每周学习进展阶段汇报

---CS131 Stanford University 计算机视觉基础课程

汇报人：胡小婉

时间段：2018年2月20日至2018年8日26日

Lecture #17: Motion

Lecture 18: Tracking

Lecture 19 :Introduction to Deep Learning

CS131 Computer Vision: Foundations and Applications

Practice Final (Solution)

Lecture #17: Motion

1. Optical Flow and Key Assumptions

1.1 Introduction

光流（optical flow）是空间运动物体在观测成像面上的像素运动的瞬时速度。光流的研究是利用图像序列中的像素强度数据的时域变化和相关性来确定各自像素位置的“运动”，即研究图像灰度在时间上的变化与景象中物体结构及其运动的关系。将二维图像平面特定坐标点上的灰度瞬时变化率定义为光流矢量。

光流场（optical flow field）是指图像灰度模式的表观运动。它是一个二维矢量场，它包含的信息即是各像点的瞬时运动速度矢量信息。研究光流场的目的就是为了从序列图像中近似计算不能直接得到的运动场。

当人的眼睛观察运动物体时，物体的景象在人眼的视网膜上形成一系列连续变化的图像，这一系列连续变化的信息不断“流过”视网膜（即图像平面），好像一种光的“流”，故称之为光流（optical flow）。光流表达了图像的变化，由于它包含了目标运动的信息，因此可被观察者用来确定目标的运动情况。

1.2 算法

一般而言，光流是由于场景中前景目标本身的移动、相机的运动，或者两者的共同运动所产生的。其计算方法可以分为三类：

（1）基于区域或者基于特征的匹配方法；

（2）基于频域的方法；

（3）基于梯度的方法；

简单来说，光流是空间运动物体在观测成像平面上的像素运动的“瞬时速度”。光流的研究是利用图像序列中的像素强度数据的时域变化和相关性来确定各自像素位置的“运动”。研究光流场的目的就是为了从图片序列中近似得到不能直接得到的运动场

1.3 基本实现

光流法的前提假设：

（1）相邻帧之间的亮度恒定；

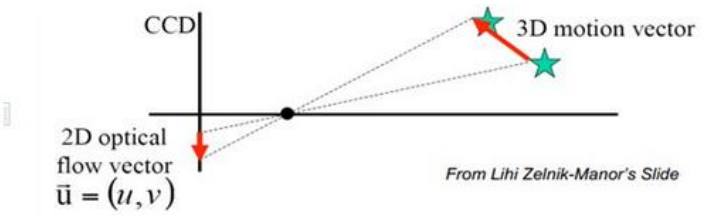
（2）相邻视频帧的取帧时间连续，或者，相邻帧之间物体的运动比较“微小”；

（3）保持空间一致性；即，同一子图像的像素点具有相同的运动

这里有两个概念需要解释：

运动场，其实就是物体在三维真实世界中的运动；

光流场，是运动场在二维图像平面上的投影



光流场的目的是找到图片中每个像素点的速度向量：$\bar{u} = (u, v)$，需要注意的是，这里的速度是个矢量，不仅有运动大小信息，还包括了运动的方向信息。

根据前面提到的光流的微小运动和亮度恒定这两个假设，我们可以得到：

$$I(x, y, t) = I(x + dx, y + dy, t + dt)$$

该式用一阶泰勒级数展开，我们将得到：

$$I(x + dx, y + dy, t + dt) = I(x, y, t) + \frac{\partial I}{\partial x}dx + \frac{\partial I}{\partial y}dy + \frac{\partial I}{\partial t}dt$$

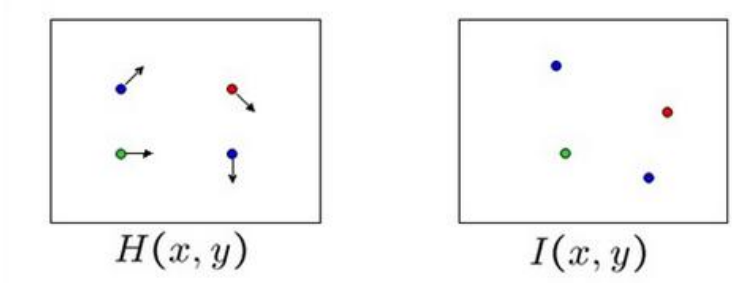
即：$I_x dx + I_y dy + I_t dt = 0$，令 $u = \frac{dx}{dt}$，$v = \frac{dy}{dt}$，那么：

$I_x u + I_y v = -I_t$，即：$[I_x I_y] \bullet \begin{bmatrix} u \\ v \end{bmatrix} = -I_t$，假设在（u,v）的的一个小的局部领域内，亮度是恒定的，那么：$\begin{bmatrix} I_{x1} & I_{y1} \\ I_{x2} & I_{y2} \\ \vdots & \vdots \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} I_{t1} \\ I_{t2} \\ \vdots \end{bmatrix}$，即：$A\bar{u} = b$

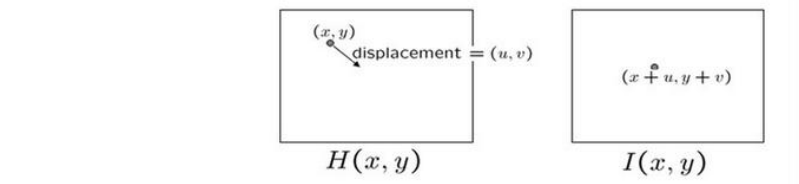
光流计算的目的，就是使得 $\|A\bar{u} - b\|^2$ 最小，

$$A\bar{u} = b \quad \Rightarrow \quad A^T A\bar{u} = A^T b \quad \Rightarrow \quad \bar{u} = (A^T A)^{-1} A^T b$$

$$A^T A = \begin{bmatrix} \sum I_x^2 & \sum I_x I_y \\ \sum I_x I_y & \sum I_y^2 \end{bmatrix}$$



如上图所示，假设已有 H(x,y)，I(x,y)两个图片，如何计算 H 到 I 中像素点之间的运动？显然，对于 H 中特定的像素点，我们应该在 I 图片中对应位置的周围来寻找像素值一致或者接近一致的像素点。如此一来，不难发现，解决此类问题，一般有两个关键的假设：
（1）颜色一致（对灰度图而言，可理解为亮度一致）；
（2）微小运动；即，每个像素点，都不会产生较大的运动偏移。
显然，这种问题，可以用光流方法来解决。



如上图所示，H 中的像素点(x,y)在 I 中的移动到了(x+u,y+v)的位置，偏移量为(u,v)。

2. Lucas-Kanade

2.1 Introduction

这个算法是最常见，最流行的。它计算两帧在时间 t 到 t + δt 之间每个每个像素点位置的移动。 由于它是基于图像信号的泰勒级数，这种方法称为差分，这就是对于空间和时间坐标使用偏导数

2.2 算法实现

2.2.1 假设条件

（1）亮度恒定，就是同一点随着时间的变化，其亮度不会发生改变。这是基本光流法的假定（所有光流法变种都必须满足），用于得到光流法基本方程；

（2）小运动，这个也必须满足，就是时间的变化不会引起位置的剧烈变化，这样灰度才能对位置求偏导（换句话说，小运动情况下我们才能用前后帧之间单位位置变化引起的灰度变化去近似灰度对位置的偏导数），这也是光流法不可或缺的假定；

（3）空间一致，一个场景上邻近的点投影到图像上也是邻近点，且邻近点速度一致。这是 Lucas-Kanade 光流法特有的假定，因为光流法基本方程约束只有一个，而要求 x，y 方向的速度，有两个未知变量。我们假定特征点邻域内做相似运动，就可以连立 n 多个方程求取 x，y 方向的速度（n 为特征点邻域总点数，包括该特征点）

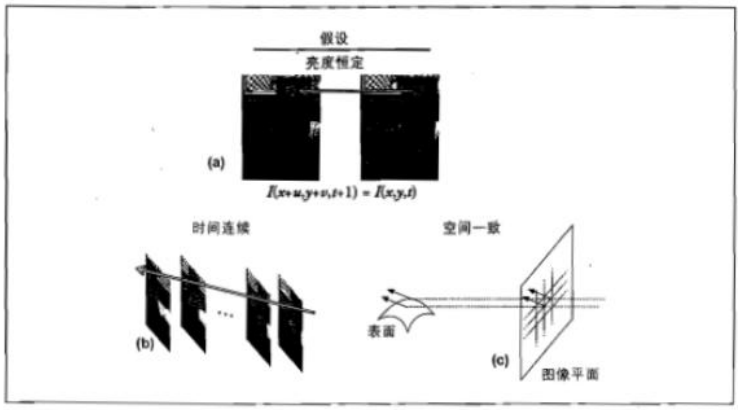


图 10-4：Lucas-Kanade 光流假设：(a)场景中物体被跟踪的部分的亮度不变；(b)运动相对于帧率是缓慢的；(c)[相邻的点保持相邻(图由 Michael Black [Black82]提供)] 【324】

2.2.2 方程求解

图像约束方程可以写为 I（x ,y ,z ,t ）= I（x + δx ,y + δy ,z + δz ,t + δt ）

I(x, y,z, t) 为在（x,y,z）位置的体素。

我们假设移动足够的小，那么对图像约束方程使用泰勒公式，我们可以得到：

$$I(x+\delta x, y+\delta y, z+\delta z, t+\delta t) = I(x,y,z,t)+\frac{\partial I}{\partial x}\delta x+\frac{\partial I}{\partial y}\delta y+\frac{\partial I}{\partial z}\delta z+\frac{\partial I}{\partial t}\delta t+H.O.T.$$

H.O.T. 指更高阶，在移动足够小的情况下可以忽略。从这个方程中我们可以得到：

$$\frac{\partial I}{\partial x}\delta x+\frac{\partial I}{\partial y}\delta y+\frac{\partial I}{\partial z}\delta z+\frac{\partial I}{\partial t}\delta t=0$$

$$\frac{\partial I}{\partial x}V_x+\frac{\partial I}{\partial y}V_y+\frac{\partial I}{\partial z}V_z+\frac{\partial I}{\partial t}=0$$

V x ,V y ,V z 分别是 I(x,y,z,t)的光流向量中 x，y，z 的组成。

所以

I x V x + I y V y + I z V z = − I t。

写做：$\nabla I^T \cdot \vec{V} = -I_t$

这个方程有三个未知量，尚不能被解决，这也就是所谓光流算法的光圈问题。那么要找到光流向量则需要另一套解决的方案。而 Lucas-Kanade 算法是一个非迭代的算法：

