

Peking University  
Introduction to Computer Vision, Spring 2024

---

# Introduction to Computer Vision

## Course Notes

---

Prof. He Wang<sup>\*1</sup>, 林晓疏<sup>†2</sup>, and Yutong Liang<sup>‡2</sup>

<sup>1</sup> 主讲教师

<sup>2</sup> 笔记整理

2024 年 4 月 25 日

---

<sup>\*</sup><https://hughw19.github.io/>

<sup>†</sup>wangyuanqing@pku.edu.cn

<sup>‡</sup>lyt0112@outlook.com

---

## 前言

这本笔记是作者于 2022 年春信息科学技术学院王鹤老师开设的计算机视觉导论课程期间的笔记。王鹤老师在 Stanford 获得 Ph.D 学位，课程中也毫不令人意外地带有许多 CS231n: Convolutional Neural Network for Visual Recognition 和 CS231A: Computer Vision, From 3D Reconstruction to Recognition 等课程的影子。课程从对计算机视觉领域的传统方法的介绍开始，介绍了 CNN 和诸多深度学习的基本知识，如 BatchNorm, Regularization 等。随后进入 3D 视觉部分，详细介绍了 Pinhole Camera 这一模型以及相机标定，对极几何等相关知识。期中之后转入 3D 数据，语义分割，物体位姿判定以及 RNN 和生成模型部分。

笔记主要是对王鹤老师上课内容的记录，部分内容由笔者在课余时间了解后添加，这些内容都给出了参考文献或链接。除此之外，笔者还依惯例添加了几节附录，以补充正文当中一些没有展开的细节，以供参考。

这门课是笔者三年以来在信科上过的水准最高的课程，无论是课程内容，教师讲授水平，作业质量，考试区分度还是答疑，都是笔者体验过的课程中最高水准的一档。若信科未来能有一半专业课能达到本课的水平，则世界一流大学指日可待（）。

最后，感谢王鹤老师和张嘉墨，陈嘉毅两位助教。笔者曾多次向张助教询问问题，均得到了细致的回答，在此一并表示感谢。

林晓疏  
2022 年春

作为北京大学信息科学技术学院的学生，长期以来饱受糟糕课程质量、糟糕课程作业、糟糕考试难度的折磨。比如算法设计与分析的等课程的教学质量极低，教考分离，ICS 考试一面黑板的考试错误题目订正等等。在这样的环境下，我很幸运地遇到了王鹤老师开设的计算机视觉导论课程，这门课程的内容丰富、难度适中，作业质量高，考试难度适中，绝对称得上是精品课程（与算分这种国家精品课程相区别）。

王鹤老师将计算机视觉的发展脉络呈现给大家，在这个深度学习时代，老师并没有完全忽视传统 CV 的方法，而是挑选了其中具有代表性的工作，这些工作为深度学习时代的 CV 打下了良好的基础，提供了许多基础工具和数据集的构建方式。同时老师也更加注重深度学习的基础知识，如 BatchNorm 的特性和与其他 Norm 的区别，许多人仅仅只是会 PyTorch 的积木搭建，但是对于这些基础知识的原理和性质却不甚了解，导致在实际使用中遇到问题时无法解决，王老师在这方面往往提出 intuitive 的问题，引人深思。

我是在大三下学期选修了这门课程，即使我已经具有了一定的深度学习基础，但是我仍然很享受上课看回放的过程，因为对于许多已经了解的知识，王老师会再度给出解释，总是让我在同一个地方有不同的收获。

我在本学期期中考试之前偶然了解到曾经有学长撰写了一本笔记，但是许多内容已经进行了更新或者删改，因此我联系上林晓疏（笔名）学长，获取了这份笔记的源代码，并在此基础上进行更新，以飨后人。

该笔记按照讲授先后顺序进行排列，但是章节编排按照知识结构划分，因此章节划分可能与课程进度有所不同。同时本笔记不能替代课程，只是对这部分知识的总结和思考，初学者建议与课程回放配合食用（老手感觉不用了）。

Yutong Liang  
2024 年 4 月 24 日

## 目录

1	Edge Detection	4
1.1	What is an Edge? . . . . .	4
1.2	Criteria for Optimal Edge Detection . . . . .	4
1.3	Non-Maximal Suppression (NMS) . . . . .	4
1.4	A Simplified Version of NMS . . . . .	5
1.5	Hysteresis Thresholding . . . . .	5
2	Keypoint Detection	6
2.1	The Basic Idea of Harris Corner . . . . .	6
2.2	Harris Corner . . . . .	6
2.3	equivariant V.S. invariant . . . . .	7
2.4	How to prove Harris detector is equivariant? . . . . .	7
2.5	How to do NMS with corner-response-function? . . . . .	7
2.6	Scale Invariant Detectors . . . . .	8
3	Line Fitting	9
3.1	Least Square Method . . . . .	9
3.2	RANSAC . . . . .	9
3.3	RANSAC calculation . . . . .	9
3.4	Hough Transform . . . . .	10
4	Convolutional Neural Network	11
4.1	CNN and MaxPooling . . . . .	11
4.2	Summary of CNN-based classification networks . . . . .	11
4.3	Pooling Layer affects the parameter number . . . . .	11
4.4	Comparison of MLP and CNN . . . . .	12
5	CNN Training	14
5.1	Data Preparation . . . . .	14
5.2	Weight Initialization . . . . .	14
5.3	Start Optimization . . . . .	15
5.4	Learning Rate Schedule . . . . .	16
6	CNN Training improvement	18
6.1	Underfitting . . . . .	18
6.1.1	Batch Normalization . . . . .	18
6.2	Some Questions . . . . .	20

6.2.1	ResNet or Skip Links . . . . .	21
6.3	Overfitting . . . . .	23
6.3.1	Data Augmentation . . . . .	23
6.3.2	Regularization . . . . .	24
6.3.3	Dropout . . . . .	24
6.3.4	BatchNorm 作为正则化 . . . . .	25
6.4	Summary of Mitigating Overfitting . . . . .	25
7	Classification	26
7.1	Nearest Neighbour Classifier . . . . .	26
7.2	Using CNN for image Classification . . . . .	26
7.2.1	SoftMax . . . . .	26
7.3	Cross Entropy Loss V.S. Accuracy . . . . .	27
8	CNNs for Image Classification	28
8.1	Reception Field . . . . .	28
9	Segmentation	29
10	3D Vision	31
10.1	Intrinsics . . . . .	31
10.2	Extinsics . . . . .	32
10.3	weak perspective . . . . .	34
11	Camera Calibration	36
11.1	Some Problems with Camera . . . . .	37
12	Single View Geometry	39
12.1	Transformation in $\mathbb{R}^2$ . . . . .	39
12.2	Vanishing Points . . . . .	40
13	Epipolar Geometry	42
13.1	Epipolar Constraint . . . . .	42
14	3D data	46
14.1	Mesh . . . . .	46
14.2	Point Cloud . . . . .	46
14.3	CD vs EMD . . . . .	47
14.4	Implicit Shape . . . . .	47

15 3D Deep Learning	48
15.1 PointNet++ V.S. Conv . . . . .	49
16 Object Detection and Instance Segmentation	50
16.1 任务简介 . . . . .	50
16.2 Region-based CNN . . . . .	51
16.3 Fast R-CNN . . . . .	52
16.4 Faster R-CNN . . . . .	53
16.5 two-stage detector and one-stage detector . . . . .	53
16.6 Evaluation Metric: mAP . . . . .	53
17 Instance Segmentation	55
17.1 Mask R-CNN . . . . .	55
17.2 3D Object Detection and Instance Segmentaiton . . . . .	55
18 Pose and Motion	56
18.1 Beyond Detection: Pose . . . . .	56
18.2 Euler Angle . . . . .	56
18.3 Axis Angle . . . . .	56
18.4 Quaternion . . . . .	56
19 Instance-Level 6D Object Pose Estimation	58
19.1 Beyond Object Pose . . . . .	59
20 Motion	60
21 RNN	61
21.1 Characer-Level Language Model . . . . .	61
21.2 Vanilla RNN Gradient Flow . . . . .	62
21.3 LSTM . . . . .	62
22 Video analysis	63
23 Generative Models	64
23.1 Approximate density:Variational Autoencoder . . . . .	64
23.2 重学 VAE . . . . .	65
23.3 VAE . . . . .	67
23.3.1 ELBO . . . . .	68
24 GAN	69

## 目录

---

A Condition Number	70
B Transfromation in $\mathbb{R}^n$	71
C DOF and rank in essential matrix and fundamental matrix	73
D QR Decomposition	74

## 1 Edge Detection

### 1.1 What is an Edge?

“边缘”是图像中的一个区域，在这个区域中，沿着图像的一个方向，像素强度值（或者说对比度）发生了“显著”的变化，而在其正交方向上，像素强度值（或对比度）几乎没有变化。

### 1.2 Criteria for Optimal Edge Detection

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{FP} + \text{TN} + \text{FN}} \quad (1)$$

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \quad (2)$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \quad (3)$$

Precision 和 Recall 都代表着你检测出的真正边缘所占比例，但是 Precision 的分母是你检测出的边缘，Recall 的分母是真正的边缘。

### 1.3 Non-Maximal Suppression (NMS)

非最大值抑制，顾名思义，就是抑制非最大值，这里的最大值指的是梯度的局部最大值。

在计算出了所有点的梯度之后，会有很多像素的梯度大于设定的阈值，而我们希望最后得出的边缘像素真的看起来像一条线而不是一块区域，所以 NMS 的目的是为了抑制那些不是边缘的像素，只保留那些是边缘的像素。

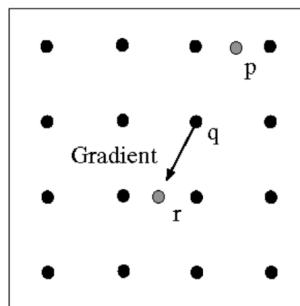


图 1: NMS 示意图

对于一个边缘像素的候选点，我们认为它是边缘当：它比它梯度方向的两个点  $q + \nabla q$  和  $q - \nabla q$  的梯度值大，也就是这个点的梯度大小是局部最大值的时候。

计算这个点梯度方向的点的梯度值可以使用双线性插值法，就是把这个点周围的四个点的梯度按照横纵距离反比加权。

当然，NMS 是一个思想而不是针对边缘检测的算法，比如对于 keypoint detection, object detection (like YOLO) 都可以使用 NMS，实现的思路都很类似，使用一个打分函数看这个备选点 (bounding box) 是不是比跟它相邻 (冲突) 的点 (bounding box) 好，如果是就保留，否则就抑制。

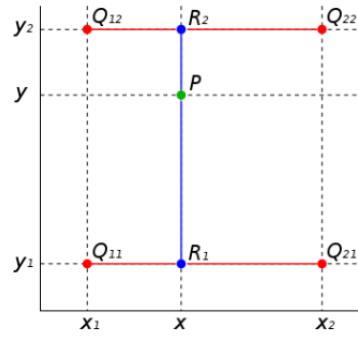


图 2: 双线性插值

#### 1.4 A Simplified Version of NMS

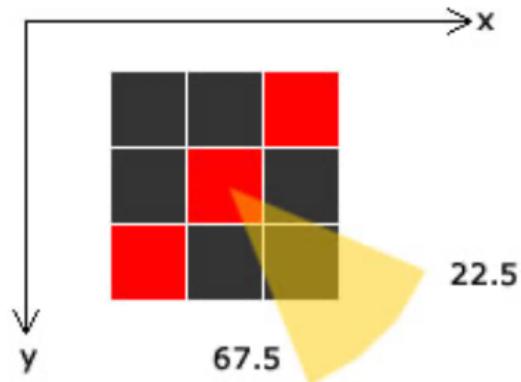


图 3: 简化版本的双线性插值

一个 NMS 的简化版本是把双线性插值省去, 直接让这个像素的梯度大于它梯度方向的那两个相邻像素的梯度.

#### 1.5 Hysteresis Thresholding

使用高阈值 (maxVal) 开始边缘曲线, 使用低阈值 (minVal) 继续它们。

- Pixels with gradient magnitudes  $> \text{maxVal}$  should be reserved.
- Pixels with gradient magnitudes  $< \text{minVal}$  should be removed.

How to decide maxVal and minVal? Examples:

- $\text{maxVal} = 0.3 \times \text{average magnitude of the pixels that pass NMS}$
- $\text{minVal} = 0.1 \times \text{average magnitude of the pixels that pass NMS}$

## 2 Keypoint Detection

### 2.1 The Basic Idea of Harris Corner

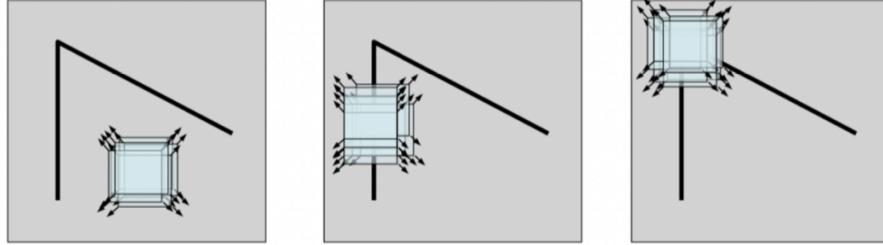


图 4: 移动窗口

Move a window and explore intensity changes within the window.

Corner: significant change in all directions.

### 2.2 Harris Corner

一个 window, 给定它的移动方向  $(u, v)$ :

$$\begin{aligned}
 E(u, v) &= \sum_{x,y} w(x, y)[I(x + u, y + v) - I(x, y)]^2 \\
 &\approx \sum_{x,y} w(x, y)[I(x, y) + uI_x + vI_y - I(x, y)]^2 \\
 &= \sum_{x,y} w(x, y)[uI_x + vI_y]^2 \\
 &= w * \begin{bmatrix} u & v \end{bmatrix} \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} \\
 &= \begin{bmatrix} u & v \end{bmatrix} \begin{bmatrix} w * I_x^2 & w * I_x I_y \\ w * I_x I_y & w * I_y^2 \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} \\
 &= \begin{bmatrix} u & v \end{bmatrix} R^{-1} \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} R \begin{bmatrix} u \\ v \end{bmatrix} \\
 &= \lambda_1 u_R^2 + \lambda_2 v_R^2
 \end{aligned} \tag{4}$$

根据这两个特征值的大小可以判断这个点是不是角点.

这个点是角点一般需要满足:

- $\lambda_1, \lambda_2 > b$
- $\frac{1}{k} < \frac{\lambda_1}{\lambda_2} < k$

一个快速的判断公式:

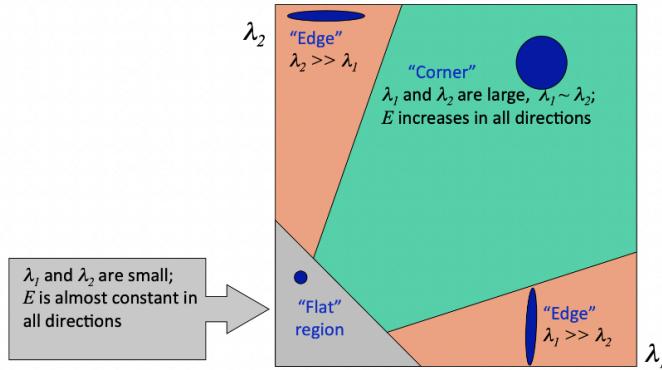


图 5: 特征值大小和这个点的是什么种类的点的关系

$$\begin{aligned}
 \theta &= \frac{1}{2}(\lambda_1\lambda_2 - \alpha(\lambda_1 + \lambda_2)^2) + \frac{1}{2}(\lambda_1\lambda_2 - 2t) \\
 &= \lambda_1\lambda_2 - \alpha(\lambda_1 + \lambda_2)^2 - t \\
 &= \det(R) - \alpha\text{Trace}(R)^2 - t
 \end{aligned} \tag{5}$$

其中  $\alpha \in [0.04, 0.06]$ ,  $t \in [0.01, 0.03]$ .

Harris Corner 对平移和图像旋转是 equivariant 的, 对规模不是 equivariant 的.

### 2.3 equivariant V.S. invariant

等变 (equivariant):  $F(TX) = T(F(x))$ , 对于 translation 和 rotation 是等变的.

