

# 计算机视觉：人脸识别

<https://github.com/yangminz/tensorface>

赵阳旻，14307130067

# 目录

## 1. 人脸识别介绍

从 FBI 的文章开始

## 2. 卷积神经网络

一点点深度的学习

## 3. 主成分分析

精彩又典雅的传统数学方法

## 4. 总结

人脸识别：一项既成熟又不成熟的技术

# 人脸识别介绍

# 当然应用是非常广泛的

静态照片的人脸识别应用已经很广泛了，毋论动态视频中的识别



FBI 在某次枪击案中使用了人脸识别技术，从在场人群手机上的和监控中的照片里找到了凶手

# 简单的历史回顾

- ❖ 1960s, 计算图片上眼耳鼻特征的距离和比例，然后和参考值比较
- ❖ 1970s, 使用发色、唇厚等进行自动计算
- ❖ 1988, PCA 的应用横空出世，一个 milestone
- ❖ 1991, 利用 eigenface 的残差误差建立实时系统
- ❖ now, CNN 大行其道

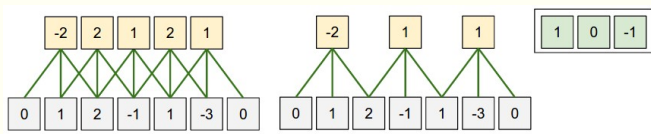
# 卷积神经网络

# 卷积不卷积

数学上独立随机变量  $\xi$  与  $\eta$  的离散卷积是：

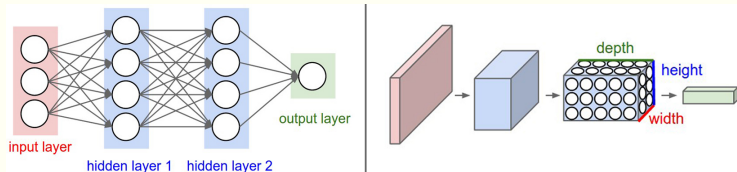
$$P\{\zeta = r\} = \sum_{k=0}^r P\{\xi = k, \eta = r - k\} = \sum_{k=0}^r a_k b_{r-k}$$

CNN 的卷积是：



它只是局部加权平均

# 来自 CS231N 的网络

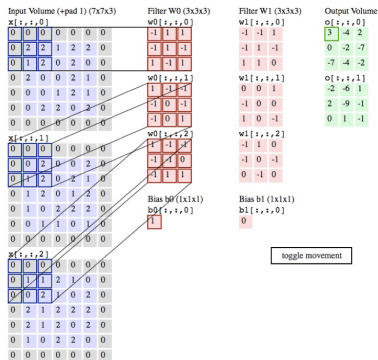


左边是 3 层全连接神经网络，右边是一个卷积神经网络  
CNN 在于，对于复杂的输入——图片，大量地减少了网络参数的数量



# 来自 CS231N 的网络

与一般网络的差别:



带来的问题是，虽然 CNN 通过卷积有效地减少了网络参数但是相比于一般网络，它为什么仍然能够有效？

# Kolmogorov 定理

$\Phi(y)$  是单调有界连续增函数, 其中  $y = f(x_1, x_2, \dots, x_n)$  只是一个有界闭子集上的连续函数, 则:

$\forall \epsilon > 0$ , 存在正整数  $H$  和  $c_j, \theta_j$ , 有  $w_{ij}, i, j \in \{1, 2, \dots, n\}$ ,

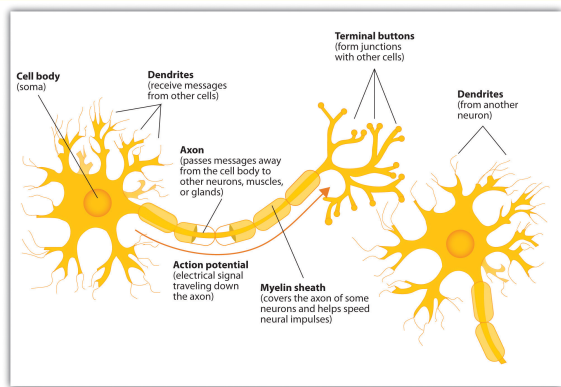
$$g(x_1, x_2, \dots, x_n) = \sum_{j=1}^H c_j \Phi \left( \sum_{i=1}^n w_{ij} \cdot x_i - \theta_j \right)$$

$$\text{s.t. } \max |f(x_1, x_2, \dots, x_n) - g(x_1, x_2, \dots, x_n)| < \epsilon$$