假设流(Vx,Vy,Vz)在一个大小为 m*m*m(m>1)的小窗中是一个常数，那么从像素 1...n ，n = m 3 中可以得到下列一组方程：(也就是说，对于这多个点，它们三个方向的速度是一样的)即，假设了在一个像素的周围的一个小的窗口内的所有像素点的光流大小是相同的。

$$A = \begin{bmatrix} I_x(q_1) & I_y(q_1) \\ I_x(q_2) & I_y(q_2) \\ \vdots & \vdots \\ I_x(q_n) & I_y(q_n) \end{bmatrix}, \quad v = \begin{bmatrix} V_x \\ V_y \end{bmatrix}, \quad \text{and} \quad b = \begin{bmatrix} -I_t(q_1) \\ -I_t(q_2) \\ \vdots \\ -I_t(q_n) \end{bmatrix}$$

http://blog.csdn.net/xu

)

$$I_{x_1}V_x + I_{y_1}V_y + I_{z_1}V_z = -I_{t_1}$$

$$I_{x_2}V_x + I_{y_2}V_y + I_{z_2}V_z = -I_{t_2}$$

⋮

$$I_{x_n}V_x + I_{y_n}V_y + I_{z_n}V_z = -I_{t_n}$$

三个未知数但是有多于三个的方程，这个方程组自然是个超定方程，也就是说方程组内有冗余，方程组可以表示为：

$$\begin{bmatrix} I_{x1} & I_{y1} & I_{z1} \\ I_{x2} & I_{y2} & I_{z2} \\ \vdots & \vdots & \vdots \\ I_{x_n} & I_{y_n} & I_{z_n} \end{bmatrix} \begin{bmatrix} V_x \\ V_y \\ V_z \end{bmatrix} = \begin{bmatrix} -I_{t1} \\ -I_{t2} \\ \vdots \\ -I_{t_n} \end{bmatrix}$$

记作 $A\vec{v} = -b$ 为了解决这个超定问题，我们采用最小二乘法：

$$A^T A\vec{v} = A^T(-b)$$

or

$$\vec{v} = (A^T A)^{-1} A^T(-b)$$

$$\begin{bmatrix} V_x \\ V_y \\ V_z \end{bmatrix} = \begin{bmatrix} \sum I_{x_i}^2 & \sum I_{x_i} I_{y_i} & \sum I_{x_i} I_{z_i} \\ \sum I_{x_i} I_{y_i} & \sum I_{y_i}^2 & \sum I_{y_i} I_{z_i} \\ \sum I_{x_i} I_{z_i} & \sum I_{y_i} I_{z_i} & \sum I_{z_i}^2 \end{bmatrix}^{-1} \begin{bmatrix} -\sum I_{x_i} I_{t_i} \\ -\sum I_{y_i} I_{t_i} \\ -\sum I_{z_i} I_{t_i} \end{bmatrix}$$

其中的求和是从 1 到 n。

这也就是说寻找光流可以通过在四维上图像导数的分别累加得出。我们还需要一个权重函数 W(i,j,k) ， $i,j,k \in [1,m]$ 来突出窗口中心点的坐标。高斯函数做这项工作是非常合适的，

这个算法的不足在于它不能产生一个密度很高的流向量，例如在运动的边缘和黑大的同质区域中的微小移动方面流信息会很快的褪去。它的优点在于有噪声存在的鲁棒性还是可以的。

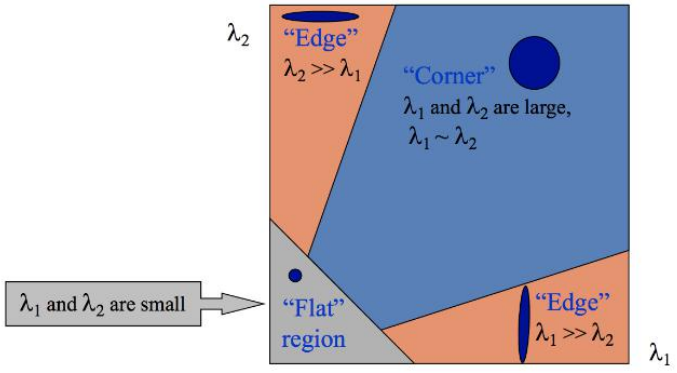


Figure 2: Conditions for a solvable matrix $A^T A$ may be interpreted as different edge regions depending on the relation between λ_1 and λ_2. Corner regions produce more optimal conditions.

3．LK 光流提取算法

3.1 光流法的第一层推导

基本假设：光流在像素点的邻域是一个常数
然后使用最小二乘法对邻域中的所有像素点求解基本的光流方程。因为方程个数已经超过了为未知量个数
假定像素位置p周围的领域像素由$q_1 q_2 q_3 \ldots$，那么就有
$I_x(q_1)V_x + I_y(q_1)V_y + I_t(q_1) = 0$ ；
$I_x(q_2)V_x + I_y(q_2)V_y + I_t(q_2) = 0$ ；
$I_x(q_3)V_x + I_y(q_3)V_y + I_t(q_3) = 0$ ；
…
将其写为矩阵的形式，则有

$$A = \begin{bmatrix} I_x(q_1) & I_y(q_1) \\ I_x(q_2) & I_y(q_2) \\ \vdots & \vdots \\ I_x(q_n) & I_y(q_n) \end{bmatrix}, \quad v = \begin{bmatrix} V_x \\ V_y \end{bmatrix}, \quad \text{and} \quad b = \begin{bmatrix} -I_t(q_1) \\ -I_t(q_2) \\ \vdots \\ -I_t(q_n) \end{bmatrix}$$

然后通过伪逆的形式解最小二乘就可以求解到光流v
$v = (A^T A)^{-1} A^T b$

Tips：L-K 光流算法的缺点

我们首先要设置一个邻域的窗口，之后计算光流
当窗口较大时，光流计算更鲁棒
当窗口较小时，光流计算更正确
原因在于，当图像中每一个部分的运动都不一致的时候，如果开的窗口过大，很容易违背假设：窗口（邻域）内的所有点光流一致，这可能与实际不一致，所以窗口小，包含的像素少，更精确些。当运动较为剧烈的时候，无法实现光流的基本假设，
即$I(x+\Delta x, y+\Delta y, t+\Delta t) = I(x, y, t)$,因为Δx Δy 而我们的领域小于实际运动的位移，所以当我们的领域加大的时候，算法更鲁棒。

如何解决这一问题，我们采用了金字塔的方法，即窗口固定，将图像生成金字塔，在每一层金字塔上都用同一个大小的窗口来进行光流计算。那么在图像大小较小的时候，窗口显得较大，此时的光流可以跟踪速度较快的目标，而在原始图像的时候，光流窗口相对较小，得到的光流就更准确。这是一个由粗到精的过程。

3.2 光流法的第二层推导

光流法的基本方程

$$\epsilon(\mathbf{d}) = \epsilon(d_x, d_y) = \sum_{x=u_x-\omega_x}^{u_x+\omega_x} \sum_{y=u_y-\omega_y}^{u_y+\omega_y} (I(x,y) - J(x+d_x, y+d_y))^2$$

其中$I(x, y)$ 指原图，$J(x, y)$ 指参考图。d_x, d_y为在(u_x, u_y)处的光流向量。w_x, w_y为基于坐标点(u_x, u_y)的搜索窗口（半径）。

默认情况下，我们认为d_x, d_y应该小于(u_x, u_y)，也就是在I图中的一个点对应J图时，应该不会超过(u_x, u_y)这个窗口的。那么基于这个假设就出现了上面提到的关于窗口大小与光流准确性及鲁棒性带来的矛盾。下面讲如何利用图像金字塔来解决这个矛盾。

3.2.1 如何建立图像金字塔

$$
\begin{aligned}
I^L(x,y) = & \frac{1}{4}I^{L-1}(2x,2y) + \\
& \frac{1}{8}\left(I^{L-1}(2x-1,2y) + I^{L-1}(2x+1,2y) + I^{L-1}(2x,2y-1) + I^{L-1}(2x,2y+1)\right) + \\
& \frac{1}{16}\left(I^{L-1}(2x-1,2y-1) + I^{L-1}(2x+1,2y+1) + I^{L-1}(2x-1,2y+1) + I^{L-1}(2x+1,2y+1)\right)
\end{aligned}
$$

按照此公式建立，其中I^L代笔第L层的图像金字塔。初始情况下使I^0为原始图像。按照上述公司，由第L-1层图像到第L层图像是通过以下步骤

- 对L-1层图像进行低通滤波，滤波模板如下

$\frac{1}{16}$	$\frac{1}{8}$	$\frac{1}{16}$
$\frac{1}{8}$	$\frac{1}{4}$	$\frac{1}{8}$
$\frac{1}{16}$	$\frac{1}{8}$	$\frac{1}{16}$

- 将滤波后的图像抽取偶数行偶数列的像素重组为第L层图像，因此最后L-1层的图像大小是L层的一半。

3.2.2 根据金字塔逐层计算光流

首先引入两个光流量：剩余光流量d^L和猜测光流量g^L。

核心思想在于：对第L层的光流，是通过由第L-1层的精确光流得到的导出光流量g^L以及这一层以g^L为基准再次进行匹配微调后得到的剩余光流量d^L。即第L层的光流k^L为

$$k^L = g^L + d^L$$

g^L由上一层的精确光流猜测得来

$$g^L = 2 \times k^{L-1} = 2 \times (g^{L-1} + d^{L-1})$$

基于的事实是：既然第L-1到第L层是降采样一半，那么光流量也应该减一半。

但是这样粗略的估算是不准确的，所以第L层还需要通过基本的光流方程得出剩余光流量d^L去修正这个值，使得第L层的光流更精确.

怎么得到d^L，就是基本的光流方程

$$\epsilon^L(\mathbf{d}^L) = \epsilon^L(d_x^L, d_y^L) = \sum_{x=u_x^L-\omega_x}^{u_x^L+\omega_x}\sum_{y=u_y^L-\omega_y}^{u_y^L+\omega_y}\left(I^L(x,y) - J^L(x+g_x^L+d_x^L, y+g_y^L+d_y^L)\right)^2$$

最后得到的原始图像的光流按照以下公式计算而来 $\mathbf{d} = \sum_{L=0}^{L_m} 2^L \mathbf{d}^L$

3.2.3 计算光流基本方程

首先光流基本方程是一个最优化的问题，可以通过求导求其最优解

$$\varepsilon(\overline{\nu}) = \varepsilon(\nu_x, \nu_y) = \sum_{x=p_x-\omega_x}^{p_x+\omega_x} \sum_{y=p_y-\omega_y}^{p_y+\omega_y} (A(x,y) - B(x+\nu_x, y+\nu_y))^2.$$

对基本光流方程求导

$$\left.\frac{\partial \varepsilon(\overline{\nu})}{\partial \overline{\nu}}\right|_{\overline{\nu}=\overline{\nu}_{\text{opt}}} = [0 \;\; 0].$$

$$\frac{\partial \varepsilon(\overline{\nu})}{\partial \overline{\nu}} = -2 \sum_{x=p_x-\omega_x}^{p_x+\omega_x} \sum_{y=p_y-\omega_y}^{p_y+\omega_y} (A(x,y) - B(x+\nu_x, y+\nu_y)) \cdot \begin{bmatrix} \frac{\partial B}{\partial x} & \frac{\partial B}{\partial y} \end{bmatrix}$$

对$B(x+v_x, y+v_y)$ 进行一阶泰勒展开

$$\frac{\partial \varepsilon(\overline{\nu})}{\partial \overline{\nu}} \approx -2 \sum_{x=p_x-\omega_x}^{p_x+\omega_x} \sum_{y=p_y-\omega_y}^{p_y+\omega_y} \left(A(x,y) - B(x,y) - \begin{bmatrix} \frac{\partial B}{\partial x} & \frac{\partial B}{\partial y} \end{bmatrix} \overline{\nu}\right) \cdot \begin{bmatrix} \frac{\partial B}{\partial x} & \frac{\partial B}{\partial y} \end{bmatrix}$$