不变 (invariant):  $F(T(X)) = F(X)$ , 也就是对于不同位置导出的角点还是那样, 所以其实我们想要的是等变, 也就是对于不同位置导出的角点做了同样的变化.

### 2.4 How to prove Harris detector is equivariant?

只要说明角点检测函数也是 equivariant 即可.

角点检测函数包括了求导和卷积两个操作, 显然求导是 equivariant 的, 因为导数会随着 trans 和 rot 做相同的变化.

很有趣的是卷积也是 equivariant 的: 当你的 filter function 是各向同性的, 那么这个卷积就是 equivariant 的; 但是如果是一个椭圆形的 window, 那这个卷积就不是 equivariant 的了.

这个高光说明不是环境 Illumination Invariant 的

### 2.5 How to do NMS with corner-response-function?

一个简单的想法:

先给出一个阈值, 把所有 response 排序, 成为一个 list, 从上到下按顺序把这个 pixel 周围的大于阈值的踢出 list. 这个跟之前的 NMS 区别在于之前需要一条边, 现在只需要一个点, 那么现在比之前踢出的像素点更多.



图 6: Illumination invariant

## 2.6 Scale Invariant Detectors

Harris-Laplacian, SIFT (Lowe)

### 3 Line Fitting

#### 3.1 Least Square Method

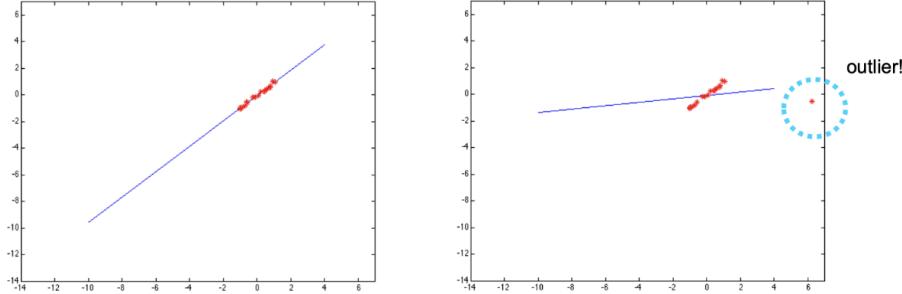


图 7: Least Square Method is not robust to outliers

Robust to small noises but sensitive to outliers.

#### 3.2 RANSAC

RANSAC: RANdom SAmple Consensus

Idea: we need to find a line that has the largest supporters (or inliers)

RANSAC loop:

假设这个直线(平面)需要两个(n个)点来确定。

1. 随机选择  $k$  组能确定这个直线的点, 也就是在所有点里面选出一个  $k \times 2$  的矩阵
2. 对每一组点计算出一条直线
3. 对每一组点的直线计算出所有点到这条直线的距离, 如果小于阈值, 则认为这个点是这条直线的 inlier
4. 找到最大的 inlier 数量的直线, 如果大于阈值, 则认为这条直线是最优的
5. 对这个最优的直线, 用这个直线所有的 inlier 重新计算一次直线

#### 3.3 RANSAC calculation

假设我们有所有 inliner 占比为  $w$  的先验知识, 同时希望有不低于  $p$  的概率能够找到一个最优的直线, 那么我们需要多少次迭代呢?

$$\Pr[\text{一组点全部是 inliner}] = w^n \quad (6)$$

如果一组点中有一个点是 outlier, 那么我们称这组点 fail.

$$\Pr[k \text{ 组点全部 fail}] = (1 - w^n)^k \quad (7)$$

我们希望  $k$  组点全部 fail 的概率小于  $1 - p$ .

$$(1 - w^n)^k < 1 - p \Rightarrow k > \frac{\log(1 - p)}{\log(1 - w^n)} \quad (8)$$

### 3.4 Hough Transform

其实就是把一条直线从实际空间的表示转换到参数空间的表示. 但是如果存在垂直的直线, 可能需要考虑使用极坐标来作为参数空间.

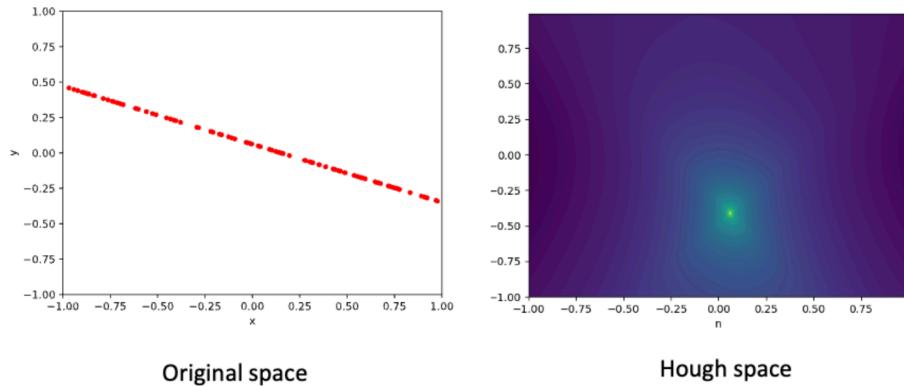


图 8: Hough Transform w/o Noise

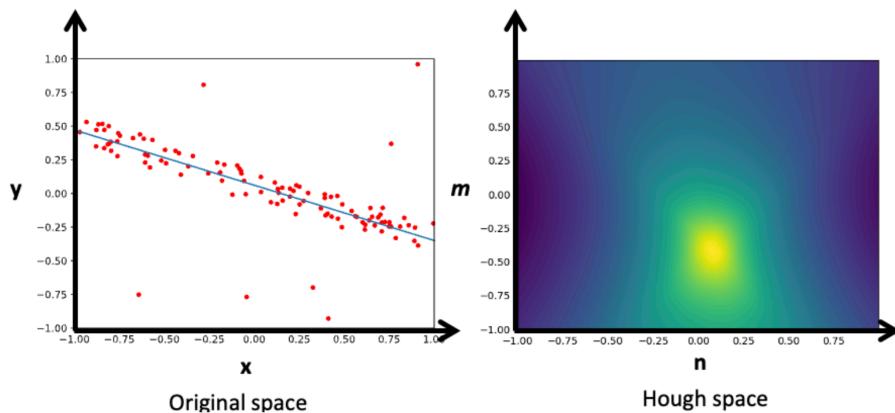


图 9: Hough Transform w/ Noise and Outliers

## 4 Convolutional Neural Network

### 4.1 CNN and MaxPooling

假定输入是  $W_1 \times H_1 \times C$  的矩阵, 可以看作一张图片的宽度, 高度, 通道数. 卷积层需要四个超参数: filter 的数量  $K$  和大小  $F$ , 步长  $S$  和零填充参数  $P$ . 经过卷积层后, 原本的输入变成  $W_2 \times H_2 \times K$ , 其中

$$\begin{aligned} W_2 &= \frac{W_1 - F + 2P}{S} + 1 \\ H_2 &= \frac{H_1 - F + 2P}{S} + 1 \end{aligned} \tag{9}$$

一共需要  $F^2CK + K$  个参数, 其中额外的  $K$  是每层的 bias.

选用卷积核, 可以降低 feature map 的大小. 如果直接将图片进行 flatten, 将会导致大量信息的丢失.

所谓池化操作, 就是一种降采样, 可以降低图片的大小. 比如采用  $2 \times 2$  的大小进行最大值池化, 就是将每个  $2 \times 2$  范围内最大的像素的值作为池化后这个像素的取值, 将原来的图片缩减到四分之一大小. 池化可以增大感受野. 虽然可以通过步长大于一的卷积来代替池化, 但是池化不需要参数, 更容易优化.

假定输入是  $W_1 \times H_1 \times C$  的矩阵, 则池化层需要两个超参数: 大小  $F$  和步长  $S$ . 池化结果是产生  $W_2 \times H_2 \times K$  的矩阵, 其中

$$\begin{aligned} W_2 &= \frac{W_1 - F}{S} + 1 \\ H_2 &= \frac{H_1 - F}{S} + 1 \end{aligned} \tag{10}$$

池化层无需参数.

### 4.2 Summary of CNN-based classification networks

ConvNets 堆叠了卷积层, 池化层和全连接层, 倾向于使用更小的卷积核和更深的网络结构, 尽量避免使用池化层和全连接层 (只使用 Conv 层). 其网络结构大致如下:

$[(\text{Conv-ReLu}) * N - \text{Pool?}] * M - (\text{FC-ReLu}) * K - \text{SoftMax}$

其中  $N$  一般不超过 5,  $M$  比较大,  $0 \leq K \leq 2$ . 但最近的网络如 ResNet, GoogleNet 等也开始突破这些范围.

### 4.3 Pooling Layer affects the parameter number

三层  $3 \times 3$  比  $7 \times 7$  多了两个 relu, 非线性性质更好并且参数变少

每 MaxPooling 一次, 长宽减半, Channel 数量变成两倍

$Param = k^2C^2$ , 参数变成四倍

$Mem = mnC$ , 显存变成二分之一

#### 4.4 Comparison of MLP and CNN

如果输入为  $W_1 \times H_1 \times C$  的矩阵, 输出  $W_2 \times H_2 \times K$  的矩阵, 那么 FC 需要  $W_1 W_2 H_1 H_2 C K$  个参数, 一层卷积核大小为  $F$  的 CNN 需要  $F^2 C K$  个参数. 后者一般比前者小得多.

对于一维情形,  $h \in \mathbb{R}^m, x \in \mathbb{R}^n$ , 则  $y = h * x$  可以被表示为矩阵乘法:

$$y = h * x = \begin{pmatrix} h_1 & 0 & \cdots & 0 & 0 \\ h_2 & h_1 & & \vdots & \vdots \\ h_3 & h_2 & \cdots & 0 & 0 \\ \vdots & h_3 & \cdots & h_1 & 0 \\ h_{m-1} & \vdots & \ddots & h_2 & h_1 \\ h_m & h_{m-1} & & \vdots & h_2 \\ 0 & h_m & \ddots & h_{m-2} & \vdots \\ 0 & 0 & \cdots & h_{m-1} & h_{m-2} \\ \vdots & \vdots & & h_m & h_{m-1} \\ \vdots & \vdots & \cdots & 0 & h_m \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{pmatrix} \quad (11)$$

左侧的矩阵被称为 Toeplitz 矩阵. 由于深度学习当中的卷积参数是学习而来, 故不需要像传统的卷积操作一样进行翻转. 二维情形的卷积则是一个 double block circulant matrix.

由于我们选取较小的卷积核, 所以前后层网络之间的连接更为稀疏. 而且有更加明显的 Parameter Sharing 效应, 即对每块区域采取相同的参数进行处理.

**FC 和 CNN 两者哪个表达能力更强呢?**

很显然是 FC 更强, 因为 FC 的表达范围是 CNN 的超集. 但事实上, 使用 Conv 的结果远好于 FC/MLP. 那么问题出在哪里呢?

一个显而易见的答案是, FC 需要的参数量过于庞大了, 一层可能需要上亿甚至更多的参数, 这使得其非常难以被优化. 另外一个非常重要的原因是, 它并没有突出我们在视觉任务中目标的特点, 即等变性 (equivariance).

我们知道, 在目标分类等任务当中, 将图片进行轻微平移, 旋转, 改变亮度等操作, 并不会影响结果. 但是, 它们都改变了输入的几乎每一个值. 而即使移动了一个像素, FC 的输出都将天差地别. 换言之, 我们要求 FC 将在它看来输出完全不同的图像归为一类, 这样的矩阵是非常难以寻找的, 会在优化过程中产生极大的困难.

那么 CNN 的表现如何呢? 我们先解释上文 equivariance 的含义. 其一般定义为:

$$S_A[\phi(X)] = \phi[T_A(X)] \quad (12)$$

这里  $A$  指代某种操作, 而  $T_A, S_A$  分别代表  $X, \phi(X)$  空间下的变换. 例如将  $A$  看作左移一像素, 忽略边界就有  $T[\phi(X)] = \phi[T(X)]$ . 不变性是等变性的特殊情形.

我们需要指出的是, 在这里 Parameter Sharing 就等同于 Equivariance to Translation. 我们用同样的参数处理每一个局部, 那么忽略掉边界后, 2D Conv 就是等变的.

举个例子, 当处理图像的时候, 在第一个卷积层进行边的探测是非常有用的, 而同样的边或多或少地会出现在图片的其他位置, 所以在整个图片范围内进行参数共享是非常可行的操作. 换言之,

对于反映同种信息的 pixel 的识别, 在不同位置应该是相同的, 这种人类视觉给出的先验知识才是 Conv 的根本思路.

但需要强调的是, Conv 并不是万能的, 一个 layer 作为一个函数, 其需要具有何种性质与目标的性质高度相关. 我们使用 CNN 正是因为当前我们的目标具有这种不变性. 但有时你需要位置相关的处理, 比如统计一张照片上的绵羊数量, 那显然就不能将不同地方的绵羊做相同处理. 这方面的工作就是 semi-Conv 的工作, 发表于 CVPR 2018. 同样, 它也不具有图像缩放, 图像旋转等情形下的不变性, 需要引入其他机制进行处理.

## 5 CNN Training

### 5.1 Data Preparation

不仅是在 CV 当中, 在诸如数据科学等学科之中, 对数据进行分析之前通常要进行预处理 (processing). 以处理 CIFAR-10 的  $32 \times 32 \times 3$  数据集, 我们有逐通道减去均值 (VGG) 或逐通道减去均值然后除以标准差 (ResNet).

标准化有什么作用呢? 例如, 如果神经元的输入都是正数, 那么权重的导数就会都是正的或者负的. 而非标准化的数据可能具有很大的数值, 对于权重矩阵的微小变化非常敏感, 难以训练.

### 5.2 Weight Initialization

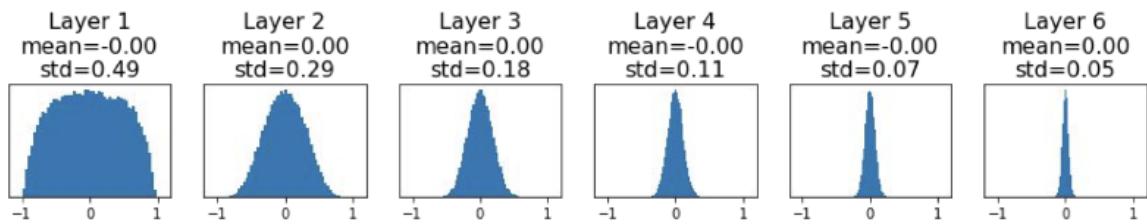


图 10: 梯度方差示意图

权重矩阵的初始化并不是一件随意的事情, 它代表了我们优化的起点. 一个最简单的想法是: 用一些小的零均值随机数, 比如调用  $0.01 * np.random.randn(Din, Dout)$  生成参数矩阵. 但是这样做, 由于参数都是一些服从高斯分布的随机数, 这会导致梯度的方差逐层递减, 最后梯度非常小, 训练困难, 如图所示10.

一个优化方法是每层都进行标准化, 同时令参数的方差为  $\frac{1}{D_{in}}$ .

数学推导:

假设  $y = x_1w_1 + \dots + x_nw_n$ , 我们希望  $Var(y) = Var(x_i)$ , 这样可以保证梯度的方差不会随着层数的增加而变小. 那么:

$$\begin{aligned}
 Var(y) &= Var(x_1w_1 + \dots + x_nw_n) \\
 &= Var(x_1w_1) + \dots + Var(x_nw_n) \\
 &= Var(x_1)Var(w_1) + \dots + Var(x_n)Var(w_n) \\
 &= nVar(x)Var(w)
 \end{aligned} \tag{13}$$

$$\Rightarrow Var(w) = \frac{1}{n}$$

在使用 ReLU 时, 可以视为去掉一半神经元, 参数方差为  $\sqrt{\frac{2}{D_{in}}}$ .