$\Phi(\cdot)$  是我们平时用的非线性函数,  $c, w, \theta$  是随机梯度训练的参数  
CNN 修改了  $w$ , 让函数逼近收敛得更快

# 一种玄学的解释

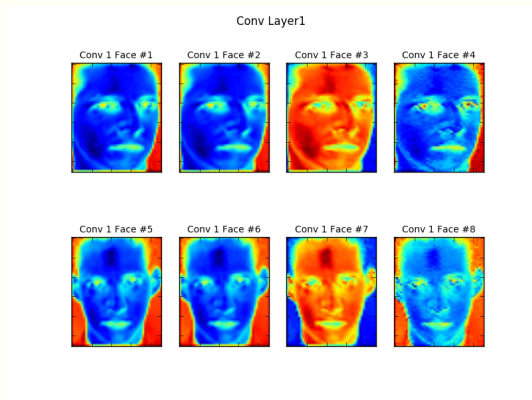
## 局部感知



因为人类的神经元也只具有局部感知能力

# Keras 实验：实际上发生了什么

使用 AT&T 的数据集：



使用了一个训练好的，正确率在 99% 的卷积网络的参数，第一层卷积不同深度的 feature mapping

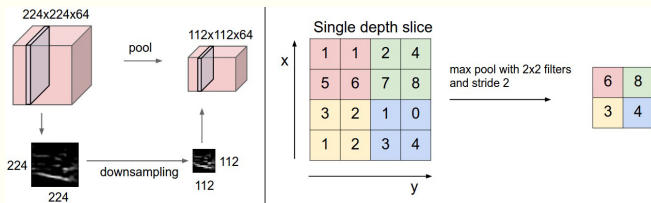
# Keras 实验：模型

一个简单的 CNN:

```
model = Sequential()
print 'Building: conv1 - tanh - maxpooling'
model.add(Convolution2D(filter1, conv_side, conv_side, border_mode='same', sub
model.add(Activation('tanh'))
model.add(MaxPooling2D(pool_size=(pool_side, pool_side)))
print 'Building: conv2 - tanh - maxpooling'
model.add(Convolution2D(filter2, conv_side, conv_side))
model.add(Activation('tanh'))
model.add(MaxPooling2D(pool_size=(pool_side, pool_side)))
print 'Building: flat - dense - tanh - dense - softmax'
model.add(Flatten())
model.add(Dense(1000)) #Full connection
model.add(Activation('tanh'))
model.add(Dense(SUBJECT_NUM))
model.add(Activation('softmax'))
sgd = SGD(Lr=sgd_lr, decay=sgd_decay, momentum=sgd_momentum, nesterov=True)
model.compile(loss='categorical_crossentropy', optimizer=sgd)
return model
```

# 除了卷积以外还应该做什么

max pooling:



对于一张合理的图像而言，临近点之间具有某种“连续性”  
在某种意义上类似于“积分中值定理”：average pooling, max pooling

# 为什么要做 pooling

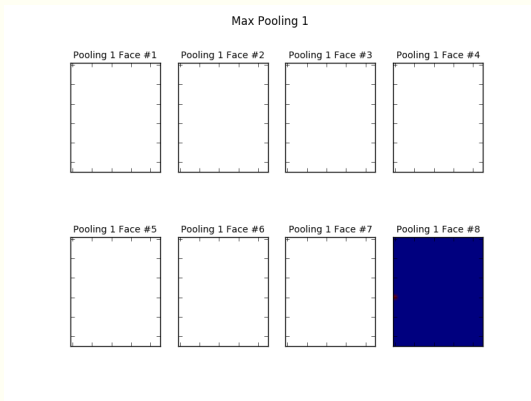
Pooling 的时候，“随意地”丢弃了很多特征图中的信息，所以可以对抗过拟合

如果用同样的方法去理解卷积 filter 呢？

或许，卷积 filter 可以看做一种更加精细的“积分中值定理”（实际上数学上的卷积就是这样的积分）

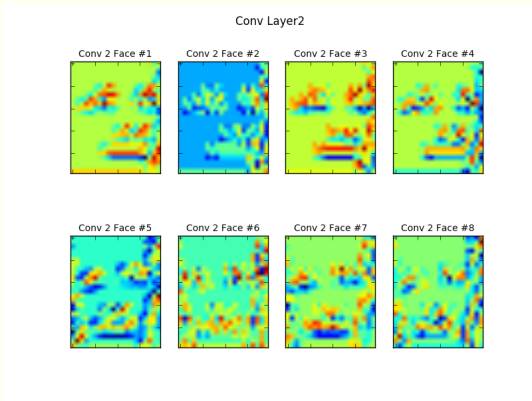
卷积 filter+ 非线性和 Pooling 经常成对出现

# Keras 实验: Max Pooling



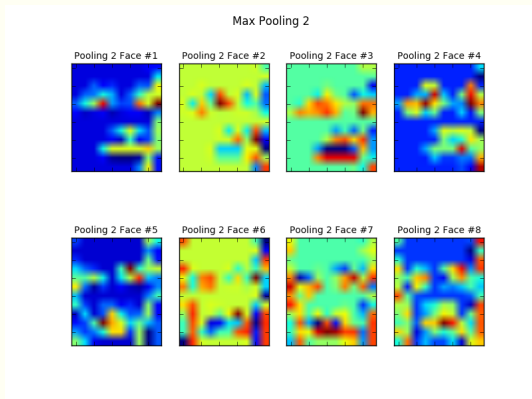


# Keras 实验: 第二层卷积



可以看到, 图片已经丢失了大部分信息  
但是最主要的特征: 眼睛、鼻子、嘴巴被保留了

# Keras 实验: 第二层 Pooling



更加抽象，但是网络已经抓取到了最具有特征的人脸五官信息  
此时再做全连接网络的分类器，计算代价降低了很多

# 影响结果的各种参数

大家都熟知，所以不想讲太多

比如网络层数，层数越高，之前图片的信息“密度”越高，越从“局部感知”走向“全局感知”

我主要想讲的就是之前的原理

# 主成分分析

# PCA 是如何想到的……

PCA 是一个很棒的降维方法！

图片是维度特别高的输入，所以之前我们用 CNN 去抓取特征。

在某些情况下，直接用 PCA 得到最重要的方向！

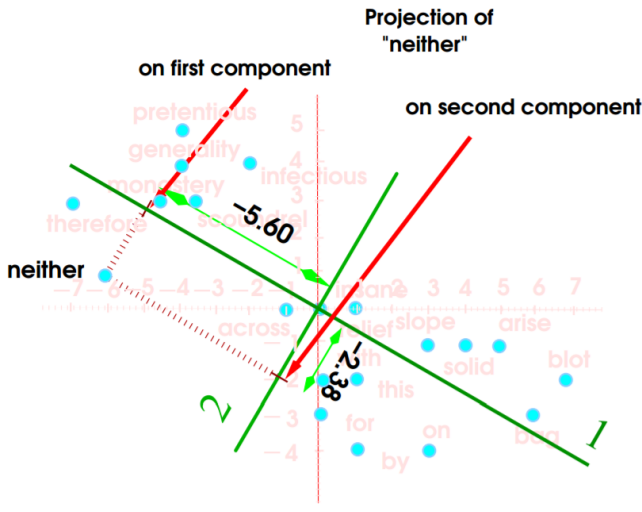
# PCA 的前世今生

源自 1901 年的统计学家：Karl Pearson

PCA 的本质是找到高维空间上，对方差影响最大的数据方向  
这样就能够权衡数据损失与空间维度！

数学上，PCA 是特征分析多元统计分布的最简单的方法  
现在，在复杂数据，比如人脸识别中非常有用

# 一个简单的图示



得到了两个最重要的正交的方向

# 优雅的数学推理

为什么之前那两个方向是最重要的？

人脸  $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\} \in \mathbb{R}^{n \times d}$