> 计算两张图像的差
>
> $$\delta I(x,y) \doteq A(x,y) - B(x,y).$$
>
> 计算图像在w_x, w_y邻域内的差分，这里可以采用Sharr算子代表差分
>
> $$\nabla I = \begin{bmatrix} I_x \\ I_y \end{bmatrix} \doteq \begin{bmatrix} \frac{\partial B}{\partial x} & \frac{\partial B}{\partial y} \end{bmatrix}^T$$

$$G \doteq \sum_{x=p_x-\omega_x}^{p_x+\omega_x} \sum_{y=p_y-\omega_y}^{p_y+\omega_y} \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} \quad \text{and} \quad \overline{b} \doteq \sum_{x=p_x-\omega_x}^{p_x+\omega_x} \sum_{y=p_y-\omega_y}^{p_y+\omega_y} \begin{bmatrix} \delta I \, I_x \\ \delta I \, I_y \end{bmatrix}$$

$$\overline{\nu}_{\text{opt}} = G^{-1} \overline{b}.$$

核心思想在于：在上一次计算出 LK 光流后，将得到的光流 vk 用来更新参考图像的像素位置（平移），之后再次计算光流。重新更新 vk。如此循环直到迭代次数达到上限或者误差量 ε 小于设定的阈值时停止。 按照上述情况来看，停止迭代后，A 图和 B 图两张图在(x,y)点处像素完全重合的

$$B_k(x,y) = B(x+\nu_x^{k-1}, y+\nu_y^{k-1}).$$

$$\varepsilon^k(\overline{\eta}^k) = \varepsilon(\eta_x^k, \eta_y^k) = \sum_{x=p_x-\omega_x}^{p_x+\omega_x} \sum_{y=p_y-\omega_y}^{p_y+\omega_y} \left(A(x,y) - B_k(x+\eta_x^k, y+\eta_y^k)\right)^2$$

$$\overline{\eta}^k = G^{-1}\,\overline{b}_k,$$

这里是最终迭代结束后的精确光流量

$$\overline{\nu} = \mathbf{d}^L = \overline{\nu}^K = \sum_{k=1}^{K} \overline{\eta}^k$$

4. Horn-Schunk

$$E = \int\int [(I_x u + I_y v + I_t)^2 + \alpha^2(||\nabla u||^2 + ||\nabla v||^2)]\mathrm{d}x\mathrm{d}y$$

To minimize the energy function, we take the derivative with respect to u and v and set to zero. This yields the following two equations

$$I_x(I_x u + I_y v + I_t) - \alpha^2 \Delta u = 0$$
$$I_y(I_x u + I_y v + I_t) - \alpha^2 \Delta v = 0$$

where $\Delta = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}$ is called the Lagrange operator, which in practice is computed as

$$\Delta u(x,y) = \bar{u}(x,y) - u(x,y)$$

where $\bar{u}(x,y)$ is the weighted average of u in a neighborhood around (x,y). Substituting this expression for the Lagrangian in the two equations above yields

$$(I_x^2 + \alpha^2)u + I_x I_y v = \alpha^2 \bar{u} - I_x I_t$$
$$I_x I_y u + (I_y^2 + \alpha^2)v = \alpha^2 \bar{v} - I_y I_t$$

which is a linear equation in u and v for each pixel.

Since the solution for u and v for each pixel (x,y) depends on the optical flow values in a neighborhood around (x,y), to obtain accurate values for u and v we must recalculate u and v iteratively once the neighbors have been updated. We can iteratively solve for u and v using

$$u^{k+1} = \bar{u}^k - \frac{I_x(I_x\bar{u}^k + I_y\bar{v}^k + I_t)}{\alpha^2 + I_x^2 + I_y^2}$$
$$v^{k+1} = \bar{v}^k - \frac{I_y(I_x\bar{u}^k + I_y\bar{v}^k + I_t)}{\alpha^2 + I_x^2 + I_y^2}$$

where $\bar{u}^k$ and $\bar{v}^k$ are the values for $\bar{u}$ and $\bar{v}$ calculated during the k'th iteration, and u^{k+1} and v^{k+1} are the updated values for u and v for the next iteration.

4.1 Pyramids for Large Motion

4.1.1 Introduction

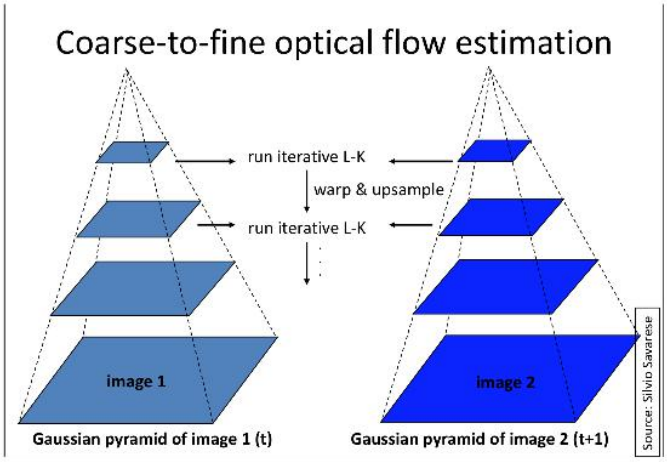
大而不连贯的运动是普遍存在的情况,而 Lucas-Kanade 光流正因为这个原因在实际的跟踪效果中并不是很好,我们需要一个大的窗口来捕获大的运动,而大窗口往往会违背运动连贯性的假设,图像金字塔可以解决这个问题,即最初在较大的空间尺度上进行跟踪,再通过对图像金字塔向下直到图像像素的处理来修正初始运动速度的假定.

所以,建议的跟踪方法是,在图像金字塔的最高层计算光流,用得到的结果作为下一层金字塔的起点.重复这个过程直到到达金字塔的最底层,这样就把不满足运动假设的可能性降到最小从而实现对更快和更长的运动的跟踪.

为什么要用金字塔？因为 1k 算法的约束条件即：小速度，亮度不变以及区域一致性都是较强的假设，并不很容易得到满足。如当物体运动速度较快时，假设不成立，那么后续的假设就会有较大的偏差，使得最终求出的光流值有较大的误差。

考虑物体的运动速度较大时，算法会出现较大的误差。那么就希望能减少图像中物体的运动速度。一个直观的方法就是，缩小图像的尺寸，假设当图像为 400×400 时，物体速度为[16 16]，那么图像缩小为 200×200 时，速度变为[8,8]。缩小为 100*100 时，速度减少到[4,4]。所以在源图像缩放了很多以后，原算法又变得适用了。所以光流可以通过生成 原图像的金字塔图像，逐层求解，不断精确来求得。简单来说上层金字塔（低分辨率）中的一个像素可以代表下层的两个。

4.1.2 算法实现



Now, when we try to find the flow vector, the small motion condition is fulfilled, as the downsampled pixels move less from frame to consecutive frame than pixels in the higher resolution image. Here is another example from the slides using Lucas-Kanade with pyramids:

首先，光流和仿射变换矩阵在最高一层的图像上计算出；将上一层的计算结果作为初始值传递给下一层图像，这一层的图像在这个初始值的基础上，计算这一层的光流和仿射变化矩阵；

再将这一层的光流和仿射矩阵作为初始值传递给下一层图像，直到传递给最后一层，即原始图像层，这一层计算出来的光流和仿射变换矩阵作为最后的光流和仿射变换矩阵的结果

参考网址： http://blog.csdn.net/u014568921/article/details/46638557

算法流程

图像不匹配向量：$\bar{\mathbf{b}}_k \doteq \sum_{x=p_x-w_x}^{p_x+w_x}\sum_{y=p_y-w_x}^{p_y+w_x}\begin{bmatrix}\delta I_k(x,y)I_x(x,y)\\ \delta I_k(x,y)I_y(x,y)\end{bmatrix}$

L-K 光流：$\bar{\boldsymbol{\eta}}^k = G^{-1}\bar{\mathbf{b}}_k$

估计下一次迭代：$\bar{\mathbf{v}}^k = \bar{\mathbf{v}}^{k-1} + \bar{\boldsymbol{\eta}}^k$

End

在第 L 层上的最终光流：$\mathbf{d}^L = \bar{\mathbf{v}}^K$

计算下一层 $L-1$ 层上的光流：$\mathbf{g}^{L-1} = \begin{bmatrix} g_x^{L-1} & g_y^{L-1}\end{bmatrix}^T = 2(\mathbf{g}^L + \mathbf{d}^L)$

End

最后的光流矢量：$\mathbf{d} = \mathbf{g}^0 + \mathbf{d}^0$

在图像 J 上的对应特征点：$\mathbf{v} = \mathbf{u} + \mathbf{d}$

上述流程解决了两幅图像间特征点跟踪问题。

5. Common Fate

通过共同命运的镜头分析，我们可以获得更多关于图像的信息，在这种情况下，图像的给定片段中的每个像素将以相似的方式移动方式。我们的目标是确定一起移动的图像片段或“图层”.

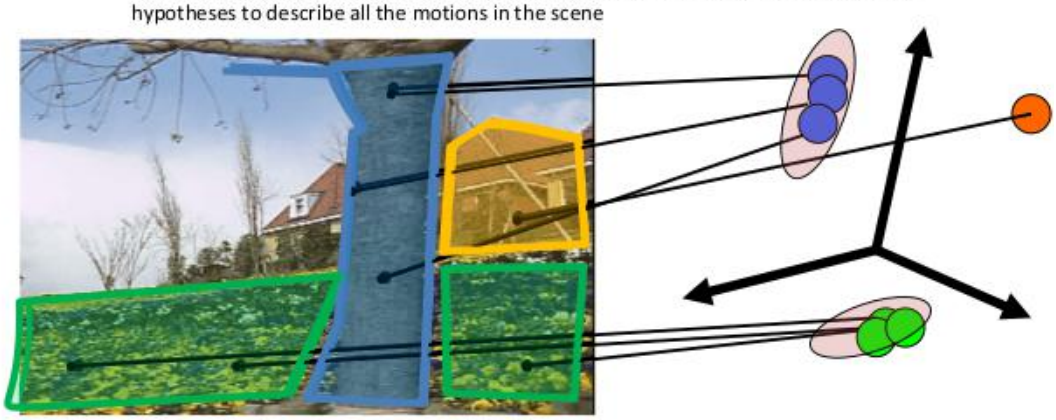
我们通过将图像分成块并根据仿射运动参数的相似性进行分组来计算图像中的图层。对于每个块，找到最小化的向量 a

$$Err(a) = \sum [I_x(a_1 + a_2x + a_3y) + I_y(a_4 + a_5x + a_6y) + I_t]^2$$

从那里，我们将我们的参数向量 a i 映射到运动参数空间中并且对仿射运动参数向量执行 k 均值聚类。

k 均值聚类的最后中心是使上述误差最小化的参数 a 1 ... a 6 函数，并且每个分组中的向量 a i 对应于应该分组的原始块在单一层。直观地说，图层应该由具有相似参数的块组成，这意味着它们的仿射运动是相似的。

- 1. Obtain a set of initial affine motion hypotheses
  - Divide the image into blocks and estimate affine motion parameters in each block by least squares
    - Eliminate hypotheses with high residual error
  - Map into motion parameter space
  - Perform k-means clustering on affine motion parameters
    - Merge clusters that are close and retain the largest clusters to obtain a smaller set of hypotheses to describe all the motions in the scene