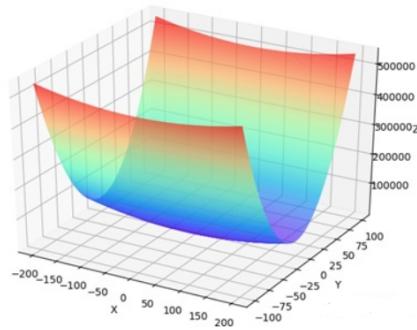


图 11: 山谷示意图

### 5.3 Start Optimization

前面我们提到过用 SGD 的方法进行训练,但在矩阵的 Condition Number<sup>1</sup> 较大的时候,SGD 的效果仍然不够好. 对 SGD 来说, 山谷和鞍点仍然是巨大的麻烦. 所谓山谷, 就是类似右图的地形<sup>2</sup>:

如果梯度下降进入如图所示的地形, 则由于山谷两侧的梯度指向谷底, 因此如果山谷中比较平缓, 则很可能在两侧山壁来回打转, 或需要很长时间才能到达谷底的最小值.

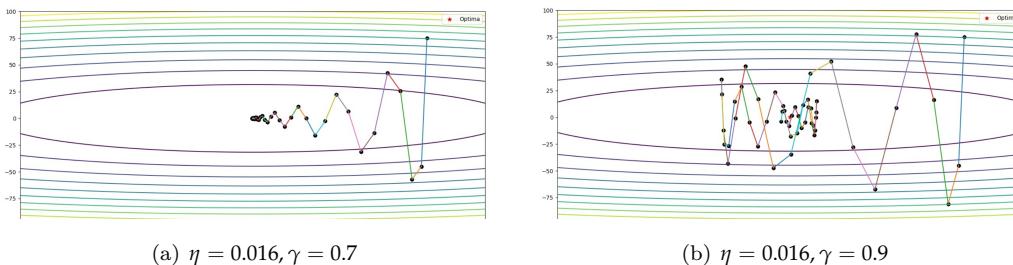
因此, 在 SGD 的基础上我们可以添加动量方法 (Momentum)<sup>3</sup>, 所谓动量方法, 就是对历史梯度进行记录, 并以一定权重影响当前的梯度.

具体来说, 结合物理学上的动量思想, 在梯度下降问题中引入动量项  $m$  和折扣因子  $\gamma$ , 公式变为

$$\begin{aligned} m &\leftarrow \gamma m + \eta \nabla_{\theta} \mathcal{L}(\theta) \\ \theta &\leftarrow \theta - m \end{aligned} \tag{14}$$

其中,  $\gamma$  表示历史梯度的影响力,  $\gamma$  越大, 历史梯度对现在的影响也越大. 直观上来说, 要是当前时刻的梯度与历史梯度方向趋近, 这种趋势会在当前时刻加强, 否则当前时刻的梯度方向减弱.

我们用 SGD+Momentum 来对图11进行梯度下降. 下面分别是两个不同参数情形时的俯视图. 如果不加 momentum, 则可以想见点将会以近乎垂直于等高线的方向来回震荡. 左图中每个较长的梯度下降之后, 跟着的一段较短并且一定程度矫正了方向, 这正是历史梯度的作用. 而右图则是  $\gamma$  较大的情形, 此时历史梯度的权重过大, 积重难返, 优化效果要差一些.



再来考察另一个麻烦: 鞍点. 存在鞍点意味着此处的 Hessian Matrix 有正负特征值, 是不定的,

<sup>1</sup>有关矩阵 Condition Number 的简单叙述, 请见附录.

<sup>2</sup>本小节的配图和叙述参考了 @ 郑思座的[这篇文章](#). 此文还介绍了牛顿动量法和自然梯度法.

<sup>3</sup>为什么动量优化 work? 可以参考这个可视化可交互网站:[Why Momentum Really Works](#)

因而随机梯度下降很可能无法找到下降的方向. 由于深度学习实践中普遍维度较高, 比如在 1000 维当中只有两个维度不是 local minimum, 那么 SGD 也很难跳出这里. 这也是偶尔会看到 loss 长时间不动而突然间骤减的可能原因.

若用  $G_t$  表示第  $t$  轮迭代的动量,  $g_t = \eta \nabla_{\theta} \mathcal{L}(\theta)$  表示第  $t$  轮迭代的更新量, 当  $t \rightarrow \infty$ ,  $G_{\infty} = \frac{\eta_0}{1-\gamma}$ , 该式的含义是如果梯度保持不变, 最终的更新速度会是梯度项乘以学习率的  $\frac{1}{1-\gamma}$  倍. 举例来说,  $\gamma = 0.9$  时, 动量算法最终的更新速度是普通梯度下降法的 10 倍, 意味着在穿越“平原”和“平缓山谷”从而摆脱局部最小值时, 动量法更有优势.

在书写代码时, 我们通常这样写:

$$\begin{aligned} v_{t+1} &\leftarrow \rho v_t + \nabla f(x_t) \\ x_{t+1} &\leftarrow x_t - \alpha v_{t+1} \end{aligned} \tag{15}$$

也就是将 velocity 项作为梯度的运行时均值. 一般取  $\rho = 0.9$  或  $\rho = 0.99$ .

Adam 就是一种应用了 Momentum 的优化方法. 具体如下图.

```

first_moment = 0
second_moment = 0
for t in range(1, num_iterations):
    dx = compute_gradient(x)
    first_moment = beta1 * first_moment + (1 - beta1) * dx
    second_moment = beta2 * second_moment + (1 - beta2) * dx * dx
    first_unbias = first_moment / (1 - beta1 ** t)
    second_unbias = second_moment / (1 - beta2 ** t)
    x -= learning_rate * first_unbias / (np.sqrt(second_unbias) + 1e-7))

```

**Momentum**

**Bias correction**

**AdaGrad / RMSProp**

**Bias correction for the fact that first and second moment estimates start at zero**

**Adam with  $\text{beta1} = 0.9$ ,  $\text{beta2} = 0.999$ , and  $\text{learning\_rate} = 1e-3$  or  $5e-4$  is a great starting point for many models!**

Kingma and Ba, "Adam: A method for stochastic optimization", ICLR 2015

图 12: Adam

## 5.4 Learning Rate Schedule

在神经网络的训练过程中, 如何设定学习率的变化策略也是很重要的, 因为越靠近最小值, 其对学习率的取值可能越敏感, 需要更加精细地取值. 我们先来看一张不同大小的学习率对于 loss function 的影响. 由此可以看出, 好的学习率就应该像临界阻尼一样, 让阻尼振荡又快又稳地回到平衡位置... (胡言乱语)

较低的学习率会使得训练所需时间较长, 而过高的学习率会限制最终的结果. 如果学习率非常高, 可能 loss 反而会上升<sup>4</sup>.

一种策略是, 在固定轮数之后, 缩减学习率. 比如 ResNet 在每 30 个 epoch<sup>5</sup> 后将学习率乘以

<sup>4</sup> 这里老师在课上说不一定, 并提问: 对于 classification 问题, 也会上升吗? 有想法的同学欢迎联系笔者, 我将在此添加您的真知灼见. 两年之后的笔者对这个问题也没有想法 orz

<sup>5</sup> 此处王鹤老师详细解释了 iteration 和 epoch 两个概念的区别. 一次 iteration, 就是跑完一次 mini-batch 的过程. 而 epoch 的含义则更加灵活. 它可指代: 1. 训练完整个 dataset 称为一个 epoch. 例如将 3200 张图片分为 100 个 mini-batch, 则 1 epoch = 100 iteration. 2. 每个 epoch 结束保存一次模型. 3. epoch 作为绘制 curve 的单元. 4. 进行 model validation 的单元.

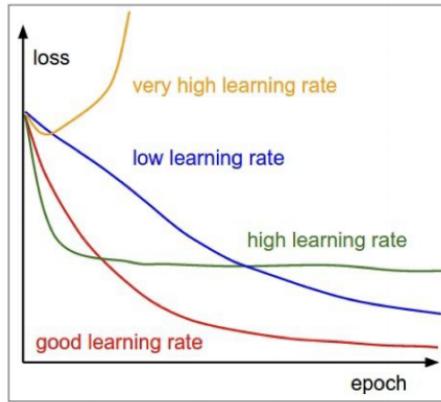


图 13: 不同学习率对损失函数的影响

0.1. 除此之外还有多种形式, 如余弦, 线性递减, 除以轮数的平方根等.

除了一定轮数之后的递减策略, 在训练伊始, 也有“热身”的递增策略. 过高的初始学习率对优化会产生不利影响, 但在约前 5000 次迭代中线性增长学习率可以避免这种影响. 另外, 有一个经验定律: 如果将 batch 的大小增长了  $N$ , 那么学习率也变为  $N$  倍.

总之, 对于学习率策略而言, Adam+ 默认学习率在多数情形下都是一个默认较好的选择, 甚至它对于常数学习率也常常运行得不错. 而 SGD+Momentum 的组合可以取得比 Adam 更好的效果, 但恐怕要在学习率策略上花费更多头发. (余弦函数是个不错的较少参数的策略.)

## 6 CNN Training improvement

### 6.1 Underfitting

#### 6.1.1 Batch Normalization

所谓 Batch Normalization, 中文译名为“批标准化”. 其基本操作是在进入激活函数层之前, 将数据逐通道进行标准化. 假设输入为  $n$  个数据, 通道数为  $D$ , 如右图.

那么我们需要计算

$$\begin{aligned}\mu_j &= \frac{1}{N} \sum_{i=1}^N x_{i,j} \\ \sigma_j^2 &= \frac{1}{N} (x_{i,j} - \mu_j)^2 \\ \hat{x}_{i,j} &= \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \epsilon}}\end{aligned}\tag{16}$$

其中  $\epsilon$  是为防止方差为 0 而添加的小量. 这一步是标准化的过程, 此层还需要进行一步处理:

$$y_{i,j} = \gamma_j \hat{x}_{i,j} + \beta_j\tag{17}$$

我们需要学习  $\gamma, \beta$  这两个  $D$  维参数.

那么, 为什么要进行 BN 操作呢?

在机器学习领域有个很重要的假设: 独立同分布假设, 就是假设训练数据和测试数据是满足相同分布的, 这是通过训练数据获得的模型能够在测试集获得好的效果的一个基本保障. 那 Batch-Norm 的作用是什么呢? BatchNorm 的作用就是在深度神经网络训练过程中使得每一层神经网络的输入保持相同分布.

对于深度学习这种包含很多隐层的网络结构, 在训练过程中, 因为各层参数不停在变化, 所以每个隐层都会面临 covariate shift 的问题, 也就是在训练过程中, 隐层的输入分布经常发生变化, 这就是所谓的“Internal Covariate Shift”, Internal 指的是深层网络的隐层, 是发生在网络内部的事情.

BatchNorm 的基本思想就是: 能不能让每个隐层节点的激活输入分布固定下来呢? 这样就避免了“Internal Covariate Shift”问题了.

之前的研究表明如果在图像处理中对输入图像进行白化操作的话——所谓白化, 就是对输入数据分布进行 PCA, 并将各个分量变换到 0 均值, 单位方差的正态分布——那么神经网络会较快收敛. 但是白化操作较难进行, 而且操作不可导. 而 BN 可以理解为对深层神经网络每个隐层神经元的激活值做简化版本的白化操作.

BN 的基本思想其实相当直观: 因为深层神经网络在做非线性变换前的激活输入值随着网络深度加深或者在训练过程中, 其分布逐渐发生偏移或者变动, 之所以训练收敛慢, 一般是整体分布逐渐往非线性函数的取值区间的上下限两端靠近 (对于 Sigmoid 函数来说, 意味着激活输入值是大的负值或正值), 所以这导致反向传播时低层神经网络的梯度消失, 这是训练深层神经网络收敛越来越慢的本质原因, 而 BN 就是通过一定的规范化手段, 把每层神经网络任意神经元这个输入值的分布强行拉回到均值为 0 方差为 1 的标准正态分布, 其实就是把越来越偏的分布强制拉回比较标准的分布, 这样使得激活输入值落在非线性函数对输入比较敏感的区域, 这样输入的小变化就会导致

**Input:**  $x : N \times D$

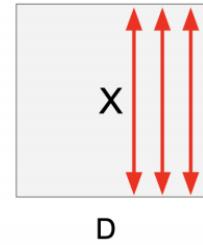


图 14: Input of Batch Normalization

损失函数较大的变化, 意思是这样让梯度变大, 避免梯度消失问题产生, 而且梯度变大意味着学习收敛速度快, 能大大加快训练速度.

对于每个隐层神经元, 把逐渐向非线性函数映射后向取值区间极限饱和区靠拢的输入分布强制拉回到均值为 0 方差为 1 的比较标准的正态分布, 使得非线性变换函数的输入值落入对输入比较敏感的区域, 以此避免梯度消失问题.

以上关于 BN 基本思想的叙述来自参考文献 [?]. 但是在 BN 原论文 [?] 当中, 作者也提出了只进行 16 的问题: 它可能限制模型的表达能力. 以 Sigmoid 为例, 标准化将所有的输入集中于 0 附近, 而这里正是 Sigmoid 的线性近似区域, 这样 Sigmoid 层的作用就和线性层类似了. 因此 BN 需要再加一项平移和拉伸, 并把这两项参数作为学习对象.

总之, BatchNorm 的提出试图通过标准化解决深层神经网络当中内部协变量漂移的问题, 将数据限制在一定范围内, 并通过平移和拉伸操作减少此操作对模型表达能力的影响. 但是, 后来有人提出了新的看法: BatchNorm 对 Internal Covariate Shift 的减缓微乎其微, 它是通过平滑化 loss landscape 来起到作用的.

但 BatchNorm 有一点需要额外注意: 那就是在测试集上运行时, BatchNorm 层并不是计算输入的期望方差, 而是调用在训练时储存的  $\mu, \sigma$ . 具体来说, 按照以下方式迭代每次运算得到的  $\mu_{rms}$ :

$$\begin{cases} \mu'_{rms} = \rho\mu_{rms} + (1 - \rho)\mu_t \\ \sigma'_{rms} = \rho\sigma_{rms} + (1 - \rho)\sigma_t \end{cases} \quad (18)$$

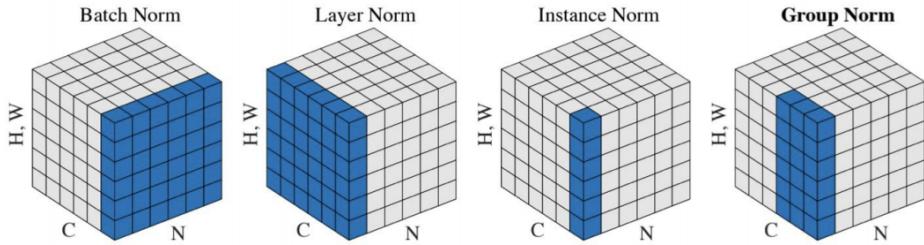
BatchNorm 通常位于 FC 层或 Conv 层之后, 非线性层之前. 它有如下的优点:

1. 使得深层网络更加容易训练, 因为它可以减缓梯度消失.
2. 允许使用更大的学习率, 并且收敛更快. 这是因为如果不加 BatchNorm, 则深层网络容易产生级联效应, 学习率如果比较大, 则前面的 layer 发生变化将会引起后面的层的剧烈变化, 极不稳定.
3. 网络对于参数的初始化要求降低, 因为会进行标准化操作.
4. 训练的时候使用 BatchNorm 具有正则化的特性: 这是因为进行了噪声注入, Batchnorm 在每个批次上计算均值和方差, 这意味着每个批次的统计特征都会略有不同, 这种基于小批量的统计计算引入了噪声, 类似于数据增强或 Dropout 的效果. 这种噪声可以帮助模型避免过拟合, 在一定程度上增强了模型的泛化能力.
5. 测试时不花费开销, 可与 Conv 一同使用

但是同时需要注意: BatchNorm 在训练和测试时行为不同! 这是 Bug 的一个常见原因. 此外, 一旦使用了 BatchNorm, 那么 Batch 就不能太小, 否则会使  $\mu, \sigma$  变得极不稳定, 这可能会导致模型在训练集和测试集上的性能的巨大差异.

这样的特性使得人们自然而然提出了一个问题: 我们能否绕过 BatchSize 的限制? 如果可行, 那么就不会存在训练和测试的差异.