$$\mathbf{x}_0 \in \mathbb{R}^d$$

希望它能最好地代表这些脸，或者说：与所有样本之间的距离的平方和越小越好：

$$J_0(\mathbf{x}_0) = \sum_{k=0}^n \|\mathbf{x}_0 - \mathbf{x}_k\|^2$$



# 优雅的数学推理

拆开目标函数：

$$\begin{aligned}J_0(\mathbf{x}_0) &= \sum_{k=1}^n \|(\mathbf{x}_0 - \mu) - (\mathbf{x}_k - \mu)\|^2 \\&= n \cdot \|\mathbf{x}_0 - \mu\|^2 + \sum_{k=0}^n \|\mathbf{x}_k - \mu\|^2 - 2 \cdot \|\mathbf{x}_0 - \mu\| \cdot \sum_{k=0}^n (\mathbf{x}_k - \mu) \\&= n \cdot \|\mathbf{x}_0 - \mu\|^2 + \sum_{k=0}^n \|\mathbf{x}_k - \mu\|^2\end{aligned}$$

右边一项并不依赖于  $\mathbf{x}_0$  是常量，所以左边一项取均值  $\mu$  时目标函数取极小值

# 优雅的数学推理

这是否意味着就应该用均值  $\mu$  代表所有的人脸？  
这样的观点基于这样一种看法，样本均值是数据的一种零维表达：

$$\mu \in \mathbb{R}^0$$

所以当我们用样本均值来降维的时候，降维降得过头了。所以现在考虑降到一维的情况，也就是将样本点投影到一条经过均值点的直线上，让这根直线上的投影点来代表全体数据点：

$$\mathbf{x} = \mu + a \cdot \mathbf{e}$$

在这里，标量  $a$  反应了投影点对均值的偏离程度， $\mathbf{e}$  是单位方向向量

# 优雅的数学推理

现在，我们希望最小化投影点和实际点之间的差异：

$$\|(\mu + a_k \cdot \mathbf{e}) - \mathbf{x}_k\|$$

所以构造平方误差函数：

$$J_1(a_1, \dots, a_n, \mathbf{e}) = \sum_{k=1}^n \|(\mu + a_k \cdot \mathbf{e}) - \mathbf{x}_k\|^2$$

# 优雅的数学推理

展开处理：

$$J_1 = \sum_{k=1}^n a_k^2 - 2 \sum_{k=1}^n a_k \mathbf{e}^T(\mathbf{x}_k - \mu) + \sum_{k=1}^n \|\mathbf{x}_k - \mu\|^2$$

这样就可以对  $a_k$  求偏导：

$$\frac{\partial J_1}{\partial a_k} = 2a_k - 2\mathbf{e}^T(\mathbf{x}_k - \mu) = 0 \Rightarrow a_k = \mathbf{e}^T(\mathbf{x}_k - \mu)$$

# 优雅的数学推理

从几何上讲，这告诉我们向量  $\mathbf{x}_k$  只要向直线  $\mathbf{e}$  做垂直投影，就能得到最小方差。

接下来的问题是：如何找到直线  $\mathbf{e}$  的最优方向？

散布矩阵

$$\mathbf{S} = \sum_{k=1}^n (\mathbf{x}_k - \mu)(\mathbf{x}_k - \mu)^T$$

协方差矩阵的  $n - 1$  倍

# 优雅的数学推理

用散布矩阵来描述目标函数：

$$J_1(\mathbf{e}) = -\mathbf{e}^T \mathbf{S} \mathbf{e} + \sum_{k=1}^n \|\mathbf{x}_k - \mu\|^2$$