Lecture 18: Tracking

1. Simple Kanade—Lucas—Tomasi feature tracker(KLT)

1.1 Introduction

KLT 属于光流法的一种，其前提假设与上一致

1）亮度恒定

2）时间连续或者是运动是“小运动”

3）空间一致，临近点有相似运动，保持相邻

KLT 角点检测方法最初是用于满足 Lucas-Kanade 光流法选择合适特征点的需求，Lucas-Kanade 光流法是通过先在前后两帧图像里分别建立一个固定大小窗口，然后找到让两个窗口间像素强度差的平方和最小的位移。然后将窗口内像素的移动近似为这样的位移向量，然后实际上，一方面像素移动并不会那么简单，另一方面窗口内像素并不都是同样的移动方式，因为这样的近似必然会带来误差。而现在的问题就是如何去选择合适的窗口，或者特征点，从而获得最为精确的跟踪。KLT 角点检测方法就是为了选择一个适合跟踪的特征点，它认为一个好的特征点的定义应当就是能被好的跟踪

1.2 实现步骤

（1）像素点的光强度函数可以由其泰勒展开式表示，g 指强度梯度，d 指像素位移

$$I(\mathbf{x}-\mathbf{d}) = I(\mathbf{x}) - \mathbf{g}\cdot\mathbf{d}$$

（2）选择 d 使窗口内偏差能量最小，即使其导数为 0

$$\epsilon = \int_{\mathcal{W}} [I(\mathbf{x}) - \mathbf{g}\cdot\mathbf{d} - J(\mathbf{x})]^2 w\, d\mathbf{x} = \int_{\mathcal{W}} (h - \mathbf{g}\cdot\mathbf{d})^2 w\, d\mathbf{x}$$

$$\int_{\mathcal{W}} (h - \mathbf{g}\cdot\mathbf{d})\mathbf{g}\, w\, dA = 0$$

$$\left(\int_{\mathcal{W}} \mathbf{g}\mathbf{g}^T w\, dA\right)\mathbf{d} = \int_{\mathcal{W}} h\mathbf{g} w\, dA$$

令

$$G = \int_{\mathcal{W}} \mathbf{g}\mathbf{g}^T w\, dA \quad \text{及} \quad \mathbf{e} = \int_{\mathcal{W}} (I - J)\mathbf{g} w\, dA$$

故 $G\mathbf{d} = \mathbf{e}$

由上式，我们就可以估计出位移向量 d，显然这个估计是建立在窗口内的像素的是平滑的，即其强度是一致的，而实际上这会在窗口内一些变化较大（如角点）的位置，造成其位移估计 d 偏差较大。然而可以通过迭代的方式，不断的应用新的 d 值，而图像块每次迭代都可以采用双线性插值（获得子像素精确度）得到新位置。

（3）下面最为重要的是这个角点的选择问题，这个角点是根据 G 矩阵的两个特征值来选择的，一方面两个特征值不能太小，排除噪声影响，另一方面两个特征值不能差别太大，说明这是角点。这里提出了下面的公式：

$$\min(\lambda_1, \lambda_2) > \lambda$$

这里的阈值是为区分目标前景同背景的，在实际操作中，可以选择图像上大约一致亮度区域的特征值为阈值下限，而将角点或高纹理区域的特征值视为上限，一般情况下我们取上下限

（4）特征点的选择：首先将最小特征值降序排列，优先选择最小特征值大的。为了防止窗口重叠，如果发生选择的特征点在先前选择特征点的窗口内，就删除。

这个现象虽然对于跟踪没有影响，但会影响对于兴趣点（角点）的定位，其主要因为是像素在明亮区域的变化要比在黑暗区域的变化要大。

（5）通过设置实际的能量偏差函数的阈值来排除，被阻挡的点

1.3 总结

KLT 角点检测方法，实际上同 Harris 方法一样，都是计算梯度矩阵的特征值，都认为特征值都较大且接近的点是我们要寻找特征点，然而 KLT 不同的是，一个它计算的是窗口内所以点梯度矩阵的权值和，而 Harris 是一个像素点。二个就是它需要分别计算最小特征值大小，将比较其大小来选择特征点，而 Harris 并不需要具体计算特征值，而通过矩阵迹与行列式的比值来获得特征值比的。

2. 2D Transformations

2.1 Types of 2D Transformations

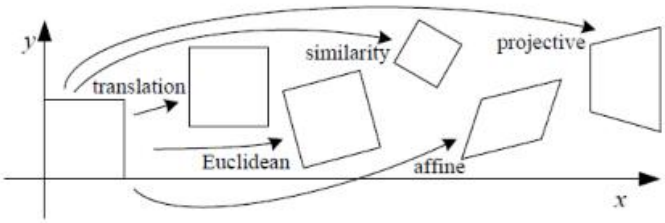


Figure 4: Types of 2D transformations.

There are several types of 2D transformations. Choosing the correct 2D transformations can depend on the camera (e.g. placement, movement, and viewpoint) and objects. A number of 2D transformations are shown in Figure 3.1. Examples of 2D transformations include:

- **Translation Transformation**. (e.g. Fixed overhead cameras)
- **Similarity Transformation**. (e.g. Fixed cameras of a basketball game)
- **Affine Transformation**. (e.g. People in pedestrian detection)
- **Projective Transformation**. (e.g. Moving cameras)

2.2 Translation(平移)

Translational motion is the motion by which a body shifts from one point in space to another. Assume we have a simple point m with coordinates (x, y). Applying a translation motion on m shifts it from (x, y) to (x', y') where

$$\begin{aligned} x' &= x + b_1 \\ y' &= y + b_2 \end{aligned} \tag{1}$$

We can write this as a matrix transformation using homogeneous coordinates:

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} 1 & 0 & b_1 \\ 0 & 1 & b_2 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \tag{2}$$

Let W be the above transformation defined as:

$$W(\boldsymbol{x}; \boldsymbol{p}) = \begin{pmatrix} 1 & 0 & b_1 \\ 0 & 1 & b_2 \end{pmatrix} \tag{3}$$

where the parameter vector is $p = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}$.

Taking the partial derivative of W with respect to $\boldsymbol{p}$ we get:

$$\frac{\partial W}{\partial \boldsymbol{p}}(\boldsymbol{x}; \boldsymbol{p}) = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \tag{4}$$

This is called the Jacobian.

2.3 Similarity Motion

Similarity motion is a rigid motion that includes scaling and translation.

We can define the similarity as:

$$\begin{aligned} x' &= ax + b_1 \\ y' &= ay + b_2 \end{aligned} \tag{5}$$

The similarity transformation matrix W and parameters p are defined as the following:

$$\begin{aligned} W &= \begin{pmatrix} a & 0 & b_1 \\ 0 & a & b_2 \end{pmatrix} \\ \boldsymbol{p} &= \begin{pmatrix} a & b_1 & b_2 \end{pmatrix}^T \end{aligned} \tag{6}$$

The Jacobian of the similarity transformation is then:

$$\frac{\partial W}{\partial \boldsymbol{p}}(\boldsymbol{x}; \boldsymbol{p}) = \begin{pmatrix} x & 1 & 0 \\ y & 0 & 1 \end{pmatrix} \tag{7}$$

2.4 Affine motion

Affine motion includes scaling, rotation, and translation. We can express this as the following:

$$\begin{aligned} x' &= a_1 x + a_2 y + b_1 \\ y' &= a_3 x + a_4 y + b_2 \end{aligned} \tag{8}$$

The affine transformation can be described with the following transformation matrix W and parameters p:

$$\begin{aligned} W &= \begin{pmatrix} a_1 & a_2 & b_1 \\ a_3 & a_4 & b_2 \end{pmatrix} \\ p &= (a_1 \quad a_2 \quad b_1 \quad a_3 \quad a_4 \quad b_2)^T \end{aligned} \tag{9}$$

Finally, the Jacobian for affine motion is the following:

$$\frac{\partial W}{\partial \boldsymbol{p}}(\boldsymbol{x};\boldsymbol{p}) = \begin{pmatrix} x & y & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x & y & 1 \end{pmatrix} \tag{10}$$

3. Iterative KLT tracker

Steps:

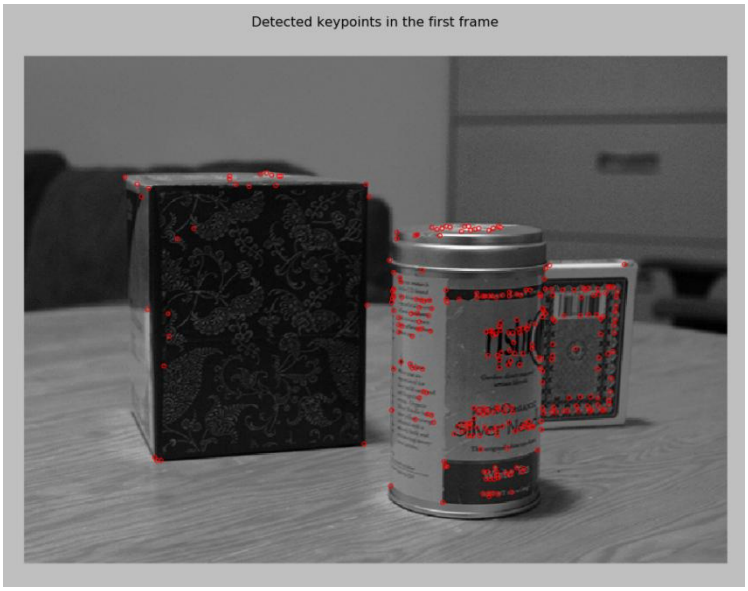
1. First, use Harris corner detection to find the features to be tracked.
2. For each feature at location $x = [x, y]^T$: Choose a feature descriptor and use it to create an initial template for that feature (likely using nearby pixels): $T(x)$.
3. Solve for the transform p that minimizes the error of the feature description around $x_2 = W(x; p)$ (your hypothesis for where the feature's new location is) in the next frame. In other words, solve the equation
$$\sum_x [T(W(x;p)) - T(x)]^2$$
4. Iteratively reapply this to link frames together, storing the coordinates of the features as the transforms are continuously applied. This should give you a measure of how objects move through frames.
5. Just as before, every 10-15 frames introduce new Harris corners to account for occlusion and "lost" features.

4. Hw8

4.1 Lucas-Kanade Method for Optical Flow

4.1.1 Implementation of Lucas-Kanade method

In this section, we are going to implement basic Lucas-Kanade method for feature tracking. In order to do so, we first need to find keypoints to track. Harris corner detector is commonly used to initialize the keypoints to track with Lucas-Kanade method. For this assignment, we are going to use `skimage` implementation of Harris corner detector.



Implement function `lucas_kanade` in `motion.py` and run the code cell below. You will be able to see small arrows pointing towards the directions where keypoints are moving.

We can estimate the position of the keypoints in the next frame by adding the flow vectors to the keypoints.

4.2 Feature Tracking in multiple frames

Now we can use Lucas-Kanade method to track keypoints across multiple frames. The idea is simple: compute flow vectors at keypoints in i-th frame, and add the flow vectors to the points to keep track of the points in i+1-th frame. We have provided the function track_features for you. First, run the code cell below. You will notice that some of the points just drift away and are not tracked very well.