图 15 当中的  $C, N$  分别代表通道数和样本数,  $H, W$  等维度因为地位相同, 被压缩为一个维度, 可以视为图片的宽和高. 左一逐通道对所有样本求平均值, 正是 BatchNorm. 要想脱离 BatchSize 的限制, 就必须不沿  $N$  方向处理.



Wu and He, "Group Normalization", ECCV 2018

图 15: Normalization Techniques

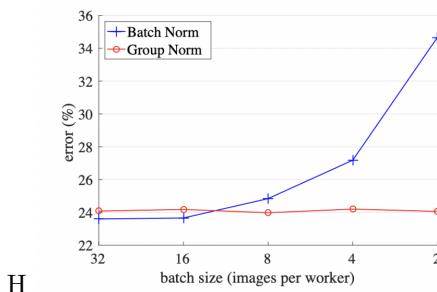


图 16: classification error vs batch sizes

左二是对  $C, H, W$  三个维度求平均, 这其实蕴含了所有数据服从同一分布的假设, 但实际上可能并非如此. 比如, 偏绿色的图片在  $G$  通道上的  $\mu$  更高, 这样做会破坏不同 channel 的差异性, 效果比 BatchNorm 更差, 因此极少应用于 CV 领域, 但在 NLP 领域应用广泛.

右二其实就对应了 BatchSize=1 的情形, 其问题前文已述. 虽然其存在巨大的不稳定性, 但其  $\mu, \sigma$  表征了图片风格, 因此在 Style Transfer 当中常用.

右一是由何恺明等人提出的 GroupNorm, 它将 channel 分组进行计算, 相当于对图 2 和图 3 进行了插值.

图16是 He 在论文中给出的一个 ResNet-50 模型的训练结果, 可以看出在 BatchSize 较大的时候, BatchNorm 表现略优于 GroupNorm, 但随着 BatchSize 减小, 前者的错误率快速增长, 而后者的错误率则没有明显变化. 当训练集的 BatchSize 不得不很小或 Batch 之间差异极大时, 应该选用 GroupNorm.

## 6.2 Some Questions

**问题:** 为什么 BatchNorm 不是对  $N$  上的一维数组进行 norm?

这是因为会破坏 translation invariance, 也就是平移一格变化很大.

**问题:** 后面三个 Norm 需要进行 moving average 吗?

跟 BN 不同, 后面三个在 train 和 test 上都是一样的, 没有 moving avg mean, 因为没有对 batch size 做 norm.

为了防止 test 的时候 batch size 过小, BN 需要在 test 的时候使用 train 的 moving avg mean.

而后面三个 norm 没有这个问题, 因为它们不是对 batch size 做 norm 的.

### 6.2.1 ResNet or Skip Links

介绍 BatchNorm 之后,CNN 网络结构可修改如下:

$[(\text{Conv-BN-ReLu}) * N - \text{Pool?}] * M - (\text{FC-BN-ReLu}) * K - \text{FC} - \text{SoftMax}$

注意最后的 FC-SoftMax 之间无需添加 BN. 那么, 层数是越深越好吗? 我们来看下图.



图 17: 两种深度的神经网络的测试误差 (左) 和训练误差 (右)

56 层神经网络在测试集上误差大于 20 层, 我们可以解释为是过拟合导致, 但其在训练集上的误差竟然也更大, 这就说明更深的神经网络表现不佳并不是过拟合导致. 那么在 CNN 变深的过程中, 究竟产生了什么问题呢?

现在的事实是: 更深的模型具有更强的表达能力 (即更多参数), 但是表现却更差. 我们提出一个假设: 可能是因为深层的神经网络更加难以进行优化. 这个时候我们应该控制变量找出原因.<sup>6</sup> 一种让深层网络至少和浅层网络一样好的方法是, 将浅层网络加上一些恒等变换的层,”变成”深层网络. 从这种意义上讲, 更深的网络不应该比浅的差, 只要中间的一些层学习成恒等变换即可. 这可能预示着, 对于深层网络而言,”恒等”是一个较难学习的特征. 这或许是因为深层网络引入太多非线性表达, 在高维的参数空间中难以学习到恒等吧.<sup>7</sup>

既然如此, 那么一种自然的思路就是: 构造天然的恒等. 假设神经网络非线性层的输入输出维度一致, 那么可以将要拟合的函数分成两个部分:

$$\mathbf{z}^{(l)} = \mathcal{H}(\mathbf{a}^{(l-1)}) = \mathbf{a}^{(l-1)} + \mathcal{F}(\mathbf{a}^{(l-1)}) \quad (19)$$

其中  $\mathcal{F}(\cdot)$  是残差函数. 在网络高层, 学习一个恒等映射  $\mathcal{H}(\mathbf{a}^{(l-1)}) \rightarrow \mathbf{a}^{(l-1)}$  即等价于令残差部分趋近于 0, 即  $\mathcal{F}(\mathbf{a}^{(l-1)}) \rightarrow 0$ .

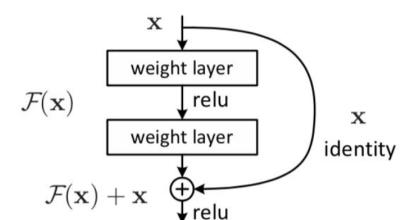


图 18: 残差神经网络单元

残差单元可以以跳层连接的形式实现, 即将单元的输入直接与单元输出加在一起, 然后在激活. 因此残差网络可以轻松地用主流的自动微分深度学习框架实现, 直接使用 BP 算法更新参数.

<sup>6</sup>在接下来将较浅层的输出通过一条旁路直连深层的操作, 老师称之为“寻找一种中间状态”, 即将浅层网络加上恒等变换层”视为”深层网络, 再将两个进行叠加.

<sup>7</sup>本小节余下的内容取自 [?].

实验表明, 残差网络很好地解决了深度神经网络的退化问题, 并在 ImageNet 和 CIFAR-10 等图像任务上取得了非常好的结果, 同等层数的前提下残差网络也收敛得更快. 这使得前馈神经网络可以采用更深的设计. 除此之外, 去除个别神经网络层, 残差网络的表现不会受到显著影响, 这与传统的前馈神经网络大相径庭.

那么, 残差神经网络为什么有效呢?

从梯度反向传播的角度,Skip Link 提供了一条梯度反向传播的旁路, 使得梯度可以更加顺畅地传递到浅层. 我们考虑式19描述的层, 假设残差块不采取任何激活函数, 即

$$\mathbf{a}^{(l)} = \mathbf{z}^{(l)} \quad (20)$$

考虑  $l_1 > l_2$  两个层, 递归地进行展开, 有

$$\mathbf{a}^{(l_2)} = \mathbf{a}^{(l_1)} + \sum_{i=l_1}^{l_2-1} \mathcal{F}(\mathbf{a}^{(i)}) \quad (21)$$

在前向传播时, 输入信号可以从任意低层直接传播到高层. 由于包含了一个天然的恒等映射, 一定程度上可以解决网络退化问题. 这样, 最终的损失函数  $\mathcal{L}$  对某低层输出的梯度可以展开为

$$\frac{\partial \mathcal{L}}{\partial \mathbf{a}^{(l_1)}} = \frac{\partial \mathcal{L}}{\partial \mathbf{a}^{(l_2)}} + \frac{\partial \mathcal{L}}{\partial \mathbf{a}^{(l_2)}} \frac{\partial}{\partial \mathbf{a}^{(l_1)}} \sum_{i=l_1}^{l_2-1} \mathcal{F}(\mathbf{a}^{(i)}) \quad (22)$$

上式说明, 反向传播时, 错误信号可以不经过任何中间权重矩阵变换直接传播到低层, 一定程度上可以缓解梯度弥散问题 (即便中间层矩阵权重很小, 梯度也基本不会消失). 综上, 可以认为残差连接使得信息前后向传播更加顺畅.

在文章 [?] 中, 作者提出了另一种观点, 认为残差神经网络是由一系列路径集合组装成的模型.

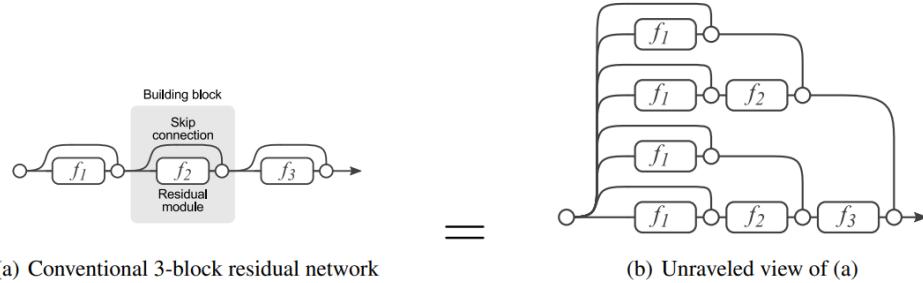


Figure 1: Residual Networks are conventionally shown as (a), which is a natural representation of Equation (1). When we expand this formulation to Equation (6), we obtain an *unraveled view* of a 3-block residual network (b). Circular nodes represent additions. From this view, it is apparent that residual networks have  $O(2^n)$  implicit paths connecting input and output and that adding a block doubles the number of paths.

图 19: 文章给出的配图

Andreas Veit 等人展开了几组实验, 在测试时, 删去残差网络的部分网络层 (即丢弃一部分路径)、或交换某些网络模块的顺序 (改变网络的结构, 丢弃一部分路径的同时引入新路径). 实验结果表明, 网络的表现与正确网络路径数平滑相关 (在路径变化时, 网络表现没有剧烈变化), 这表明残差

网络展开后的路径具有一定的独立性和冗余性,使得残差网络表现得像一个集成模型(ensemble).作者还通过实验表明,残差网络中主要在训练中贡献了梯度的是那些相对较短的路径.

### 6.3 Overfitting

过拟合产生的原因是 data 和 model 之间的不平等. 例如分类问题, 我们给网络输入的数据, 对于机器来说就是某个概念的“定义”. 但同类事物有共性也有差别, 定义应当只取其共性, 但神经网络并不一定能做到这一点. 所以, 减弱 model 增强 data 的手段, 有助于打破这种不平衡.

首先我们来看看在神经网络训练过程中常见的 generalization gap:

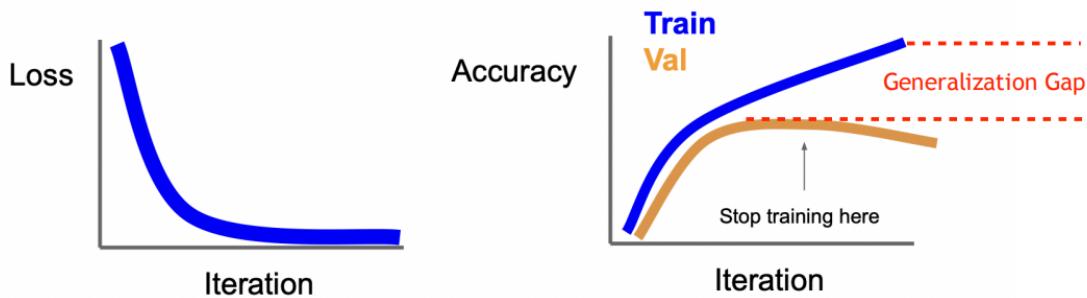


图 20: generalization gap 示意图

所谓 generalization gap, 就是指模型在同分布的训练集和测试集上表现出的准确率差异. 这里右图略有瑕疵, 我们说的 gap 应该是指同一轮测试之后两个集合上表现的差异. 另外需要注意的是: 在训练时即使 loss 一直在减小, 也并不说明在测试集上的准确率在提升, 因为可能有过拟合的问题.

对一个过拟合的模型来说, 它包含了多于区分数据所必须的数量的参数. 为减少过拟合, 我们可以从数据和模型两方面着手处理. 数据方面, 一种自然的想法是, 如果训练集数据充足而且包含较多各种无关特征, 那么模型更不容易过拟合. 最简单的方法是增加数据量, 但有时这是代价很大的, 我们可以对现有的数据进行处理, 获取更多数据. 而在模型方面, 我们将采取一些限制模型复杂度的处理. 数据方面包括 Data Augmentation 等, 而后者我们会提及正则化与 Dropout.

#### 6.3.1 Data Augmentation

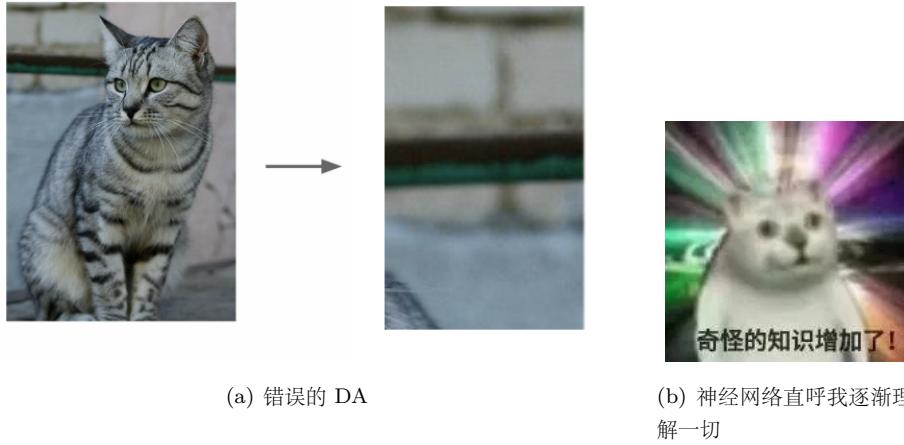
所谓数据增强 (data augmentation), 就是一种利用已有数据人工生成新数据的手段. 常用的手段有翻转 (flip), 旋转 (rotation), 缩放 (scale), 裁剪 (crop), 移位 (translation) 和高斯噪声<sup>8</sup> (Gaussian Noise) 等.

例如, 如果我们的网络需要识别猫的图片, 那么在训练时, 我们可以将一张图片水平翻转, 这当然还是一只猫. 于是我们轻松将数据量翻了个倍.

但是! 要注意的是, 你必须保证数据增强之后, 你所希望学习的特征并没有发生改变. 这隐含着一种对称性, 而诺特定理指出每种对称性对应一种守恒量, 我们不难看出数据增强的依赖的对称性对应的守恒量就是标

<sup>8</sup>当神经网络试图学习可能无用的高频特征(大量出现的模式)时, 通常会发生过度拟合. 具有零均值的高斯噪声基本上在所有频率中具有数据点, 加入图片中可以有效地扭曲高频特征. 这也意味着较低频率的组件(通常是预期数据)也会失真, 但这种困难可以被克服.

(胡言乱语) 即每一只猫是左右对称的. 相对应地, 我们很少对猫图片进行上下翻转. 所以, 进行正确的数据增强是很重要的. 比如下面这种反面教材:



总之, DA 不能太强也不能太弱, 很多时候其程度还是需要人工评判.

在空间位置方面的数据增强手段有缩放, 裁剪, 翻转, 填充, 旋转, 移位, 仿射变换 (affine transformation) 等. 色彩方面, 可以调整亮度 (brightness), 对比度 (contrast), 饱和度 (saturation), 色调 (hue) 等. 此外还可以应用 GAN/Reinforcement Learning 完成数据增强.

### 6.3.2 Regularization

所谓正则化, 就是限制模型复杂度的一种手段. 具体做法就是在损失函数当中添加一项正则项:

$$L(W) = \underbrace{\frac{1}{N} \sum_{i=1}^N L_i(f(x_i, W), y_i)}_{DataLoss: \text{模型预测需要拟合训练集}} + \underbrace{\lambda R(W)}_{Regularization: \text{防止模型在训练集上表现太好}} \quad (23)$$

举个例子, 对于 7 个散点, 我们可以拟合成一条直线, 也可以用六次函数经过每一个点, 但后者常常不是我们想要的, 而正则项的添加就是对模型训练程度的一个限制. 设想一个模型是六次函数, 我们如何限制其表达近似线性的数据? 只需将常数项和一次项系数之外的系数限制在 0 附近即可.

这种想法与一种哲学原理不谋而合: 即 Occam's Razor. 我们常用的正则函数有  $L_2$  Regularization:  $R(W) = \sum_{k,l} W_{k,l}^2$ , 以及  $L_1$  Regularization:  $R(W) = \sum_{k,l} |W_{k,l}|$ , 以及其混合等. 使用时需要注意:  $\lambda$  需要控制正则项的量级, 使其与损失函数相当, 否则如果正则项过大, 会使得网络全力向简单发展; 反之则没有效果.