用 Lagrange 乘子法求  $\|\mathbf{e}\| = 1$  的条件极值：

$$u = -\mathbf{e}^T \mathbf{S} \mathbf{e} + \lambda (\mathbf{e}^T \mathbf{e} - 1)$$

# 优雅的数学推理

对  $\mathbf{e}$  求偏导:

$$\frac{\partial u}{\partial \mathbf{e}} = 2\mathbf{S}\mathbf{e} - 2\lambda\mathbf{e} = 0$$
$$\Rightarrow \mathbf{S}\mathbf{e} = \lambda\mathbf{e}$$

所以它一定是特征向量!

要让  $\mathbf{e}^T \mathbf{S} \mathbf{e} = \lambda \mathbf{e}^T \mathbf{e} = \lambda$  最大

就要选取散布矩阵最大的特征值所对应的特征向量作为方向向量

# 优雅的数学推理

如果映射到高维空间  $d'$  呢？

$$\mathbf{x} = \mu + \sum_{i=1}^{d'} a_i \mathbf{e}_i$$

取散布矩阵前  $d'$  个最大的特征值所对应的特征向量就行了

另外，散布矩阵

$$\mathbf{S} = \sum_{k=1}^n (\mathbf{x}_k - \mu)(\mathbf{x}_k - \mu)^T$$

实对称，所以这些特征向量相互正交



# numpy 实验

还是用 AT&T 的 400 张人脸灰度图像  
因为数据集简单，所以虽然是简单模型，效果也很好，正确率在 99% 左右

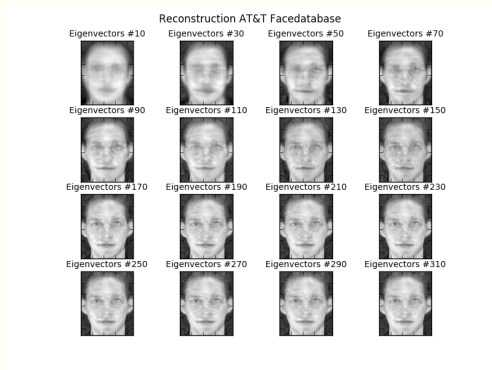
# numpy 实验

```
W,  $\mu$  = PCA(X);  
for  $x_i \in \mathbf{X}$  do  
    | P  $\oplus$  Projection(W,  $x_i$ ,  $\mu$ );  
end  
Q = Projection(W,  $x_i$ ,  $\mu$ );  
for  $i \in \text{len}(\mathbf{P})$  do  
    |  $dist = ||\mathbf{P}[i], \mathbf{Q}||$ ;  
    | if  $dist < dist_{min}$  then  
        | |  $dist_{min} = dist$ ;  
        | |  $class_{min} = y_i$ ;  
    | end  
end
```

**Algorithm 1:** Prediction Algorithm of PCA

# numpy 实验: 维度

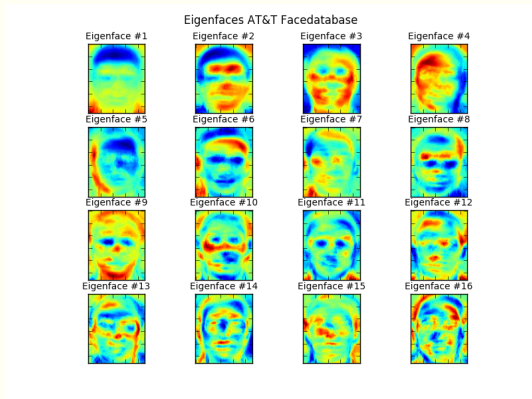
中间的结果也比较有趣，下图是一张脸在不同数量的特征向量上投影的结果：



PCA 投影的维度越高当然越接近原图

# numpy 实验: Eigen Face

从整个训练集中得到特征向量的矩阵  $\mathbf{M} \in \mathbb{R}^{(H \cdot W) \times d'}$   
取 16 个特征向量，归一化到  $[0, 255]$ ，还原成图片：