Instead of keeping these 'bad' tracks, we would want to somehow declare some points are 'lost' and just discard them. One simple way to is to compare the patches around tracked points in two subsequent frames. If the patch around a point is NOT similar to the patch around the corresponding point in the next frame, then we declare the point to be lost. Here, we are going to use mean squared error between two normalized patches as the criterion for lost tracks.


```
    kp_curr = keypoints
    trajs = [kp_curr]
    patch_size = 3  # Take 3x3 patches to compute error
    w = patch_size // 2  # patch_size//2 around a pixel

    for i in range(len(frames) - 1):
        I = frames[i]
        J = frames[i + 1]
        flow_vectors = optflow_fn(I, J, kp_curr, **kwargs)
        kp_next = kp_curr + flow_vectors

        new_keypoints = []
        for yi, xi, yj, xj in np.hstack((kp_curr, kp_next)):
            # Declare a keypoint to be 'lost' IF:
            # 1. the keypoint falls outside the image J
            # 2. the error between points in I and J is larger than threshold

            yi = int(round(yi))
            xi = int(round(xi))
            yj = int(round(yj))
            xj = int(round(xj))
            # Point falls outside the image
            if yj > J.shape[0] - exclude_border - 1 or yj < exclude_border or\
               xj > J.shape[1] - exclude_border - 1 or xj < exclude_border:
                continue

            # Compute error between patches in image I and J
            patchI = I[yi - w:yi + w + 1, xi - w:xi + w + 1]
            patchJ = J[yj - w:yj + w + 1, xj - w:xj + w + 1]
            error = compute_error(patchI, patchJ)
            if error > error_thresh:
                continue

            new_keypoints.append([yj, xj])

        kp_curr = np.array(new_keypoints)
        trajs.append(kp_curr)

    return trajs
```

4.3 Pyramidal Lucas-Kanade Feature Tracker

One limitation of the naive Lucas-Kanade method is that it cannot track large motions between frames. You might have noticed that the resulting flow vectors (blue arrows) in the previous section are too small that the tracked keypoints are slightly off from where they should be. In order to address this problem, we can iteratively refine the estimated optical flow vectors. Below is the step-by-step description of the algorithm:

Let $p = \begin{bmatrix} p_x & p_y \end{bmatrix}^T$ be a point on frame I. The goal is to find flow vector $v = \begin{bmatrix} v_x & v_y \end{bmatrix}^T$ such that $p + v$ is the corresponding point of p on the next frame J.

- Initialize flow vector:

$$v = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

- Compute spatial gradient matrix:

$$G = \sum_{x=p_x-w}^{p_x+w} \sum_{y=p_y-w}^{p_y+w} \begin{bmatrix} I_x^2(x,y) & I_x(x,y)I_y(x,y) \\ I_x(x,y)I_y(x,y) & I_y^2(x,y) \end{bmatrix}$$

- **for** $k = 1$ **to** K
  - Compute temporal difference: $\delta I_k(x,y) = I(x,y) - J(x + g_x + v_x, y + g_y + v_y)$
  - Compute image mismatch vector:

$$b_k = \sum_{x=p_x-w}^{p_x+w} \sum_{y=p_y-w}^{p_y+w} \begin{bmatrix} \delta I_k(x,y) I_x(x,y) \\ \delta I_k(x,y) I_y(x,y) \end{bmatrix}$$

  - Compute optical flow: $v^k = G^{-1} b_k$
  - Update flow vector for next iteration: $v := v + v^k$

- Return v

4.4 Coarse-to-Fine Optical Flow

The iterative method still could not track larger motions. If we downscaled the images, larger displacements would become easier to track. On the otherhand, smaller motions would become more difficult to track as we lose details in the images. To address this problem, we can represent images in multi-scale, and compute flow vectors from coarse to fine scale.

Following is the description of pyramidal Lucas-Kanade algorithm:

Let p be a point on image I and s be the scale of pyramid representation.

- Build pyramid representations of I and J: $\{I^L\}_{L=0,\ldots,L_m}$ and $\{J^L\}_{L=0,\ldots,L_m}$
- Initialize pyramidal guess $g^{L_m} = \begin{bmatrix} g_x^{L_m} & g_y^{L_m} \end{bmatrix}^T = \begin{bmatrix} 0 & 0 \end{bmatrix}^T$
- **for $L = L_m$ to 0 with step of -1**
  - Compute location of p on I^L: $p^L = p/s^L$
  - Let d^L be the optical flow vector at level L:
    $$d^L := IterativeLucasKanade(I^L, J^L, p^L, g^L)$$
  - Guess for next level $L-1$: $g^{L-1} = s(g^L + d^L)$
- Return $d = g^0 + d^0$

Lecture 19 :Introduction to Deep Learning

1. Supervised learning(监督学习)

Supervised Learning（监督学习）是指我们来教计算机如何“学习”，其中两个核心问题是：回归 （regression）和分类（classification），都是利用大量的标注数据进行训练，学习得到一个 model 或者说一个函数，从而对未知数据进行预测

Supervised learning is the machine learning task of inferring a function from labeled training data.[1] The training data consist of a set of training examples. In supervised learning, each example is a pair consisting of an input object (typically a vector) and a desired output value (also called the supervisory signal). A supervised learning algorithm analyzes the training data and produces an inferred function, which can be used for mapping new examples. An optimal scenario will allow for the algorithm to correctly determine the class labels for unseen instances. This requires the learning algorithm to generalize from the training data to unseen situations in a “reasonable” way.

总结来说，监督学习就是对具有标签（label）的训练样本（train data）进行学习，找到 data 和 label 之间的映射关系（mapping，更确切的说是一个 function），从而利用该映射关系对无标签的样本进行预测（predict），得到其标签。

在监督学习领域，两大研究分支是：

· Regression（回归）

· Classification（分类）

分类和回归，本质上都是对样本数据的一种预测，区别在于输出变量的类型。

定量输出称为回归，或者说是连续变量预测；

定性输出称为分类，或者说是离散变量预测。

从数学的角度来说，

分类问题和回归问题都要根据训练样本找到一个实值函数 g(x)。

回归问题是：

给定一个新的样本 x，根据训练集推断它所对应的输出 y(实数)是多少，也就是使用 y=g(x)来推断任一输入 x 所对应的输出值。

分类问题是：

给定一个新的样本 x，根据训练集推断它所对应的类别（如：+1，-1），也就是使用 y=sign(g(x))来推断任一输入 x 所对应的类别。

综上，回归问题和分类问题的本质一样，不同仅在于他们的输出的取值范围不同。

分类问题中，输出只允许取两个值；而在回归问题中，输出可取任意实数。分类一般针对离散型结果而言的，回归是针对连续型结果的，本质上是一样的。

网址：http://blog.csdn.net/walilk/article/details/50922854

1.1 线性回归

一元线性回归的标准描述

综合上述所提到的假设函数、代价函数和优化目标，我们可以发现一元线性回归问题涉及到4个部分：

假设函数、待优化参数、代价函数、优化目标

其标准化描述如下：

Hypothesis:

$$h_\theta(x) = \theta_0 + \theta_1 x$$

Parameters:

$$\theta_0,\ \theta_1$$

Cost Function:

$$J(\theta_0, \theta_1) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$$

Goal:

$$\underset{\theta_0,\theta_1}{\text{minimize}}\ J(\theta_0, \theta_1)$$

至此，我们引出了线性回归问题中的几个核心概念。更进一步说，“假设函数、待优化参数、代价函数、优化目标”也就是监督学习问题中的最核心的几个部分，更更进一步说，这也就是机器学习问题中的最核心的几个部分。

多元线性回归的标准描述

Hypothesis:

$$h_\theta(x) = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$$

where $x_0 = 1$

Parameters:

$$\theta_0,\ \theta_1,\ \theta_2, \ldots, \theta_n$$

Cost Function:

$$J(\theta_0, \theta_1, \theta_2, \ldots, \theta_n) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$$

Goal:

$$\underset{\theta_0,\theta_1,\ldots,\theta_n}{\text{minimize}}\ J(\theta_0, \theta_1, \theta_2, \ldots, \theta_n)$$

1.2 线性分类（Linear Classification）

分类算法的本质是将一个输入数据映射为一个标签，同样分为两个阶段：

训练，即利用已有的带标签数据，找到数据和标签之间的映射关系（函数）；

预测，即利用学习出的函数/模型，对新的数据进行预测，得到其标签。

在线性分类问题中，分类函数是一个线性函数。在一维空间里该函数就是一个点，在二维空间里就是一条直线，三维空间里就是一个平面，可以如此想象下去，如果不关注空间的维数，这种线性函数还有一个统一的名称——超平面（Hyper Plane）！

实际上，线性函数是一个实值函数（即 函数的值是连续的实数），而分类问题需要离散的输出值，例如用 1 表示某个样本属于类别 C1，而用 0 表示不属于 C1，也就意味着属于 C2，这时候只需要在实值函数的基础上附加一个阈值即可，即通过判断分类函数的输出是否大于这个阈值来确定类别归属。

例如有一个线性函数 g(x)=wx+b，我们可以取阈值为 0，这样当有一个样本 xi 需要判别的时候，就可以通过 g(xi) 的值与 0 比较，确定样本的类别归属。即，若 g(xi)>= 0，则判别为类别 C1，若 g(xi)<0，则判别为类别 C2。此时也等价于给函数 g(x)附加一个符号函数 sgn()，即 f(x)=sgn [g(x)]是我们真正的判别函数。

在已知大量有标签数据的情况下，如果找到这样一个线性函数，就是线性分类器所研究的问题。回想一下在线性回归中的思路，其实这里也是一样的，我们需要针对数据和标签之间的对应关系提出一种假设，并设计一个代价函数用于评判假设函数的优劣，然后利用某种优化方法来求解最优的参数。

假设函数（Hypothesis Function）

$$f(x_i, W, b) = Wx_i + b$$

代价函数（Cost Function）

cost of single sample:

$$L_i = \sum_{j \neq y_i} \max(0, w_j^T x_i - w_{y_i}^T x_i + \Delta)$$

cost of train dataset:

$$L = \frac{1}{N} \sum_i \sum_{j \neq y_i} [\max(0, f(x_i; W)_j - f(x_i; W)_{y_i} + 1)] + \alpha R(W)$$

优化目标（Optimization Objective）

$$\underset{W}{\text{minimize}} \ L(W)$$

2. Gradient descent

2.1 梯度

在微积分里面，对多元函数的参数求∂偏导数，把求得的各个参数的偏导数以向量的形式写出来，就是梯度。比如函数 f(x,y)，分别对 x,y 求偏导数，求得的梯度向量就是(∂f/∂x, ∂f/∂y)T,简称 grad f(x,y)或者▽f(x,y)。对于在点(x0,y0)的具体梯度向量就是(∂f/∂x0, ∂f/∂y0)T.或者▽f(x0,y0)，如果是 3 个参数的向量梯度，就是(∂f/∂x, ∂f/∂y，∂f/∂z)T,以此类推。