### 6.3.3 Dropout

简单地说, Dropout 就是以某个概率随机使神经元失效.

通常, Dropout 的概率设置在 0.2 到 0.5 之间, 表示有 20% 到 50% 的概率丢弃一个神经元. 需要注意的是, 在训练完成后并应用到推理/测试阶段时, Dropout 是关闭的, 此时所有神经元都会被使用. 此外, 为了保证神经网络的输出在训练和测试阶段的一致性, 在测试阶段需要将所有神经元的输出乘以一个比例 (这个比例等于 Dropout 的保留概率), 以补偿训练阶段丢弃的部分 (大小).

在实际使用 Dropout 时, 通常是在全连接层和 RNN 层中使用, 尽量避免在卷积层中使用, 因为卷积层具有局部特征的结构, 随机丢弃会破坏这种局部性.

#### 6.3.4 BatchNorm 作为正则化

BatchNorm 使得进入激活函数的输入必须服从特定的高斯分布, 这限制了模型的表达能力, 因此也是一种正则化的手段, 有助于减少过拟合. 使用 BatchNorm 后, 就不需要再添加 Dropout 了.

总之, 纵观各种正则化的方法, 处处透露出 Less is More 的思想, 即越简单的表达方式越好.<sup>9</sup>

### 6.4 Summary of Mitigating Overfitting

原则: 平衡数据的多样性和模型的容量.

方法:

1. Data Augmentation. (从数据的角度)
2. BatchNorm. (模型角度)
3. Regularization. (模型角度)
4. Dropout. (模型角度)

其中 DA 和 BN 用了总是比不用好. 后两者视情况而定. Dropout 一般只用于较大的 FC 层.

---

<sup>9</sup>Less is More, but More is different. (笑)

## 7 Classification

图片分类是 CV 领域的核心问题. 简单来说, 就是给定一张图片, 判断其属于何种分类, 比如是不是猫或狗等等, 这对图片的语义理解非常重要.

但是传统的方法对此类问题难以下手, 因为图片通常是由数字的矩阵来描述, 而从数字到语义有很大的鸿沟, 很难设计某个规则来判定是否属于某类. 比如: 对象不同的姿势, 不同的摄像机视角, 不同的背景信息, 不同的光照条件, 以及对象被隐藏和类内差异等问题.

对于一个好的图片分类器, 应该对上述无关因素不敏感, 而这也是 data augmentation 的意义. 比如 rotation 代表姿势和视角的改变, 颜色改变代表光照的变化等.

对于图片分类, 我们有下列方法: 无参方法有最近邻法, 参数方法则可以采用 CNN.

### 7.1 Nearest Neighbour Classifier

所谓最近邻, 就是将图片视为高维空间的点, 将每个训练数据作为已知点, 定义一种图片间距离的度量, 选取最近的一个(或几个)训练数据的类别作为待判断图片的类别. 这是一种非常低效的方法, 其完美避开了我们上面说到的应具有的标准, 对光照/背景/视角/姿势极为敏感, 正确率极低, 而且需要存储所有训练集. 因此, 实际中从不使用此种方法. 但是最近邻方法在度量学习当中仍有广泛应用.

### 7.2 Using CNN for image Classification

选用 CNN 之后, 我们需要面对的问题有两个: 选取何种网络结构, 以及如何设计损失函数. 如今分类问题的网络范式是 Softmax classifier + cross-entropy loss.<sup>10</sup>

#### 7.2.1 SoftMax

SoftMax 就是一个  $\mathbb{R}^k \rightarrow (0, 1)^k$  的映射.

$$\sigma(z)_i = \frac{\exp \beta z_i}{\sum_j \exp(\beta z_j)} \quad (24)$$

一般取  $\beta = 1$ . 当  $\beta \rightarrow \infty$  时, SoftMax 变成 Argmax.

所以 SoftMax 是 Soft 的 Argmax.

关于 loss 的设计, 如果正确标签是 one-hot 的, 那么我们可以使用负对数概率 (NLL) 作为损失函数. 但是如果 ground truth 也是一个概率分布(有时这是人为的), 那么我们就需要对两个概率分布的距离度量给出定义. 在信息论领域常用的度量是 KL divergence  $D(P \parallel Q)$ , 其定义如下:

$$D(P \parallel Q) = \sum_{x \in \mathcal{X}} P(x) \log \frac{P(x)}{Q(x)}. \quad (25)$$

这个度量并不满足距离的定义, 因为其满足正定性, 而不满足对称性和三角不等式.

<sup>10</sup>对二分类问题, 也可采用 SVM loss. 但是扩展的多分类 SVM loss 在如今已经极少使用了.

我们不难看出

$$D(P \parallel Q) = -\underbrace{\sum_{x \in \mathcal{X}} P(x) \log Q(x)}_{H(P,Q)} - \underbrace{\left( -\sum_{x \in \mathcal{X}} P(x) \log P(x) \right)}_{H(P)}. \quad (26)$$

即 KL divergence 是相对熵和分布  $P$  的熵之差. 如果  $P$  是 ground truth 的分布, 那么第二项成为常数, 就得到了我们的交叉熵损失函数:

$$\mathcal{L}_{CE} = H(P, Q) = -\sum_{x \in \mathcal{X}} P(x) \log Q(x). \quad (27)$$

交叉熵函数在随机初始化时, 取值约为  $\log(\text{sum of classes})$ . 它没有上界, 有下界 0.

所以 CrossEntropyLoss 应该在  $\log$  类别数 开始下降

### 7.3 Cross Entropy Loss V.S. Accuracy

1. CEL 有可能已经降到了  $\log 2$ , acc 仍是 0. 例子:  $\text{Pr} = [0.499, 0.501]$ , 仍然输出错误答案, 但是  $loss = \log 2$  很小

2.  $acc = 100\%$  的时候, CEL 仍然可能是初始化的  $\log(N)$ , 同理举一个例子:  $\text{Pr} = [0.498, 0.001, 0.001]$

综上所述, 两者没有确定关系, 训练一定要同时画两个曲线

## 8 CNNs for Image Classification

当我们分析一个 CNN 的结构时, 需要考虑以下的方面:

1. 表示能力
2. 是否适合任务
3. 是否容易优化
4. 代价

### 8.1 Reception Field

如图, 使用三层  $3 \times 3$  卷积层, 感受野与一层  $7 \times 7$  卷积层相同.

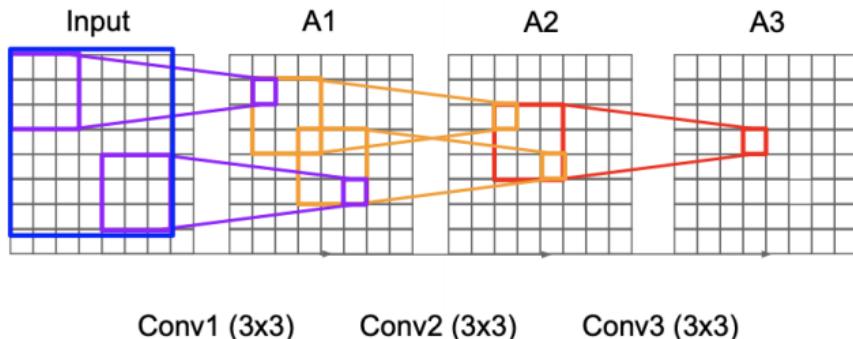


图 21: 三层  $3 \times 3$  卷积核的感受野

感受野是一个很重要的概念, 它表征一个数据可以接收到原图多大范围的信息. 我们这里姑且认为感受野相同则表达能力相同. 我们希望可以在神经网络的中部将整个图片纳入感受野, 这样在后面可以进行全图的 pixel 信息的交流, 有利于结合多个特征. 例如: 结合狗的耳朵和毛色进行判断.

既然三层  $3 \times 3$  卷积层, 感受野与一层  $7 \times 7$  卷积层相同, 那么为什么要选用小而深的网络呢? 其一, 层数增加, 网络的非线性性增加, 分割能力更强. 此外, 参数量也更小 ( $3 \times 3^2 C^2 < 7^2 C^2$ ).

## 9 Segmentation

当图片出现不同的对象 (例如同时出现猫狗), 那么能否将属于同类的 pixel 分到一起? Not a global label.

Image segmentation: 将图片分割为连贯的各个对象, 不关心其语义. 经典方法: grouping and clustering. 后者就是指将相似的数据点分为一组, 然后给予它们一个标识符.

clustering-based segmentation: 对于实际图片, 颜色当然不可能如此理想. 因此我们要选取三个“中心”来代表, 然后将每个 pixel 以离其最近的中心作为标记.

目标函数:

$$c^*, \delta^* = \arg \min_{c, \delta} \frac{1}{N} \sum_j^N \sum_i^k \delta_{ij} (c_i - x_j)^2. \quad (28)$$

很难求解.

很多时候, 这个问题是一个“先有鸡还是先有蛋”的问题. 即何者优先?

破解之法: 如果我们提前知道有  $k$  个 center, 则可以调用 kmeans. 实际就是 random initialization.

coordinate decent 是一个启发式策略 (先找一个方向的最小值, 然后下降). 但无论如何,kmeans 对于初始化非常敏感. 它必须提前知道几个 group, 并且是无监督的.kmeans 实际上进行了对 intensity 进行了 quantum. 除此之外还有 mean shift.<sup>11</sup>

总结:grouping-based: 只关心局部, 不关心语义, 不可能达到 high-level. 是 bottom-up 的方法. 而神经网络则是 top-down 的.

由于纯粹的 Instance segmentation 已经极为少见, 因此今天提及这个词, 多是带语义的.

我们今天的问题:Semantic segmentation. 实际上就是对每个 pixel 进行 classification.

segmentation 时候的一大问题: 对每个 patch, 分类极度依赖上下文.(纯黑的牛) 因此, 将每个 pixel 割裂开是不可能实现分类的.

原先的神经网络: 图像->padding/strided cnn-> 拉成 vector->MLP->softmax-> 类别信息. 这是一个不断提取特征的过程. 但在 segmentation 当中, 我们的输出应该与输入一样大. 怎么把 feature map 先变小再变大? 我们的想法是: 先缩小提取特征 (两头牛), 然后回到原图上以此为原则, 寻找.

NN 核心概念:Auto-Encoder. 自监督. 定义 reconstruction loss

$$\|x - \hat{x}\|_2 \quad (29)$$

Encoder MLP  $f : \mathbb{R}^m \rightarrow \mathbb{R}^h, h \ll m$ . decoder: $g : \mathbb{R}^h \rightarrow \mathbb{R}^m$ . 对于  $H \times W \times 3$  的图像, 实际维度到不了这么多. 因此可以进行 encode.

考虑 classification:  $\mathbb{R}^{H \times W \times 3} \rightarrow \{0, 1\}^k$ . 这样我们就用一个词表示了原图. 同样, 对于 Auto-Encoder, 也是如此, 即我不相信这么多像素都是独立分布, 应该是  $H \times W \times 3$  空间的低维曲面. 同样对于 segmentation, 我们希望从原图到特征 map, 然后回到原图, 获得分类, 而不需要原图的 RGB 信息.

In-Network Unpooling:Unpooling.

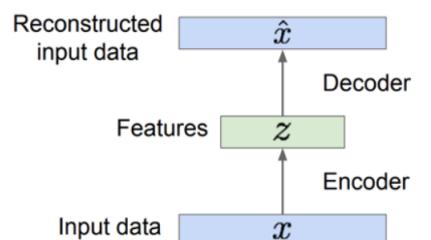


图 22: name of the figure

<sup>11</sup>其实没人会拿 intensity 做 kmeans. 因为 intensity 显然不能体现 pixel 的相关性.

第一种是 copy-paste, 需要用 conv 对 raw unsampling image 进行处理.

In-Network Unsampling:Max Unpooling: 记住哪个位置是池化中最大的. 也就是说, 并不是所有信息都在 bottle neck(中间小的部分) 处. 这也需要进行大量 conv 加工.

能不能在一步之内完成?Learnable unsampling:Transpose convolution. 学习出来一个 conv, 从  $2 \times 2 \rightarrow 4 \times 4$ .Fully Convolutional Network. 全都是 Conv 进行降采样和升采样.

每一次把 feature map 变小, 后面层的 conv 的 reception field 都会变大, 比不做 dimentional reduce 的图片感受野更大. 有助于提取 global context.bottleneck 的双重意义: 增大感受野, 减小 size.

关键: 增大感受野, 收缩 image 从而集中特征, 去除无用.

那么 FCN 有什么问题呢? 我们先来看看它的 bottle neck 存储了什么? 降低 resolution, 多个 channel. 现在在 neck 里面的内容本来就不多, 你还要求通过相同的升采样之后, 还能还原出 boundary, 这件事与 classification 相比, 不是一句话能说完的,neck 里恐怕很难同时存储下特征和原图的 boundary 信息.

那么我们让各个层次的 resolution 经过几层 conv<sup>12</sup> 直接连接到对称的升维操作, 照着把 bound 画出来就可以, 中间 neck 就只需要记 context 了. 这是一个非常经典的结构, 被称为 UNet, 也是如今 Segmentation 乃至 depth detection, dense prediction 都需要参考的结构, 甚至也是 Diffusion model<sup>13</sup>的一个重要的 Backbone.

Evaluation Metrics: Pixel Accuracy.

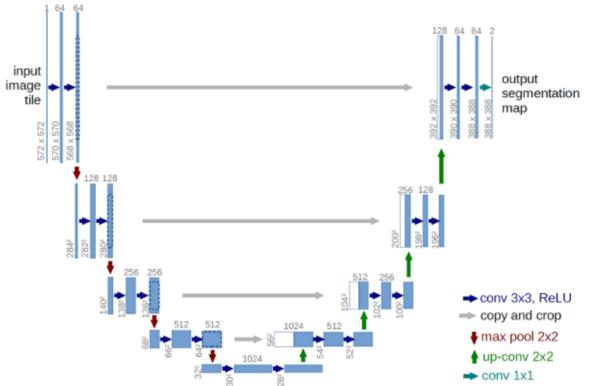


图 23: UNet 的结构示意图

$$\text{accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \quad (30)$$

当然这很有问题: 比如一个人头只占据整个图片, 那么将整个图片预测为背景, 那么错误率也很低, 但是这很显然不对.

另一个 loss:IoU->mean of every class IoU -> soft IoU.

<sup>12</sup>前几层可以理解为在寻找 corner 和 edge.

<sup>13</sup>但是非常有趣的是 U-net 最初是在生物学语义分割中提出的.2024 笔者注

## 10 3D Vision

Computer vision deals with: acquiring, processing, analyzing, and understanding, (might also include) generating or imagining visual data.

联系深度图,RGB 图和点云的概念—— camera.RGB 可以视作光打到深度图上, 对 RGB 进行采样得到。

如何设计 camera? 多个光源照在一点, 会产生 blurring. 因此产生了 Pinhole camera. 我们可以简单地描述其几何关系:

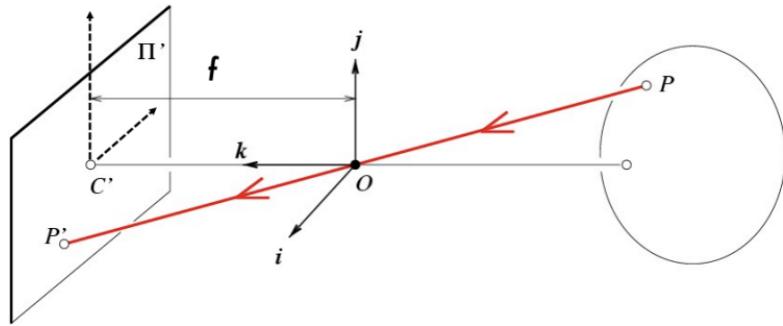


图 24: pinhole camera 的几何关系

满足如下变换

$$\begin{cases} x' = f \frac{x}{z} \\ y' = f \frac{y}{z} \end{cases} \quad (31)$$

aperture size 实际上控制模糊程度和亮度. 所以我们可以添加透镜 (lens), 透镜可能产生畸变, 因此真实相机往往复杂得多.