既成熟又不成熟的……

# 其他方法的一点简单介绍

- ❖ 古老的几何方法  
计算眼耳鼻等面部特征之间的几何关系，在一个 20 人数据库上识别率为 45% ~ 75%
- ❖ 隐 Markov 模型  
在垂直和水平方向（因为人脸会有稳定的结构）上，使用 2 维离散余弦变换系数作为特征……
- ❖ 除了 PCA, CNN 之外的统计模型  
Fisher 判别法

# 更复杂的课题

## ❖ 光照变化

光照变化带来巨大的灰度相对分布，所以光照引起的变化甚至可能打过个体差异。

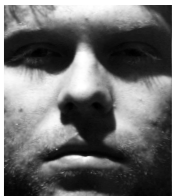
寻找不受光照影响的表达方式；改进现有算法（丢弃光照主成分）；或者合成全面的人脸模型

## ❖ 姿态变化

用 2 维投影线性组合重构 3 维对象的构想（流形）

# 光照变化实验

使用了 Yale 的扩展人脸数据集：



具有光照变化



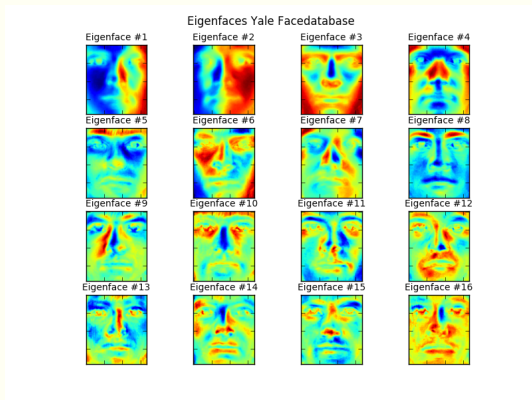
# 光照变化实验: PCA 的表现

```
loading numpy array from ./yalefaces/dataset.npy ...  
(15, 10, 32256) (15,)  
expected = 0 / predicted = 0  
expected = 1 / predicted = 1  
expected = 2 / predicted = 2  
expected = 3 / predicted = 0  
expected = 4 / predicted = 4  
expected = 5 / predicted = 8  
expected = 6 / predicted = 6  
expected = 7 / predicted = 4  
expected = 8 / predicted = 8  
expected = 9 / predicted = 9  
expected = 10 / predicted = 4  
expected = 11 / predicted = 10  
expected = 12 / predicted = 12  
expected = 13 / predicted = 13  
expected = 14 / predicted = 14
```

明显比 AT&T 数据集的表现要差一点。在 AT&T 数据集上的识别率高达 99%，在具有光照变化的 Yale 数据集上降到了 66%

# 光照变化实验: PCA 的表现

将 PCA 的特征脸抽取出来



明显可以看到前几个特征向量收到光照的影响

# 光照变化实验: CNN 的表现

CNN 的表现要好很多,基本上依然维持在全部正确识别的水平

```
Usage:
newton > 0    - save model
newton > 1    - choose one picture from each subject as test
newton > 2    - draw pictures
newton > exit - exit python

newton > 1
Test results:
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
 25 26 27 28 29]
```

这里两个模型都没有进行调整,所以 PCA 可能包含了具有光照效果的特征向量,导致识别率降低

可以看到, CNN 对光照变化是比较不敏感的

这可能得益于 CNN 在每层的抽象上过滤了光照信息

# 更复杂的应用场景

针对性的人脸识别效果很好，但是更宽泛稳健的人脸识别系统还没有完善好

非常难以解决的问题是，当一个人戴上口罩、墨镜的时候，他自己的差异甚至大过与其他人之间的差异

要建立完善的人脸识别系统，图像的预处理占据了极大的比重

# 参考资料

- ❏ Stanford, CS231n, Computer Vision
- ❏ [github.com/bytedfish/facerecognition](https://github.com/bytedfish/facerecognition) guide
- ❏ Keras, <https://keras.io/>
- ❏ FBI, <https://www.fbi.gov>
- ❏ AT&T 数据集
- ❏ Yale 数据集