那么这个梯度向量求出来有什么意义呢？他的意义从几何意义上讲，就是函数变化增加最快的地方。具体来说，对于函数 f(x,y),在点(x0,y0)，沿着梯度向量的方向就是(∂f/∂x0, ∂f/∂y0)T 的方向是 f(x,y)增加最快的地方。或者说，沿着梯度向量的方向，更加容易找到函数的最大值。反过来说，沿着梯度向量相反的方向，也就是 -(∂f/∂x0, ∂f/∂y0)T 的方向，梯度减少最快，也就是更加容易找到函数的最小值。

2.2 梯度下降与梯度上升

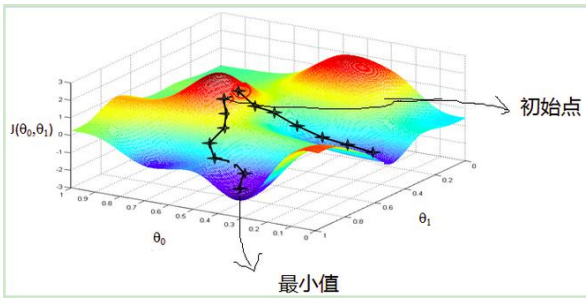
在机器学习算法中，在最小化损失函数时，可以通过梯度下降法来一步步的迭代求解，得到最小化的损失函数，和模型参数值。反过来，如果我们需要求解损失函数的最大值，这时就需要用梯度上升法来迭代了。

梯度下降法和梯度上升法是可以互相转化的。比如我们需要求解损失函数 f(θ)的最小值，这时我们需要用梯度下降法来迭代求解。但是实际上，我们可以反过来求解损失函数 -f(θ)的最大值，这时梯度上升法就派上用场了。

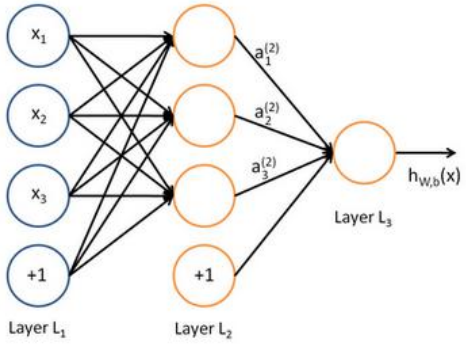
2.3 梯度下降的直观解释

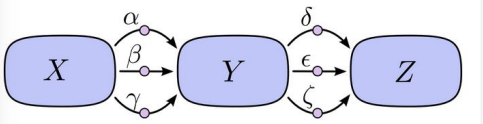
。比如我们在一座大山上的某处位置，由于我们不知道怎么下山，于是决定走一步算一步，也就是在每走到一个位置的时候，求解当前位置的梯度，沿着梯度的负方向，也就是当前最陡峭的位置向下走一步，然后继续求解当前位置梯度，向这一步所在位置沿着最陡峭最易下山的位置走一步。这样一步步的走下去，一直走到觉得我们已经到了山脚。当然这样走下去，有可能我们不能走到山脚，而是到了某一个局部的山峰低处。

从上面的解释可以看出，梯度下降不一定能够找到全局的最优解，有可能是一个局部最优解。当然，如果损失函数是凸函数，梯度下降法得到的解就一定是全局最优解。



3．Backpropagation(反向传播算法)





图中的边表示偏导数，如$\alpha = \frac{\partial Y}{\partial X}$

想要知道输入X对输出Z的影响，我们可以用偏导数$\frac{\partial Z}{\partial X} = \frac{\partial Z}{\partial Y} \cdot \frac{\partial Y}{\partial X}$，即：

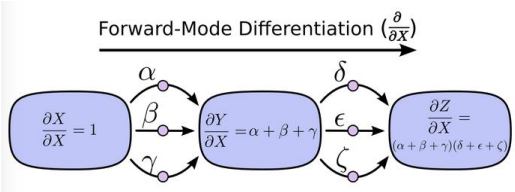
$$\frac{\partial Z}{\partial X} = \alpha\delta + \alpha\epsilon + \alpha\zeta + \beta\delta + \beta\epsilon + \beta\zeta + \gamma\delta + \gamma\epsilon + \gamma\zeta \qquad \text{...式子1}$$

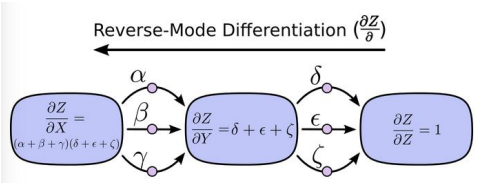
如果直接使用链式法则进行求导会出现一个问题，当路径数目增加时，该式子中的子项数目会呈指数增长，考虑到这种情况，我们把上式右侧进行合并，得到：

$$\frac{\partial Z}{\partial X} = (\alpha + \beta + \gamma)(\delta + \epsilon + \zeta) \qquad \text{...式子2}$$

合并后的式子是不是非常清晰，关键是与式子1相比，只需要进行一次乘法就可以得出结果，大大提高了运算速度。

现在的问题是，对于式子2，应该如何实现？我们知道，假设一条北京 → 郑州 → 武汉的路径，我们可以先分析北京至郑州，再分析郑州至武汉；也可以先分析郑州至武汉，再分析北京至郑州。同理，根据计算方向的不同，可以分为正向微分与反向微分。我们先看针对上图的正向微分算法：





可以看到，正向微分算法根据路径的传播方向，依次计算路径中的各结点对输入 X 的偏导数，结果中保留了输入对各结点的影响。 可以看到，该算法从后向前进行计算，结果中保留了路径中各结点对输出的影响。

Tips: numpy.random.randn()与 rand()的区别

numpy 中有一些常用的用来产生随机数的函数，randn()和 rand()就属于这其中。

numpy.random.randn(d0, d1, …, dn)是从标准正态分布中返回一个或多个样本值。

numpy.random.rand(d0, d1, …, dn)的随机样本位于[0, 1)中。

CS131 Computer Vision: Foundations and Applications

Practice Final (Solution)

1. Multiple choice answers

1.1 RANSAC vs Least squares

两者的共同点都是要首先确定模型，模型分为线性模型与非线性模型。

一般常见的应用是线性模型，如 f(x) = kx + t。

在应用上，二者的差别是，least squares 对噪声比较敏感，算法简单。而 RANSAC 能去除一些噪声的干扰，如果假定模型与实际的情形一致，那么一般由观测数据计算的 RANSAC 模型，更能接近实际情况，去除观测或过程噪声干扰，算法稍微复杂些

最小二乘 least squares 通过最小化 测量数据与模型数据的平方和， 求出模型参数。

e = sum(Yi - f(Xi)), e 是 k 与 t 的二次函数，求其最小，即求其极值。求偏导 dk = 0， dt = 0。可以计算出 k 与 t 值。

RANSAC 的线性拟合算法步骤大致如下：

while 最大尝试次数

从观测点集中随机取两点，计算出直线的参数 k, t（或者 k 用向量表示），得出一个候选的直线模型。

计算候选直线与整个点集的匹配程度，可以采用统计在直线上(或到直线的距离小于一个阈值）的点的个数。

保留匹配程度最好的直线的参数。

如果本次尝试匹配点的个数 占整个点集大部分，超出预期（阈值），提前结束尝试。

endwhile

匹配程度也可以用其他指标来衡量，替换上面的匹配计算。

误差/性能分析：

最小二乘的误差为 e，但一般用相关系数 r 来表示，r 越接近 1，模型越好，r = 0，拟合无意义，模型不实际情况不符合。RANSAC 的误差一般用在拟合直线的一个范围内，在此范围内点的个数占整个点集比例 来衡量。

1.2 SIFT

旋转后以主方向为中心取 8×8的窗口。下图所示，左图的中央为当前关键点的位置，每个[illegible]代表为关键点邻域所在尺度空间的一个像素，求取每个像素的梯度幅值与梯度方向，箭头方向代表该[illegible]梯度方向，长度代表梯度幅值，然后利用高斯窗口对其进行加权运算。最后在每个4×4的小块上绘[illegible]个方向的梯度直方图，计算每个梯度方向的累加值，即可形成一个种子点，如右图所示。每个特征点[illegible]4个种子点组成，每个种子点有8个方向的向量信息。这种邻域方向性信息联合增强了算法的抗噪声能力，同时对于含有定位误差的特征匹配也提供了比较理性的容错性。

Fork me on GitHub

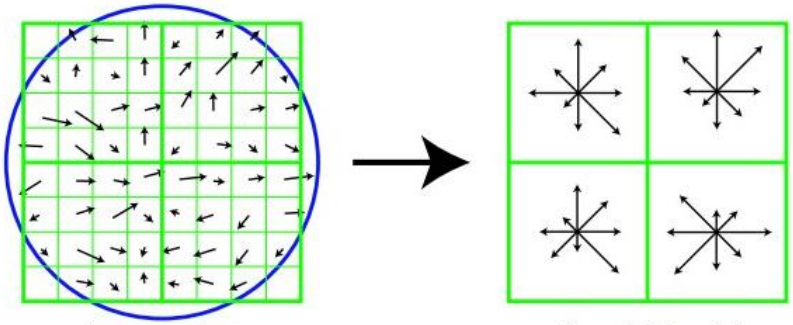


Image gradients　　Keypoint descriptor

与求主方向不同，此时每个种子区域的梯度直方图在0-360之间划分为8个方向区间，每个区间为45度，即每个种子点有8个方向的梯度强度信息。

1.3 Hessian

多元函数的黑塞矩阵

将二元函数的泰勒展开式推广到多元函数，则 $f(x_1, x_2, \cdots, x_n)$ 在 点处的泰勒展开式的矩阵形式为：

其中：

（1） $\nabla f\left(X^{(0)}\right) = \left[\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \cdots, \frac{\partial f}{\partial x_n}\right]\Big|_{X^{(0)}}^{T}$ ，它是 $f(X)$ 在 点处的梯度。

（2） $G(X^{(0)}) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}_{X^{(0)}}$ 为函数 $f(X)$ 在 点处的黑塞矩阵。[1]

黑塞矩阵是由目标函数 f 在点X处的二阶偏导数组成的 $n \times n$ 阶对称矩阵。[2]

试卷作答：

Multiple choice answers (11points)

1. **RANSAC vs Least squares (1 point).** What are the benefits of RANSAC compared to the least squares method? *Circle all that apply.* CD
   - A. RANSAC is faster to compute in all cases
   - B. RANSAC has a closed form solution
   - C. RANSAC is more robust to outliers √
   - D. RANSAC handles measurements with small Gaussian noise better √

2. **SIFT (1 point).** What does each entry in a 128-dimensional SIFT feature vector represent? (Choose the best answer) C 4×4×8=128
   - A. A principal component.
   - B. A sum of pixel values over a small image patch.
   - C. One bin in a histogram of gradient directions. √
   - D. The number of pixels whose gradient magnitude falls within some range.

3. **Clustering (1 point).** Which of the following statement is true for k-means and HAC clustering algorithm? (Choose the best answer) D
   - A. k-means and HAC are both sensitive to cluster center initialization. ×
   - B. k-means and HAC can lead to different clusters when applied to the same data.
   - C. k-means works well for non-spherical clusters.
   - D. HAC starts with all examples in the same cluster, then each cluster is split until we have the desired number of predetermined clusters. √

4. **Video Tracking (1 point).** What kinds of image regions are hard to track? *(Circle all that apply)*: ABD
   - A. low texture regions like sky √
   - B. Edges of objects √
   - C. Corners of objects
   - D. Rotating spheres. √

5. **Optical flow (1 point).** What are the methods for estimating optical flow that we covered in class? *Circle all that apply.* AC
   - A. Seam carving. √
   - B. Gibbs random field motion estimation. 吉布斯随机向的光场预计
   - C. Optical flow equation and Second order derivatives of optical flow field. √
   - D. Hidden Markov Model. 隐马尔可夫模型.