### 10.1 Intrinsics

上述变换的本质是一个映射  $E : \mathbb{R}^3 \rightarrow \mathbb{R}^2$ . 投影操作, projection mapping. 我们还需要完成两件事: 从实际的度量 (米) 转换到 pixel 为单位, 以及在图片中, 左下角为零点, 而非中心, 因此需要添加 offset.

$$P = (x, y, z) \rightarrow P' = \left( \alpha \frac{x}{z} + c_x, \beta \frac{y}{z} + c_y \right) \quad (32)$$

能否用矩阵表示? 很遗憾, 这不是线性变换. 因此引入 homogeneous coordinate system, 即增加一个维度作为除法.

$$P'_h = \begin{bmatrix} \alpha x + c_x z \\ \beta y + c_y z \\ z \end{bmatrix} = \begin{bmatrix} \alpha & 0 & c_x & 0 \\ 0 & \beta & c_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \quad (33)$$

如果考虑相机坐标系的 screw, 如下图, 则需要进行简单的变换:

$$\begin{cases} x = \alpha(\hat{x} - \cot\theta\hat{y}) + c_x \\ y = \beta\frac{\hat{y}}{\sin\theta} + c_y \end{cases} \quad (34)$$

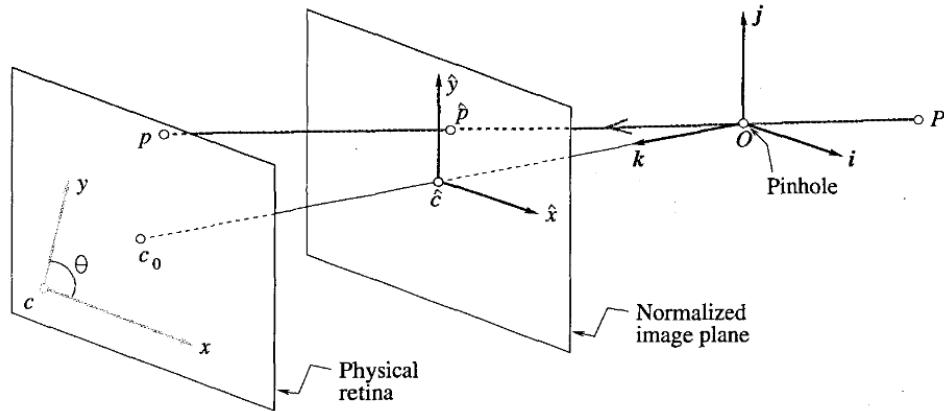


图 25: 成像过程坐标示意图

最后得到

$$P' = \begin{bmatrix} \alpha & -\alpha \cot\theta & c_x & 0 \\ 0 & \frac{\beta}{\sin\theta} & c_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \quad (35)$$

进行分解后表示为

$$P' = \mathbf{M}P = \mathbf{K} \begin{bmatrix} \mathbf{I} & \mathbf{0} \end{bmatrix} P \quad (36)$$

其中, 矩阵  $\mathbf{K}$  定义为

$$\mathbf{K} = \begin{bmatrix} \alpha & -\alpha \cot\theta & c_x \\ 0 & \frac{\beta}{\sin\theta} & c_y \\ 0 & 0 & 1 \end{bmatrix} \quad (37)$$

它是相机的内部参数, 拥有  $\alpha, \beta, \theta, c_x, c_y$  五个自由度.

## 10.2 Extinsics

上一节当中, 我们从 camera coordinate system->retina plane metric->image coordinate system. 但是我们希望完成从 world coordinate 到 image coordinate system 的转变. 不难看出, 这是两个空间直角坐标系的平移和旋转变换. 首先我们来看平移:

对于点  $P$ , 如果要将其平移向量  $\mathbf{T}$ , 则可以用如下的矩阵乘法表示:

$$P' \rightarrow \begin{bmatrix} \mathbf{I} & \mathbf{T} \\ \mathbf{0}^\top & 1 \end{bmatrix}_{4 \times 4} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \quad (38)$$

对于旋转, 我们先考虑平面直角坐标系的情形. 假如最开始的坐标系为  $S$ , 而  $S'$  是将  $S$  逆时针旋转  $\theta$ , 那么可以得出如果要将  $S$  中的向量逆时针旋转  $\theta$  就是将坐标乘以  $R_\theta$ , 其中

$$R_\theta = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix} \quad (39)$$

原理即是将所有基底旋转  $\theta$ , 坐标表示不变, 则自然就随基底旋转. 同样对于两个坐标系,  $Oijk$  和  $O'i'j'k'$ , 将前者坐标下的点  $(x, y, z)$  的基底向后者旋转, 则需要左乘

$$\mathbf{R} = \begin{bmatrix} i' \cdot i & j' \cdot i & k' \cdot i \\ i' \cdot j & j' \cdot j & k' \cdot j \\ i' \cdot k & j' \cdot k & k' \cdot k \end{bmatrix} \quad (40)$$

但是, 如果我们想在后者的坐标系中表示同一个向量 (注意这两者的区别), 那就需要左乘此矩阵的逆. 由于此矩阵正交, 所以也就是左乘其转置.

可以结合二维情形验证上述表达式. 除此之外, 我们还可以将旋转分解为三个方向的旋转:

$$R_x(\alpha) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \alpha & -\sin \alpha \\ 0 & \sin \alpha & \cos \alpha \end{bmatrix}, R_y(\beta) = \begin{bmatrix} \cos \beta & 0 & \sin \beta \\ 0 & 1 & 0 \\ -\sin \beta & 0 & \cos \beta \end{bmatrix}, R_z(\gamma) = \begin{bmatrix} \cos \gamma & -\sin \gamma & 0 \\ \sin \gamma & \cos \gamma & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (41)$$

$$P' \rightarrow \begin{bmatrix} \mathbf{R} & \mathbf{0} \\ \mathbf{0}^\top & 1 \end{bmatrix}_{4 \times 4} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \quad (42)$$

将平移和旋转结合, 我们就有了统一的形式:

$$P' = K \begin{bmatrix} I & 0 \end{bmatrix} P = K \begin{bmatrix} I & 0 \end{bmatrix} \begin{bmatrix} R & T \\ 0 & 1 \end{bmatrix}_{4 \times 4} P_w = K \begin{bmatrix} R & T \end{bmatrix} P_w \quad (43)$$

其过程示意图如下:

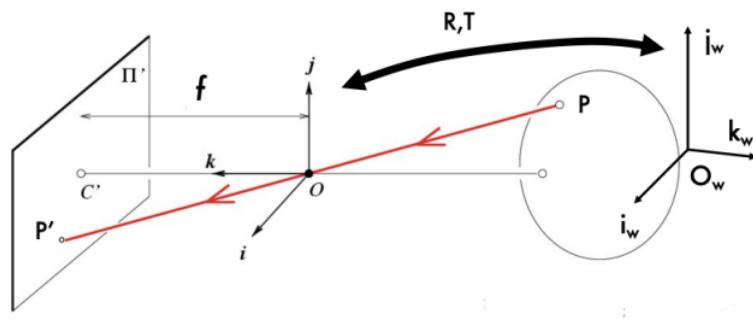


图 26: 坐标变换的全过程

若令  $M = \mathbf{K}[\mathbf{R} \ \mathbf{T}]$ , 设其三个行向量为  $\mathbf{m}_1, \mathbf{m}_2, \mathbf{m}_3$ , 则有

$$P' = \left( \frac{\mathbf{m}_1 P_w}{\mathbf{m}_3 P_w}, \frac{\mathbf{m}_2 P_w}{\mathbf{m}_3 P_w} \right) \quad (44)$$

投影变换的性质: 点映射成点, 线映射成线, 近大远小.

### 10.3 weak perspective

在弱透视模型中, 点首先用正交投影投影到参考平面, 然后用射影变换投影到图像平面. 如下图:

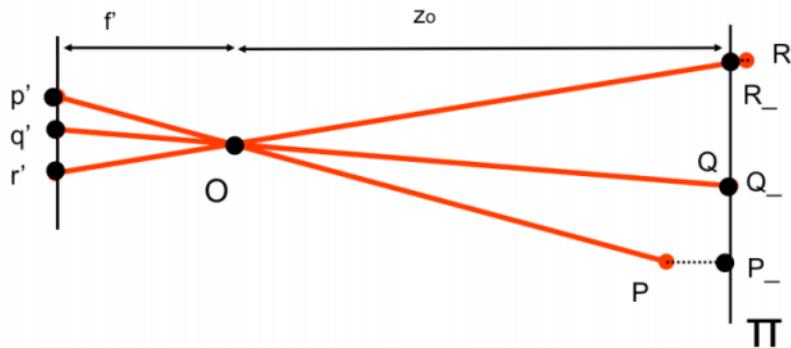


图 27: weak perspective

这里的正交投影可以表示为

$$\mathbf{O} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & z_0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \stackrel{\text{def}}{=} \begin{bmatrix} \mathbf{O}' & z_0 \\ \mathbf{0}^\top & 1 \end{bmatrix} \quad (45)$$

即将所有  $z$  分量变为  $z_0$ . 随后根据我们的变换公式

$$P' = \mathbf{K} \begin{bmatrix} \mathbf{I} & \mathbf{0} \end{bmatrix} \mathbf{O} \begin{bmatrix} \mathbf{R} & \mathbf{T} \\ \mathbf{0}^\top & 1 \end{bmatrix} P = \mathbf{K}_{3 \times 3} \begin{bmatrix} \mathbf{O}' \mathbf{R} & \mathbf{O}' \mathbf{T} + z_0 \end{bmatrix}_{3 \times 4} P_{4 \times 1} \quad (46)$$

进一步, 由于  $\mathbf{O}'$  的第三行全零, 因此中间矩阵的第三行是  $[0, 0, 0, z_0]$ . 记  $\mathbf{R}_2, \mathbf{t}_2$  分别是  $\mathbf{R}, \mathbf{T}$  的前两行, 上式可以改写为

$$P' = \mathbf{K} \begin{bmatrix} \mathbf{R}_2 & \mathbf{t}_2 \\ \mathbf{0}^\top & z_0 \end{bmatrix} P \quad (47)$$

我们记

$$\mathbf{K} = \begin{bmatrix} \mathbf{K}_2 & \mathbf{p}_0 \\ \mathbf{0}^\top & 1 \end{bmatrix}, \quad \mathbf{K}_2 = \begin{bmatrix} \alpha & -\alpha \cot \theta \\ 0 & \frac{\beta}{\sin \theta} \end{bmatrix}, \quad \mathbf{p}_0 = \begin{bmatrix} c_x \\ c_y \end{bmatrix} \quad (48)$$

随后得到

$$\mathbf{P}' = \begin{bmatrix} \mathbf{K}_2 \mathbf{R}_2 & \mathbf{K}_2 \mathbf{t}_2 + z_0 \mathbf{p}_0 \\ \mathbf{0}^\top & z_0 \end{bmatrix} \begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix} \quad (49)$$

不难看出,  $\mathbf{P}'$  的第三个坐标 (即齐次项) 为常数  $z_0$ , 可以直接写成更简单的二维坐标形式:

$$\mathbf{p} = \mathbf{M}\mathbf{P} \begin{bmatrix} \mathbf{A} & \mathbf{b} \end{bmatrix} \mathbf{P} \quad (50)$$

其中

$$\mathbf{A} = \frac{1}{z_0} \mathbf{K}_2 \mathbf{R}_2, \quad \mathbf{b} = \frac{1}{z_0} \mathbf{K}_2 \mathbf{t}_2 + \mathbf{p}_0 \quad (51)$$

更简单:orthographic (Affine) projection. 正交投影. 没有近大远小. 这种投影在你不希望有近大远小的时候可以用到.

## 11 Camera Calibration

我们前面已经知道, 从  $P_w \rightarrow P'$  的坐标变换是

$$P' = \mathbf{M}P_w = \mathbf{K} \begin{bmatrix} \mathbf{R} & \mathbf{T} \end{bmatrix} P_w \quad (52)$$

其中

$$\mathbf{M} = \begin{pmatrix} \alpha \mathbf{r}_1^T - \alpha \cot \theta \mathbf{r}_2^T + u_0 \mathbf{r}_3^T & \alpha t_x - \alpha \cot \theta t_y + u_0 t_z \\ \frac{\beta}{\sin \theta} \mathbf{r}_2^T + v_0 \mathbf{r}_3^T & \frac{\beta}{\sin \theta} t_y + v_0 t_z \\ \mathbf{r}_3^T & t_z \end{pmatrix} \quad (53)$$

那么什么是 calibration problem 呢? 如果我们已知世界坐标  $P_1, \dots, P_n$  (形式为  $[O_w, i_w, j_w, k_w]$ ) 和对应的图像坐标  $p_1, \dots, p_n$ . 我们的目标是通过这些已知数据, 获得 intrinsic and extrinsic parameters. 这个问题的意义, 比如我们希望从图像运动获取实际运动.

问题有 11 个自由度:  $5(\text{in}) + 3(\text{ex-r}) + 3(\text{ex-t}) = 11$ . 需要 11 个方程, 6 个点.

对每个  $p_i(u_i, v_i)$ , 我们有

$$\begin{aligned} u_i &= \frac{\mathbf{m}_1 P_i}{\mathbf{m}_3 P_i} \rightarrow u_i (\mathbf{m}_3 P_i) = \mathbf{m}_1 P_i \rightarrow u_i (\mathbf{m}_3 P_i) - \mathbf{m}_1 P_i = 0 \\ v_i &= \frac{\mathbf{m}_2 P_i}{\mathbf{m}_3 P_i} \rightarrow v_i (\mathbf{m}_3 P_i) = \mathbf{m}_2 P_i \rightarrow v_i (\mathbf{m}_3 P_i) - \mathbf{m}_2 P_i = 0 \end{aligned} \quad (54)$$

这样我们可以列出方程组:

$$\left\{ \begin{array}{l} u_1 (\mathbf{m}_3 P_1) - \mathbf{m}_1 P_1 = 0 \\ v_1 (\mathbf{m}_3 P_1) - \mathbf{m}_2 P_1 = 0 \\ \vdots \\ u_i (\mathbf{m}_3 P_i) - \mathbf{m}_1 P_i = 0 \\ v_i (\mathbf{m}_3 P_i) - \mathbf{m}_2 P_i = 0 \\ \vdots \\ u_n (\mathbf{m}_3 P_n) - \mathbf{m}_1 P_n = 0 \\ v_n (\mathbf{m}_3 P_n) - \mathbf{m}_2 P_n = 0 \end{array} \right. \quad (55)$$

将  $m$  展开, 获得方程组:

$$\mathbf{P}m = \mathbf{0} \quad (56)$$

其中

$$\mathbf{P} \stackrel{\text{def}}{=} \begin{pmatrix} \mathbf{P}_1^T & \mathbf{0}^T & -u_1 \mathbf{P}_1^T \\ \mathbf{0}^T & \mathbf{P}_1^T & -v_1 \mathbf{P}_1^T \\ \vdots & \vdots & \vdots \\ \mathbf{P}_n^T & \mathbf{0}^T & -u_n \mathbf{P}_n^T \\ \mathbf{0}^T & \mathbf{P}_n^T & -v_n \mathbf{P}_n^T \end{pmatrix} \quad (57)$$

以及

$$m = \begin{pmatrix} \mathbf{m}_1^T \\ \mathbf{m}_2^T \\ \mathbf{m}_3^T \end{pmatrix} \quad (58)$$

但是这个问题过定, 且有平凡解. 我们先对  $\mathbf{m}$  添加一个 constrain:  $\|\mathbf{m}\| = 1$ . 随后用 SVD 求解如下优化问题:

$$\begin{aligned} & \text{minimize} \quad \|\mathbf{P}\mathbf{m}\|^2 \\ & \text{s.t.} \quad \|\mathbf{m}\| = 1 \end{aligned} \tag{59}$$

进行 SVD 获得最小的特征向量, 即为解.

