，以多核处理器为代表；
  - 共享地址空间，以各种超级计算机为代表；
  - 分享地址存储器，以集群为代表
- 如果我们希望获得高性能，就不能完全与底层硬件脱离
- 使程序能运行在不同的并行平台上是不够的，还需要保证性能的可移植性

大作业分组

1. Group 1:胡浩天 516030910207,胡浩颐 516030910208
2. Group 2:李文浩 516030910212,张晋豪 516030910272
3. Group 3:陈小双 516030910230,陈纪坤 516030910303
4. Group 4:郭洪一 516030910306
5. Group 5:宫伟植 516030910281,叶盛隆 516030910294
6. Group 6:郎峙煜 516030910358,王晨 516021910501
7. Group 7:闫鸿宇 516030910595,易力 516030910596

大作业提交时间、要求及报告会

- 大作业提交时间：2019年4月30日
- 大作业报告：方法介绍；程序说明；结果分析
- 报告长度：8000-10000字（A4:8-10页）
- 选课的同学全员参加大作业报告会。
- 每组都要准备PPT，并作介绍。
- 每组讲演时间15分钟，提问5分钟。
- 讲演要求：准备15分钟PPT，每人介绍自己在作业中的作用和完成的主要工作。
- 每位组员的介绍时间自己组内决定，但每个组员都必须参加介绍。
- 讲演会时间将安排在5月中旬的某一天，具体时间和地点将另行通知。