6. **Hessian (1 point).** Given a vector in $\mathcal{R}^n$ (implying that the vector is n-dimensional), its Hessian is in: C.
   - A. undefined
   - B. $\mathcal{R}^n$
   - C. $\mathcal{R}^{n\times n}$ √
   - D. $\mathcal{R}^{n^2}$

7. **Similarity transformation (2 points).** Which of the following always hold(s) under an similarity transformation? *Circle all that apply.* AB
   - A. Parallel lines will remain parallel. √

B. The ratio between the two areas or two polygons will remain the same.

C. Perpendicular lines will remain perpendicular.

D. The angle between two line segments will remain the same. ✗

8. **Scale invariance (2 points).** Which of the following representations of an image region are scale invariant? *Circle all that apply.*

BCD

A. Bag of words model with SIFT features

B. Spatial Pyramid Model with SIFT features

C. A HOG template model ✓

D. Deformable Parts Model ✓

9. **Optical flow (1 point).** Optical flow is problematic in which conditions? *Circle all that apply.*

AC

A. In homogeneous image areas.

B. In textured image areas. ✗

C. At image edges. ✓

D. At the boundaries of moving objects.

True and False (11points)

1. T (1 point.) A smoothing filter sums to 1.
2. F (1 point.) A scaling matrix by a, b, c in the x, y, and z directions, respectively, has the transformation matrix: $T = \begin{bmatrix} a & 0 & 0 & 0 \\ 0 & b & 0 & 0 \\ 0 & 0 & c & 0 \\ 0 & 0 & 0 & \frac{1}{abc} \end{bmatrix}$.
3. F (1 point.) In figure 1, we applied a Gaussian filter to the original image and computed the derivative in x and y directions. **True or False:** Result 1 is the x-derivative of Gaussian, and result 2 is the y-derivative of Gaussian.

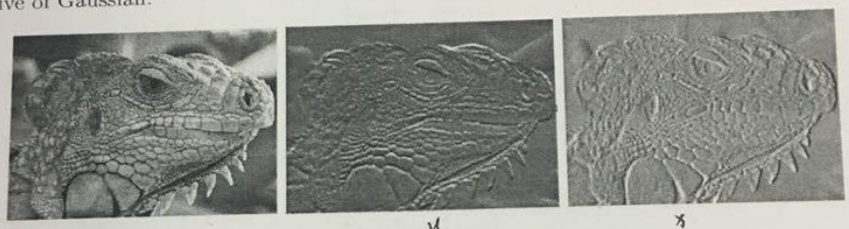


Figure 1: **Left**: Original image; **Center**: Result 1; **Right**: Result 2

4. F (1 point.) If you compute optical flow in a video sequence, a nonzero optical flow vector always indicates real movement of an object in the scene depicted by the video.
5. T (1 point.) A set of eigenfaces can be generated by performing principal component analysis (PCA) on a large set of images depicting different human faces. Any human face from this large set can be considered to be a combination of these standard faces.
6. T (1 point.) Pyramidal Lucas-Kanade method can track large motions between frames while Lucas-Kanade without the pyramids can not.
7. F (1 point.) When using Harris detector, an image region is considered to be a corner if both λ_1 and λ_2 are small. (λ_1 and λ_2 are the eigenvalues of the second moment matrix.)
8. T (1 point.) Even without using non-maximum suppression, we know that all pixels with gradient magnitudes above the 'high' threshold will be considered an edgeb by a Canny edge detector.
9. F (1 point.) Optical illusions are NOT a type of optical flow.
10. F (1 point.) In texture free regions, there is no optical flow.
11. T (1 point.) Motion could be used to improve video resolution quality.

Short Answers 1: Filters, edges, corners, keypoints and descriptors (12points.)

1. **Filters (4 points).** Give ANY 3x3 example of each of the following types of image convolution filters:

   a. **(1 point.)** A dimming filter (only decrease the image intensity)

   调光滤波矩阵

0.1	0.1	0.1
0.1	0.1	0.1
0.1	0.1	0.1

   和<1

   b. **(1 point.)** Approximation for difference of Gaussian filter DoG

   逼近

   (3,0.8)
   (3,0.1)

0.05	0.11	0.05
0.11	-15.69	0.11
0.05	0.11	0.05

$$DoG = G(x,y,k\sigma) - G(x,y,\sigma)$$

$$G(x,y\sigma) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

   c. **(1 point.)** A filter for detecting diagonal edge from top-right to bottom-left

0	0	-1
0	2	0
-1	0	0

   d. **(1 point.)** A filter that shifts pixels to the right by one pixel 右移一像素

0	0	0
-1	1	0
0	0	0

   →

2. **Median filter (4 points).** A "Median Filter" operates over a window by selecting the median intensity in the window.

1. **(1 point).** What advantage does a median filter have over a mean filter?

方法：用个窗滑过信号，用邻域或中值去取代窗口中心位置的值。

特点：① 在平滑脉冲噪声方面非常有效，可保护图像尖锐的边缘

② 中值比均值更稳健，邻域中一个非常小的有代表性的像素不会影响到中值。

③ 当滤波器跨过边缘时，中值不会创建不存在的像素值，可保留锐利边缘。

2. **(3 points).** Prove that the median filter can be written as a convolution operation or explain with an example why it can not.

非线性滤波器

领域元素，类似于卷积，但不是加权求和

把领域中像素按级别排序，选择该组中间值作为输出像素值

$$g(x,y) = \text{median}\{f(x-i, y-j)\}, (i,j) \in W$$

叠加定理适用于互相关

3. **Cross-correlation (4 points)**. Prove that the superposition principle holds for cross-correlation or provide a counterexample.

Recall that the cross-correlation between f and a template h is function $S[f] = f \star h$ defined by:

$$(S[f])[m,n] = (f \star h)[m,n] \stackrel{\text{def}}{=} \sum_k \sum_l f[m+k, n+l] h[k,l]$$

.

The superposition principle for system S is:

$$S[\sum_i \alpha_i f_i] = \sum_i \alpha_i S[f_i]$$

. where S is the the cross correlation system operator associated with filter h:

$$S[f] = f \star h$$

The α_i are all constants in $\mathbb{R}$, and the f_i are images defined on $\mathbb{R}^2$.

$$S[\sum_i \alpha_i f_i][m,n] = (\sum_i \alpha_i f_i \star h)[m,n] \stackrel{\text{def}}{=} \sum_k \sum_l \sum_i \alpha_i f_i[m+k, n+l] h[k,l]$$

$$= \sum_i \alpha_i \left\{ \sum_k \sum_l f_i[m+k, n+l] h[k,l] \right\}$$

$$= \sum_i \alpha_i S[f_i]$$

故满足叠加定理.

Short Answers 2: Segmentation and seam carving (15points.)

1. **k-means (4 points.)** A distance function d on $\mathbb{R}^n$ is said to be *invariant* under a transformation $\phi : \mathbb{R}^n \to \mathbb{R}^n$ if

$$\forall x, y \in \mathbb{R}^n, \quad d(x, y) = d(\phi(x), \phi(y))$$

A cluster center function μ is *invariant* to a transformation $\phi : \mathbb{R}^n \to \mathbb{R}^n$ if

$$\phi(\mu(x^{(1)}, \ldots, x^{(m)})) = \mu(\phi(x^{(1)}), \ldots, \phi(x^{(m)}))$$

for all sets of points $x^{(1)}, \ldots, x^{(m)} \in \mathbb{R}^n$.

A clustering algorithm is said to be invariant to a transformation ϕ if its choice of clusters is not affected when all points x are transformed to $\phi(x)$

Prove or reject with a counter-example: k-means algorithm is invariant to a transformation ϕ if and only if the distance and cluster center function are invariant to ϕ.

$$SSD = \sum_{i \in \text{clusters}} \sum_{x \in \text{cluster } i} (x - c_i)^2$$

cluster center

$$c^*, \delta^* = \operatorname*{argmin}_{c,\delta} \frac{1}{N} \sum_{j}^{N} \sum_{i}^{k} \delta_{ij} (c_i - x_j)^2$$

→ case

↓ Whether x_j is assigned to c_i

$$\because \quad d(x, y) = d[\phi(x), \phi(y)]$$

$$\phi(\mu(x^{(1)}, \ldots x^{(m)})) = \mu[\phi(x^{(1)}), \ldots \phi(x^{(m)})]$$

$$\therefore \quad \phi(c^*, \delta^*) = \operatorname*{arg\,min}_{c,\delta} \frac{1}{N} \sum_{j}^{N} \sum_{i}^{k} \delta_{ij} (\phi(c_i) - \phi(x_j))^2$$

$$= c^*, \delta^*.$$

2. **Seam Carving (11 point).** You have implemented seam carving for image resizing in one direction (vertical or horizontal). Let I be an $n \times m$ image and define a vertical seam to be:

$$\mathbf{s}^y = \{(i, y(i)\}_{i=1}^n, s.t. \forall i, |y(i) - y(i-1)| \leq 1,$$

where y is a mapping $y : [1, ..., n] \rightarrow [1, ..., m]$. Let $e(i, j)$ define the energy of the pixel at location (i, j) of image I. Now, we can define the cost of a seam as $Cost(\mathbf{s}^y) = \sum_{i=1}^n e(i, y(i))$. We look for the optimal (vertical) seam $\mathbf{s}^{y*}$ that minimizes this seam cost:

$$\mathbf{s}^{y*} = \min_{\mathbf{s}^y} Cost(\mathbf{s}^y)$$

(a) **(2 points).** What is the time complexity for brute-force searching strategy to find the optimal seam? (hint: think about how many possible vertical seams are there in $n \times m$ image.)

$S^* = \arg\min_S E(S)$, where $E(I) = \left|\frac{\partial}{\partial x} I\right| + \left|\frac{\partial}{\partial y} I\right|$.

$M(i,j) = E(i,j) + \min(M(i-1, j-1), M(i-1, j), M(i-1, j+1))$.

一个像素明显考虑因素是它和周围像素颜色的对比度来定义的.

$$S(I_k) = \sum_{\forall i, i \neq k} D(I_k, I_i) \quad (1)$$

$D(I_k, I_i)$ 为像素 I_k 和像素 I_i 在 L*a*b 空间的欧氏距离.

(b) **(4 points).** The optimal seam can be found efficiently using dynamic programming. The algorithm traverses the image from the first row to the last row, and at each row i, it computes the cumulative minimum cost $M(i, j)$ for $\forall j$. Recall that $M(i, j)$ is the minimum cost of a seam from the top row to the pixel at (i, j). Define the recurrence relationship that seam carving uses to calculate $M(i, j)$. In other words, define $M(i, j)$ using the functions $M(\cdot)$ and $e(\cdot)$.

原式: $S(I_k) = D(I_k, I_1) + D(I_k, I_2) + \cdots D(I_k, I_N)$ (2)

N为图像 I 中图像素个数.