此时还没有结束, 我们需要定出收缩因子. 假定

$$\mathcal{M} = \begin{pmatrix} \alpha \mathbf{r}_1^T - \alpha \cot \theta \mathbf{r}_2^T + u_0 \mathbf{r}_3^T & \alpha t_x - \alpha \cot \theta t_y + u_0 t_z \\ \frac{\beta}{\sin \theta} \mathbf{r}_2^T + v_0 \mathbf{r}_3^T & \frac{\beta}{\sin \theta} t_y + v_0 t_z \\ \mathbf{r}_3^T & t_z \end{pmatrix} \rho \tag{60}$$

此时我们发现  $M$  第三行可以直接定出  $\rho$ . 于是这样我们就可以获得相应参数.

实际上, 对这个问题的求解就是进行 RQ 分解<sup>14</sup>. 我们知道 RQ 分解就是将一个矩阵  $\mathbf{A}$  分解为:

$$\mathbf{A} = \mathbf{R}\mathbf{Q} \tag{61}$$

其中  $\mathbf{Q}, \mathbf{R}$  分别是正交矩阵和上三角矩阵. 当然, 得到的正交矩阵可能是瑕旋转矩阵, 即行列式为  $-1$  的正交矩阵. 这种情况我们要检查分解得到的上三角矩阵是否满足相机内参的要求, 比如  $\sin \theta > 0$ , 此时将此列和正交矩阵对应行乘以  $-1$  即可.

当然并不是所有的 6 个点都可以. 不能在同一平面 (15:50). 对于一般的有畸变的相机, 有更复杂的非线性处理方式, 可能没有解析解.

### 11.1 Some Problems with Camera

**问题:** 只知道内参或深度信息可以唯一确定一个物体吗?

只有内参或者深度信息, 都不能确定一个物体, 必须两个都知道才能得到物体真实数据

只有内参: 外参和物体大小同时变化

只有深度信息: 物体在平行于相机的那一个平面上的大小可以变化, 因为这个平面上的物体是同一个深度

**问题:** 假设我们在知道一个照片以外, 还知道每个像素的深度, 那么可以找出真实世界中两点的距离吗?

不可以. 因为不知道相机内参,  $u$  和  $v$  代表着像素差, 无法确定相机参考系下的  $\Delta x, \Delta y$  也就是真实距离差.

**相机相关计算**

1. Depth back projection:

$$(K, u, v, z) \rightarrow (x, y)$$

使用  $K$  的定义

2. Camera calibration:

---

<sup>14</sup>关于 QR 分解, 请参见附录 D

$$(x, y, z, u, v) \rightarrow K$$

如果不知道  $K$ . 那么就是相机标定

问题: 为什么相机标定的时候所有参考点不能在同一个平面上?

如果所有参考点都在同一个平面上, 那么相机的观测将缺乏深度信息, 因为所有的标定点都位于一个二维平面内. 这会导致所谓的“退化配置”(degenerate configuration), 在这种配置下, 我们不能唯一地确定相机的内外参数, 尤其是关于深度和空间位置的信息.

例如: 我们无法区分相机距离标定平面远但焦距短, 与相机距离标定平面近但焦距长的情况. 这两种情况在所有参考点都在同一平面上时可能会产生相似的投影图像.

问题: 根据 depth back projection 计算出来的相机坐标系下的  $\Delta x$  和  $\Delta y$  是不是 world coordinate 下的距离?

是的。

因为旋转和平移是保角保距离变换。

## 12 Single View Geometry

这一章节在教学中已经被删去, 为了让有兴趣的读者了解, 故保留

上面我们解释了从 3D 转换到 2D 的过程, 那么反过来, 能否从单视角还原成 3D 呢? 很显然, 这并不是一个有唯一解的问题. 无论是从维度的角度, 还是实际操作角度都是如此. 成像的时候, 一个 pixel 代表着一条射线, 而其深度无法确定.

尽管如此, 我们还是能得到一些有意思结论. 比如我们知道点会映射成点, 而线映射成线. 以及右图中, 似乎铁轨和草皮交于图像上方的一点.



图 28: 一张风景图

### 12.1 Transformation in $\mathbb{R}^2$

我们先来看看二维平面中的线. 对于  $ax + by + c = 0$ , 用齐次坐标表达为  $\mathbf{l} = [a, b, c]^\top$ , 则点在线上等价于  $\mathbf{x}^\top \mathbf{l} = 0$ . 这里  $\mathbf{x} = [x, y, 1]^\top$  为齐次坐标. 两条线  $\mathbf{l}, \mathbf{l}'$  的交点坐标是  $\mathbf{x} = \mathbf{l} \times \mathbf{l}'$ .

我们还能发现, 这样的表示还能适用于平行线的交点. 不难看出两条平行线的叉乘

$$\begin{vmatrix} i & j & k \\ a & b & c \\ a' & b' & c' \end{vmatrix} \quad (62)$$

的齐次坐标  $ab' - a'b = 0 \cdot x_\infty = [b, -a, 0]^\top$ . 不难验算得到所有与这条直线平行的点都交于  $x_\infty$ .

我们还可以定义  $\mathbf{l}_\infty = [0, 0, 1]^\top$ . 这条线是所有无穷点的共线, 即无穷远处的直线. 这是在欧式空间无法表达的一条直线. 其齐次坐标当然也可以是任何常数.

我们尝试将射影变换<sup>15</sup>作用于某个无穷远点, 得到如下结果:

$$\mathbf{p}' = \mathbf{H}\mathbf{p}_\infty = \begin{bmatrix} \mathbf{A} & \mathbf{t} \\ \mathbf{v} & b \end{bmatrix} \begin{bmatrix} a \\ b \\ 0 \end{bmatrix} = \begin{bmatrix} p'_x \\ p'_y \\ p'_z \end{bmatrix} \quad (63)$$

也就是说, 无穷远点可能会被映射到某个有限远的点. 但是对于仿射变换, 就不存在这种问题了:

$$\mathbf{p}' = \mathbf{H}\mathbf{p}_\infty = \begin{bmatrix} \mathbf{A} & \mathbf{t} \\ \mathbf{0}^\top & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ 0 \end{bmatrix} = \begin{bmatrix} p'_x \\ p'_y \\ 0 \end{bmatrix} \quad (64)$$

对于变换后的直线, 根据射影变换的性质, 如果将  $\ell$  变换为  $\ell'$ , 那么如果  $\mathbf{x} \in \ell$ , 则  $\mathbf{x}' = \mathbf{H}\mathbf{x} \in \ell'$ . 设对直线的变换是  $\mathbf{T}$ , 那么有

$$\mathbf{x}'^\top \mathbf{T}\ell = \mathbf{x}^\top \mathbf{H}^\top \mathbf{H}^{-\top} \ell = 0 \quad (65)$$

得出  $\mathbf{T} = \mathbf{H}^{-\top}$ . 同样我们可以发现射影变换可能会将无穷远处的直线映射到有限远, 而仿射变换不会.

<sup>15</sup>附录B当中简单介绍了几种变换的形式和性质.

## 12.2 Vanishing Points

有了上面的铺垫, 我们来看看 vanishing point 的概念. 我们可自然地将上述概念推广到三维的情形, 那么对三维空间中一系列平行线的无穷远点  $x_\infty$ , 经过相机变换  $\mathbf{M}$ , 就会成为二维中的有限远的一个点  $p_\infty$ , 也就是 vanishing point, 如下图:

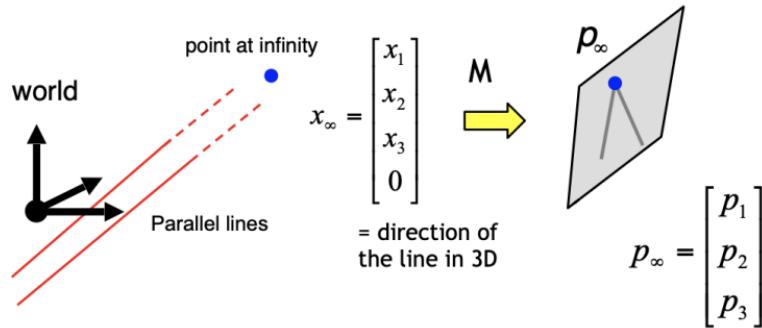


图 29: Vanishing points 在 world coordinate 中的示意图

上图的坐标是 world coordinate 当中. 如果是在 camera coordinate 当中, 设  $\mathbf{d} = (a, b, c)$  是直线的方向向量,  $\mathbf{v}$  代表消失点, 我们还可以导出如下关系<sup>16</sup>:

$$\mathbf{v} = \mathbf{K}\mathbf{d} \quad (66)$$

这一点不难证明: 由于是在相机坐标之下, 无需考虑 extrinsic 参数, 直接有

$$\mathbf{v} = \mathbf{M}\mathbf{x}_\infty = \mathbf{K} \begin{bmatrix} \mathbf{I} & \mathbf{0} \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ 0 \end{bmatrix} = \mathbf{K}\mathbf{d} \quad (67)$$

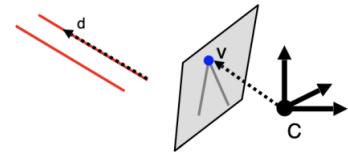


图 30: 相机坐标下的方向

这样, 如果已知 vanishing point 的坐标和相机参数, 我们可以知道相机坐标下的这一组平行线的方向:

$$\mathbf{d} = \frac{\mathbf{K}^{-1}\mathbf{v}}{\|\mathbf{K}^{-1}\mathbf{v}\|} \quad (68)$$

如果我们继续推广在二维空间中的结论, 那么一系列平行平面将会相交于无穷远处的一条线  $l_\infty$ . 将这条线也作同样的变换, 就可以得到 Vanishing Line, 即 Horizon, 译为天际线, 也就是 3 维空间  $l_\infty$  在二维的投影:

$$l_{\text{hor}} = \mathbf{H}^{-\top} l_\infty \quad (69)$$

通过这种方式, 我们可以确定两条线是否平行. 首先找出天际线, 如果照片上的两条线交于天际线的同一点则平行, 反之不平行.

我们还可以通过两个 vanishing point 确定其对应的平行线之间的夹角.

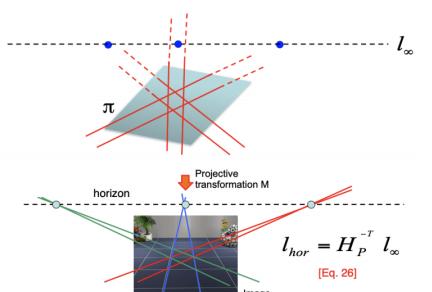


图 31: 天际线

<sup>16</sup>注意在右图里,  $\mathbf{v}$  并不是从相机坐标中心出发的三维矢量, 而是图片坐标下的齐次坐标.

如右图, 根据  $\mathbf{d} = \mathbf{K}^{-1}\mathbf{v}$ , 我们可以求出其夹角:

$$\cos \theta = \frac{\mathbf{d}_1 \cdot \mathbf{d}_2}{\|\mathbf{d}_1\| \|\mathbf{d}_2\|} \quad (70)$$

令  $\Omega = (\mathbf{K}\mathbf{K}^T)^{-1}$ , 得到

$$\cos \theta = \frac{\mathbf{v}_1^\top \Omega \mathbf{v}_2}{\sqrt{\mathbf{v}_1^\top \Omega \mathbf{v}_1} \sqrt{\mathbf{v}_2^\top \Omega \mathbf{v}_2}} \quad (71)$$

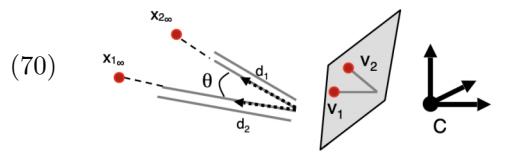


图 32: Angle from two vanishing points.

当  $\theta = \frac{\pi}{2}$  的时候有  $\mathbf{v}_1^\top \Omega \mathbf{v}_2 = 0$ .  $\Omega$  有五个独立变量. 如果我们已知空间中的三组互相垂直的平行线和它们在图片上的 vanishing point, 则我们可以获得三个方程

$$\begin{cases} \mathbf{v}_1^\top \Omega \mathbf{v}_2 = 0 \\ \mathbf{v}_1^\top \Omega \mathbf{v}_3 = 0 \\ \mathbf{v}_2^\top \Omega \mathbf{v}_3 = 0 \end{cases} \quad (72)$$

如果我们假定相机具有零偏置和方形像素, 那么不难得得到

$$\mathbf{K} = \begin{bmatrix} \alpha & 0 & c_x \\ 0 & \alpha & c_y \\ 0 & 0 & 1 \end{bmatrix}, \quad \mathbf{K}\mathbf{K}^\top = \begin{bmatrix} \alpha^2 + c_x^2 & c_x c_y & c_x \\ c_x c_y & \alpha^2 + c_y^2 & c_y \\ c_x & c_y & 1 \end{bmatrix} \quad (73)$$

$$\Omega = (\mathbf{K}\mathbf{K}^\top)^{-1} = \frac{1}{\alpha^4} \begin{bmatrix} \alpha^2 & 0 & -\alpha^2 c_x \\ 0 & \alpha^2 & -\alpha^2 c_y \\ -\alpha^2 c_x & -\alpha^2 c_y & (\alpha^2 + c_x^2)(\alpha^2 + c_y^2) - c_x^2 c_y^2 \end{bmatrix} \quad (74)$$

$\Omega$  的形式变为:

$$\Omega = \begin{bmatrix} \omega_1 & 0 & \omega_4 \\ 0 & \omega_1 & \omega_5 \\ \omega_4 & \omega_5 & \omega_6 \end{bmatrix} \quad (75)$$

四个变量, 三个线性方程, 在常数倍意义下我们可以得到  $\Omega$ , 随后运用 Cholesky 分解得到  $\mathbf{K}$ . 之后我们就可以对场景进行三维重建, 例如计算图片中所有平面的方向, 因此一张图片实际上蕴含了周围场景的丰富信息.

在这一节的最后, 我们仍然要指出, 单纯从一张图片是不可能完整重建原 3D 场景的, 这是因为物体的深度和大小之间存在 ambiguity. 在下一节当中, 我们会试图解决这个问题.

## 13 Epipolar Geometry

这一章节在教学中已经被删去, 为了让有兴趣的读者了解, 故保留

既然单个视角无法确定深度, 那么我们自然会想到如果有多台相机或许就可以确定距离, 如同人眼一样.

原则上, 两台相机就可以确定点的位置. 我们看右图: $O_1, O_2$  代表两台相机的镜心, 两个平行四边形代表各自的像平面, 如果一个点  $P$  在两个像平面的投影点  $p, p'$  已知, 我们就可以确定  $P$  点在镜心到对应点的连线交点的位置.

在前面的章节当中我们已经介绍了校准相机的方法, 而两台相机的位置参数  $\mathbf{R}, \mathbf{t}$  一般都是已知的, 此时的关键问题就是: 确定一个点在另一个相机图片的位置.

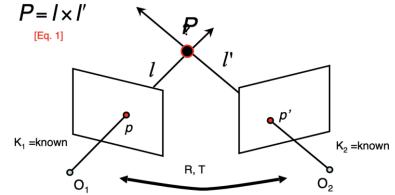
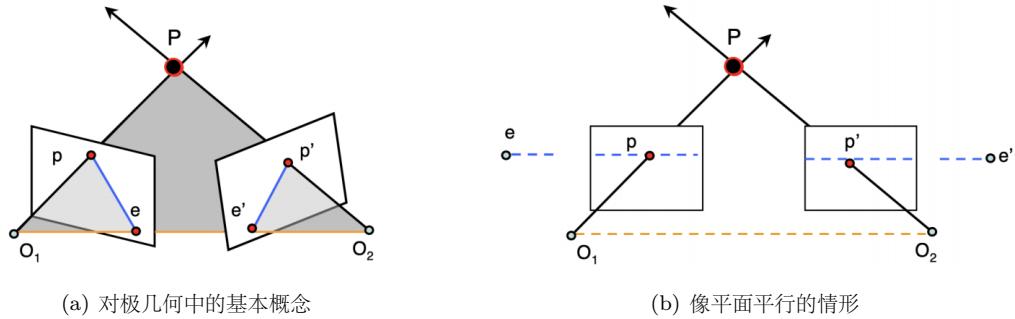


图 33: Triangulation



与这一任务相关的几何内容被称为对极几何 (epipolar geometry). 我们先给出对极几何当中的  
一些概念.