又考虑相邻颜色相近的像素具有相同的重要性.

$$S(I_k) = S(i,j) = \sum_{j=1}^{n} f_j D(c_j, c_i). \quad (3)$$

时间复杂度: (1) $\rightarrow O(N^2)$

(3) $\rightarrow O(N) + O(n^2)$

Short Answers 3: Classification and detection (10points.)

1. **Eigenpenguins (4 points).**

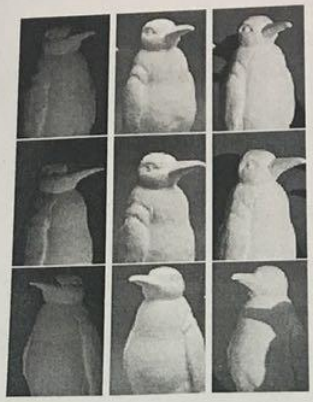


Figure 2: A database of penguins

You have collected a small dataset of penguin statues from different angles and under different lighting conditions, shown in Figure 2. In this figure note that each row contains images of penguin statues taken from the same angle, and each column contains penguin statues under the same lighting conditions.

You would like to use this dataset to recognize the angle from which a photo of a penguin statue has been taken, and you decide to use the Eigenfaces (Eigenpenguins?) algorithm. More concretely, you want to determine the angle from which the image in the upper left corner was taken, using the other images as a training dataset. You use the other images to choose a low-dimensional Eigenpenguin space, project all of the images into this space, and declare that the angle of the query image is the angle of its nearest neighbor in Eigenpenguin space.

Do you expect this algorithm to work correctly for this dataset? Justify your answer or explain why it isn't a good algorithm choice.

PCA + KNN分类算法

降维问题的优化目标：将一组N维向量降为K维，(K远小于N)，其目标是选择K个单位正交基，使得原始数据变换到这组基后，各字段两两间协方差为0，而字段的方差则尽可能大。将方差最大的方向作为主要特征

步骤：1) 将原始数据按列组成n行m列矩阵X

2) 将X的每一行进行零均值化，即减去这一行的均值，求出协方差矩阵 $C=\frac{1}{m}XX^T$

3) 求出协方差矩阵的特征值及对应的特征向量

4) 将特征向量按对应特征值大小排列为矩阵，取前K行 → 新矩阵P

5) 降维表示，即为对图像中心化，并投影到矩阵A，得到测试图像的降维表示。

6) 用K-NN分类算法分类器，对测试图像进行分类

2. **Image compression (6 points).**

You have a big database of images of dogs, with 100,000 images, each of size (200, 200, 3). The raw pixel values are stored on your database (you don't have jpeg), which means that you currently store $100,000 \times 200 \times 200 \times 3$ floats in memory. You want to to reduce the cost of storing all the data.

(a) **(2 points).** Describe the algorithm you will use to compress / decompress images.

RGB → Grayscale. PCA降维

$u = \frac{1}{N}\sum x_i$

$X_c = X - u_1^T = X - \frac{1}{N}X_{11}^T = X(I - \frac{1}{N}X_{11}^T)$

$\Sigma = \frac{1}{N}X_c X_c^T$ $\Sigma = E\{(A - E(A))^T(A - E(A))\}$.

(b) **(2 points).** How will you decide how much you can compress? Let's assume that you you want to maintain $x\%$ variance of your data.

根据奇异阵大小排序. 取前k个. > 90%

SVD分解 → PCA $M \times N \to M \times k$ $(k < N)$

(c) **(1 point).** What will the final size of the compressed database be?

$100000 \times 200 \times 2$.

(d) **(1 point).** What kind of additional information will you have to store to allow you to compress and decompress images? What is the size of that additional information?

奇异矩阵, 奇异阵.

$M \times N = (M \times r) * (r \times r) * (r \times N)$

左 奇 右

Σ V S V

3. **kNN Boundaries (4 points).** Figure 3 shows a training set of 14 examples, with 5 example for class 1 (with the crosses) and 3 examples for class 2 (with the circles).

If we apply k-nearest neighbors, we can classify any testing point into class 1 or class 2. If we do that for every point in $\mathbb{R}^2$, we obtain decision boundaries which delimit zones of $\mathbb{R}^2$ where the classification is either 1 or 2.

(a) **((2 points).** Draw on the image (figure 3) the decision boundaries for $k = 1$. Use **L2 distance** to get nearest neighbors.

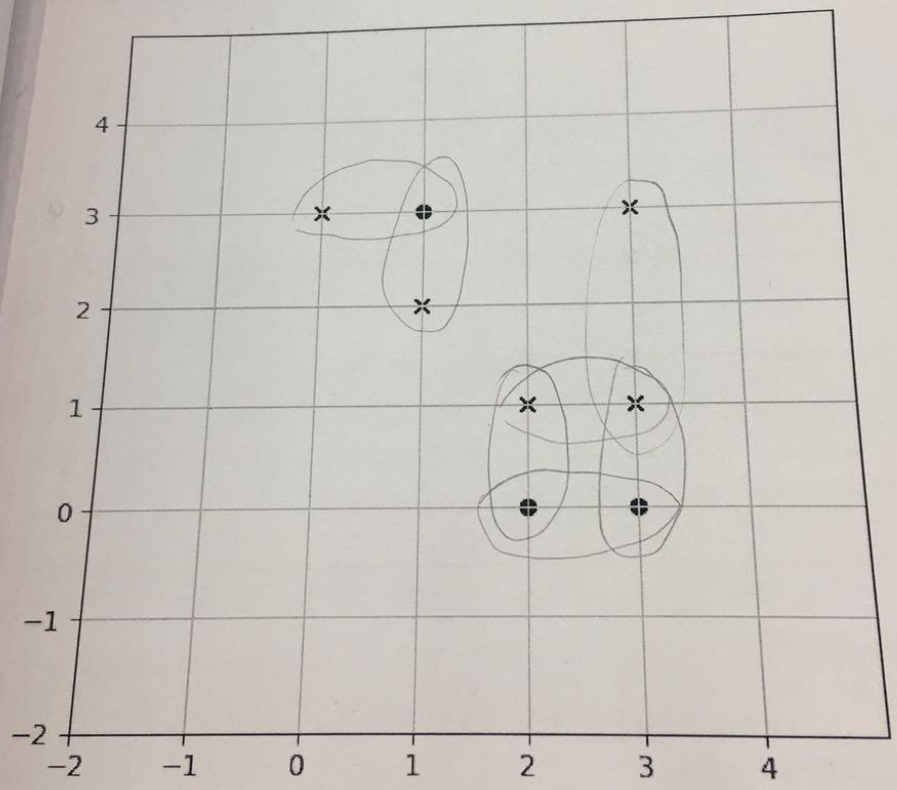


Figure 3: Decision boundaries for k-NN with $k = 1$ and L_2 distance

(b) **((2 points).** Draw on the image (figure 4) the decision boundaries for $k = 3$. Use **L2 distance** to get nearest neighbors.

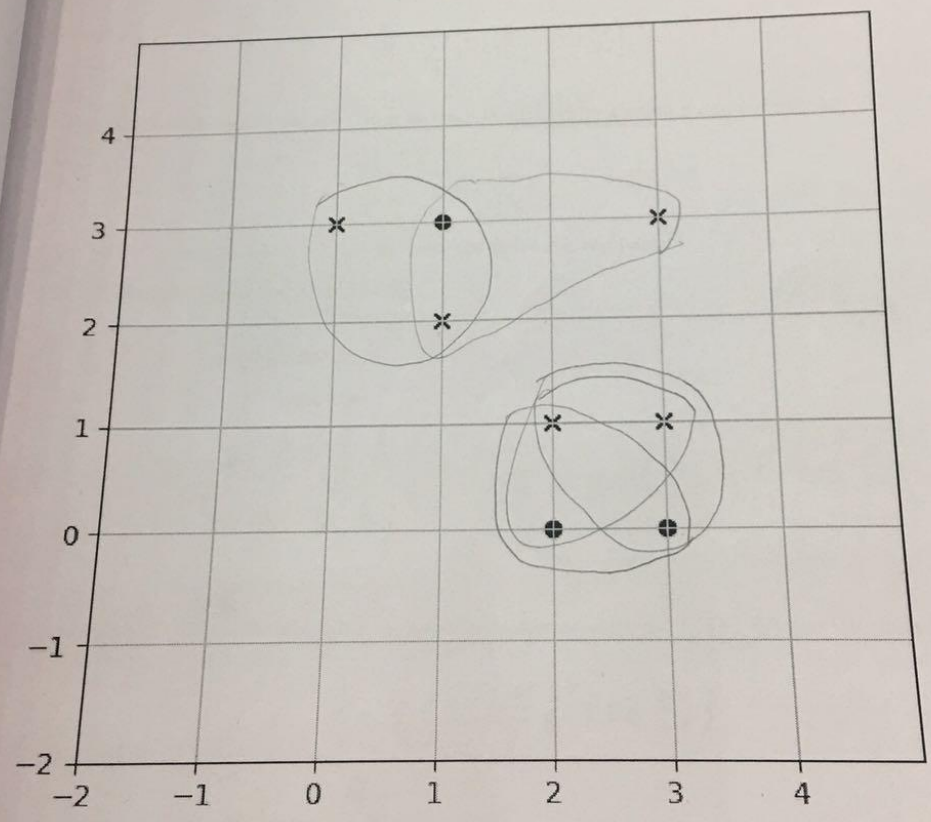


Figure 4: Decision boundaries for k-NN with $k = 3$ and L_2 distance

Short Answers 4: Optical flow and tracking (7points).

1. **KLT tracker (7 questions).** To update the change in parameters in KLT tracker, we derived the following equation:

$$\Delta p = H^{-1} \sum_{\mathbf{x}} \left[\nabla I \frac{\partial W}{\partial p}\right]^T [T(x) - I(W(\mathbf{x}, \mathbf{p_0}))]$$

where

$$H = \sum_{\mathbf{x}} \left[\nabla I \frac{\partial W}{\partial p}\right]^T \left[\nabla I \frac{\partial W}{\partial p}\right]$$

Given that the 2D motion we are trying to track is a similarity motion parameterized by

$$p = \begin{bmatrix} a \\ b_1 \\ b_2 \end{bmatrix},$$

such that $x' = ax + b_1$ and $y' = ay + b_2$, solve the following two parts:

(a) **(3 points).** Derive the Jacobian $\frac{\partial W}{\partial p}$.

$x' = ax + b_1$

$y' = ay + b_2$

$W = \begin{pmatrix} a & 0 & b_1 \\ 0 & a & b_2 \end{pmatrix}$

$p = (a, b_1, b_2)^T$

$\frac{\partial W}{\partial p}(x;p) = \begin{pmatrix} x & 1 & 0 \\ y & 0 & 1 \end{pmatrix}$.

(b) **(4 points).** Derive the H matrix in terms of image derivatives (I_x and I_y) and the pixel locations (x and y).

$$H = \sum_{x} \left[\nabla I \frac{\partial W}{\partial p}\right]^T \left[\nabla I \frac{\partial W}{\partial p}\right]$$

$\frac{\partial W}{\partial p}(x;p) = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$

$H = \begin{pmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{pmatrix}$.