左图中由  $O_1, O_2, P$  三点构成的平面 (图中灰色填充) 被称为 epipolar plane. 连线  $O_1O_2$  被称  
为 baseline. baseline 与两个像平面的交点被称为 epipole. 各个像平面内,epipole 与  $P$  的投影点被称  
为 epipolar line.

当  $P$  点移动时, 相当于极平面上下翻动, 此时对应的极线也会变化, 但是所有的极线都通过极  
点.

一种特殊情况是两个像平面相互平行, 此时极点位于无穷远处, 图中所有极线都平行于基线.  
那么, 给定图片中的一个点, 要如何去确定在另一张图片中这个点的对应呢? 我们看下图, 根据给定  
点和基线我们可以确定极平面, 极平面与右像平面的交点即为极线, 对应点  $p'$  必定在此极线上, 也  
就将对应点的搜索从 2 维降低到了 1 维.

### 13.1 Epipolar Constraint

首先我们先确定两台相机的位置. 如下图, 方便起见, 假设世界坐标与第一台相机的坐标重合,  
第二台相机的坐标系由旋转  $\mathbf{R}$  和偏置  $\mathbf{t}$  确定. 对于一点  $P$ , 如果它在两台相机的坐标系下的坐标分  
别是  $\mathbf{p}_E, \mathbf{q}_E$ , 则有关系

$$\mathbf{q}_E = \mathbf{R}^\top (\mathbf{p}_E - \mathbf{t}) \quad (76)$$

因此, 两台相机的变换矩阵分别是

$$\mathbf{M} = \mathbf{K} \begin{bmatrix} \mathbf{I} & \mathbf{0} \end{bmatrix}, \quad \mathbf{M}' = \mathbf{K}' \begin{bmatrix} \mathbf{R}^\top & -\mathbf{R}^\top \mathbf{t} \end{bmatrix} \quad (77)$$

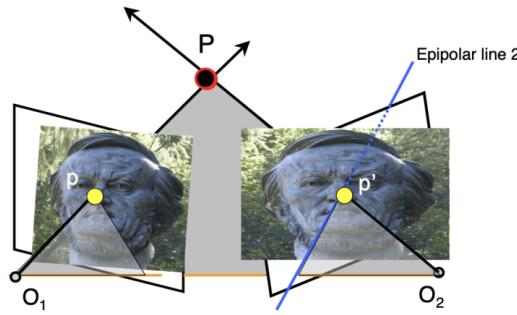


图 34: 两张图片的对应

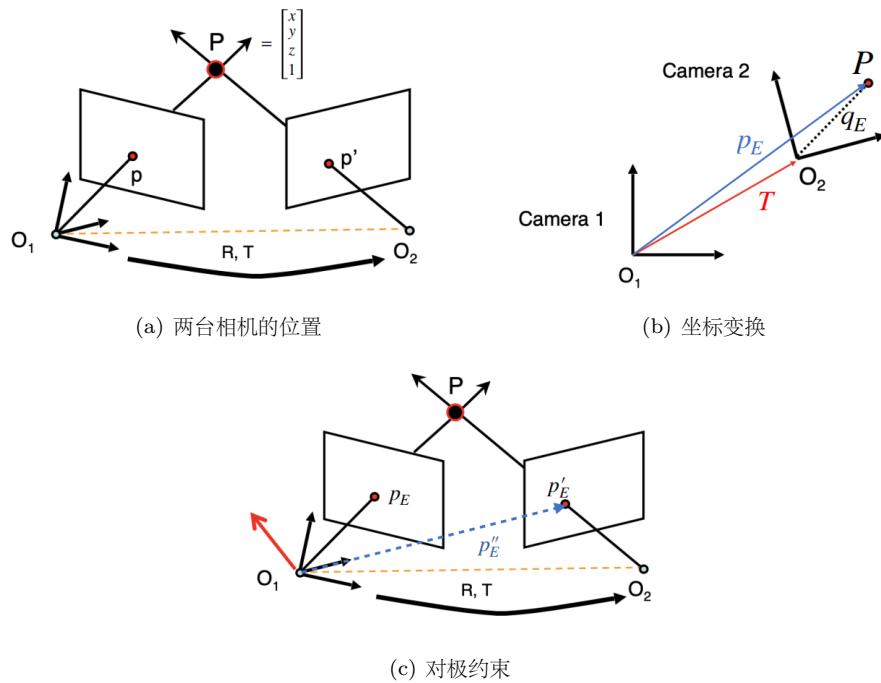


图 35:

如图 35(c), 我们规定以下符号:  $\mathbf{p}_E, \mathbf{p}'_E, \mathbf{p}''_E \in \mathbb{R}^3$  分别代表左投影点在左相机坐标系 (即世界坐标系) 的坐标, 右投影点在右相机坐标系下的坐标, 右投影点在左相机坐标系下的坐标. 我们可以求出极平面的法向量 (图中红色箭头):

$$\mathbf{n} = \mathbf{t} \times \mathbf{p}''_E = \mathbf{t} \times (\mathbf{R}\mathbf{p}'_E + \mathbf{t}) = \mathbf{t} \times \mathbf{R}\mathbf{p}'_E \quad (78)$$

对于两个向量  $\mathbf{a} = [a_x, a_y, a_z]^\top, \mathbf{b} = [b_x, b_y, b_z]^\top$  的叉乘, 可以表示成矩阵形式:

$$\mathbf{a} \times \mathbf{b} = [\mathbf{a}] \mathbf{b} = \begin{bmatrix} 0 & -a_z & a_y \\ a_z & 0 & -a_x \\ -a_y & a_x & 0 \end{bmatrix} \begin{bmatrix} b_x \\ b_y \\ b_z \end{bmatrix} \quad (79)$$

不难看出这是一个反对称矩阵, 秩为 2.

显然有  $\mathbf{p}_E \perp \mathbf{n}$ , 写成矩阵形式就是

$$\mathbf{p}_E^\top [\mathbf{t}] \mathbf{R} \mathbf{p}'_E = 0 \quad (80)$$

我们定义

$$\mathbf{E} = [\mathbf{t}_\times] \mathbf{R} \quad (81)$$

为本质矩阵 (essential matrix). 它是一个形状为  $3 \times 3$ , 秩为 2 的矩阵, 有五个自由度.<sup>17</sup>

代入式 80 得到:

$$\mathbf{p}_E^\top \mathbf{E} \mathbf{p}'_E = 0 \quad (82)$$

虽然有了本质矩阵是在两个相机坐标系下的三维坐标的变换, 我们自然希望能用二维的图片坐标进行直接对应. 考虑到

$$\mathbf{p}_E = \mathbf{K}^{-1} \mathbf{p}, \quad \mathbf{p}'_E = \mathbf{K}'^{-1} \mathbf{p}' \quad (83)$$

代入式 80 即得

$$\mathbf{p}^\top \mathbf{K}^{-\top} [\mathbf{t}_\times] \mathbf{R} \mathbf{K}'^{-1} \mathbf{p}' = 0 \quad (84)$$

我们定义

$$\mathbf{F} = \mathbf{K}^{-\top} \mathbf{E} \mathbf{K}'^{-1} = \mathbf{K}^{-\top} [\mathbf{t}_\times] \mathbf{R} \mathbf{K}'^{-1} \quad (85)$$

为基本矩阵 (fundamental matrix), 它是一个秩为 2, 自由度为 7 的  $3 \times 3$  矩阵. 式 85 变为

$$\mathbf{p}^\top \mathbf{F} \mathbf{p}' = 0 \quad (86)$$

除此之外, 基本矩阵还有如下性质: 如图 36, 首先,  $\mathbf{l} = \mathbf{F} \mathbf{p}'$  和  $\mathbf{l}' = \mathbf{F}^\top \mathbf{p}$  分别是  $\mathbf{p}, \mathbf{p}'$  对应的极线. 其次,  $\mathbf{F} \mathbf{e}' = 0, \mathbf{F}^\top \mathbf{e} = 0$ .

先证明前者. 式 86 说明  $\mathbf{p} \in \mathbf{l}$ . 设  $\mathbf{e}$  对应的世界坐标为  $\mathbf{e}_E$ , 则由于  $\mathbf{e}_E \perp \mathbf{n}$ , 因此式 80 及以后的各式将  $\mathbf{p}_E$  替换成  $\mathbf{e}_E$  都成立, 那么同式 86 的形式, 我们亦可以得到

$$\mathbf{e}^\top \mathbf{F} \mathbf{p}' = 0. \quad (87)$$

由此得出  $\mathbf{e} \in \mathbf{l}$ .<sup>18</sup>

对于后者, 我们知道  $\forall \mathbf{p}$ , 都有式 87 成立, 因此只能有

$$\mathbf{F} \mathbf{e}' = 0, \quad \mathbf{F}^\top \mathbf{e} = 0. \quad (88)$$

我们对本质矩阵和基本矩阵做一个总结.

**E** 是两个标准相机相对关系的内参, 换言之, 给定两个已标定相机的相对位置关系, **E** 即被确定.

**F** 是任意两个相机相对关系的内参, 换言之, 给定了两个相机的相对位置关系和相机内参, **F** 即被确定.

如果我们能够求出 **F**, 那么就可以在不进行三维重建的情况下, 确定两张照片的对应关系. 还记得我们在上节末尾和这节的开头提出的问题吗? 在上一节的末尾, 我们说通过两台相机即可确定深度信息, 在这一节的开头我们引入了对极几何的概念, 我们的目的是确定两张照片中的对应点, 再

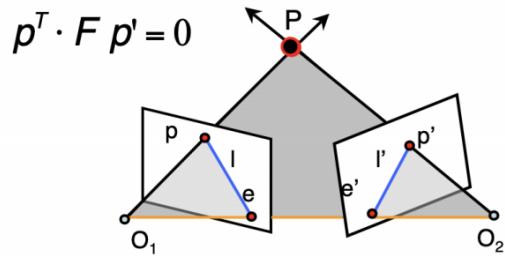


图 36: 基本矩阵的性质

<sup>17</sup> 在下文的基本矩阵的讨论中, 我们也会涉及到自由度的问题, 这一度令笔者陷入困惑. 这些内容将在附录 C 当中作进一步探讨.

<sup>18</sup> 细心的读者不难发现, 极平面内所有的点都可以这样做, 而这个结论也是非常自然的, 因为极线正是极平面射影在左像平面的结果.

通过两台相机的方向即可求出点的深度, 而方向已经在上一节中解决. 现在我们离解决这个问题只剩最后一步, 即如何求出  $\mathbf{F}$ ?

我们设  $\mathbf{F} = (F_{ij})$ , 根据式 86, 如果有 8 对对应点<sup>19</sup>的坐标  $(u_i, v_i, 1)^\top \leftrightarrow (u'_i, v'_i, 1)^\top$ , 那么我们可以得到方程组

$$\begin{pmatrix} u_1 u'_1 & u_1 v'_1 & u_1 & v_1 u'_1 & v_1 v'_1 & v_1 & u'_1 & v'_1 & 1 \\ u_2 u'_2 & u_2 v'_2 & u_2 & v_2 u'_2 & v_2 v'_2 & v_2 & u'_2 & v'_2 & 1 \\ u_3 u'_3 & u_3 v'_3 & u_3 & v_3 u'_3 & v_3 v'_3 & v_3 & u'_3 & v'_3 & 1 \\ u_4 u'_4 & u_4 v'_4 & u_4 & v_4 u'_4 & v_4 v'_4 & v_4 & u'_4 & v'_4 & 1 \\ u_5 u'_5 & u_5 v'_5 & u_5 & v_5 u'_5 & v_5 v'_5 & v_5 & u'_5 & v'_5 & 1 \\ u_6 u'_6 & u_6 v'_6 & u_6 & v_6 u'_6 & v_6 v'_6 & v_6 & u'_6 & v'_6 & 1 \\ u_7 u'_7 & u_7 v'_7 & u_7 & v_7 u'_7 & v_7 v'_7 & v_7 & u'_7 & v'_7 & 1 \\ u_8 u'_8 & u_8 v'_8 & u_8 & v_8 u'_8 & v_8 v'_8 & v_8 & u'_8 & v'_8 & 1 \end{pmatrix} \begin{pmatrix} F_{11} \\ F_{12} \\ F_{13} \\ F_{21} \\ F_{22} \\ F_{23} \\ F_{31} \\ F_{32} \\ F_{33} \end{pmatrix} = \mathbf{0} \quad (89)$$

我们记上式为

$$\mathbf{W}\mathbf{f} = \mathbf{0} \quad (90)$$

如果  $\text{rank}(\mathbf{W}) = 8$ , 则  $\mathbf{f}$  有唯一非零解. 当对应点对数目多于 8 个, 则限制  $\|\mathbf{f}\| = 1$ , 使用 SVD 求解. 但是  $\mathbf{F}$  的秩为 2, 那么问题转化为

$$\begin{aligned} & \text{minimize } \|\mathbf{F} - \hat{\mathbf{F}}\| \\ & \text{s.t. } \det \mathbf{F} = 0. \end{aligned} \quad (91)$$

而 SVD 告诉我们, 设  $\text{rank}(\mathbf{F}) = r$ , 若将  $\mathbf{F}$  按奇异值降序分解为

$$\mathbf{F} = \sum_{i=1}^r \sigma_i \mathbf{u}_i \mathbf{v}_i^\top \quad (92)$$

则在所有秩不大于  $k$  的矩阵当中,

$$\mathbf{F} = \sum_{i=1}^k \sigma_i \mathbf{u}_i \mathbf{v}_i^\top \quad (93)$$

就是使得  $\|\mathbf{F} - \hat{\mathbf{F}}\|$  最小的矩阵. 因此我们需要对式 89 求解出的  $\mathbf{F}$  再次进行 SVD, 并取前两个奇异值和左右奇异向量构成解  $\mathbf{F}$ .

<sup>19</sup>这里读者可能会产生一个问题, 既然  $\mathbf{F}$  的自由度为 7, 为什么要 8 对对应点呢? 其中原因之一是, 相机的内部性质多为非线性, 所以使用最小点数求解可能会比较麻烦, 因此经常只考虑尺度的等价性, 忽略奇异性条件, 此时自由度为 8.

## 14 3D data

from sensor or graphics.

第三种 sensor:time-of-light. 即用光的传播时间计算深度. 两种:iToF and dToF.direct or indirect. 前者使用脉冲波, 后者计算光的相位差.

dToF 精度高, 但对器件要求高, 需要使用 SPAD(单光子雪崩二极管), 无法做得很密集, 造价高.iToF 精度低 (误差与距离成正比), 但造价更低.

multiple 3D representation:

multiview images. 从各个视角获得的多张图片. 它包含 3D 信息,indirect,not a true 3D representation.

depth image. 只知道深度, 不知道相机内参, 无法确定两个点的距离. 因此被称为 2.5D.

Voxels. 物体占据了位置则设为 1. 能被 index. 但是非常昂贵. $O(n^3)$ . 没有表面的表示. 一旦分辨率低, 则无法还原丢失的信息.

Irregular 3D representation. Mesh,point Cloud, Implicit representation.

### 14.1 Mesh

在表面取点, 用过三点的平面近似表面. 实际上也不限于三角形. 下面我们讨论 triangle mesh.

$$\begin{aligned} V &= \{v_1, \dots, v_n\} \in \mathbb{R}^3 \\ E &= \{e_1, \dots, e_n\} \in V \times V \\ F &= \{f_1, \dots, f_n\} \in V \times V \times V \end{aligned} \tag{94}$$

好的 mesh:watertight(不漏), manifold(外法向量连续).

### 14.2 Point Cloud

Point cloud 是一个  $3n$  级别的存储方式, 当然也可以添加其他分量. 它是不规则 (irregular) 且无序的数据. 换言之, 其数据的序并不是必要的, 不能提供额外信息, 如同 set 一样. 好的算法应该尽可能不去运用序. 它是非常轻量级, 紧致的, 线性级别的存储空间. 容易存储, 容易理解和生成, 容易在其上构建算法.

当然它也不是完美的, 比如并不容易通过它判断表面的位置. 点云实际上是在表面上采样, 那么怎样从一个 mesh surface 上取样呢?

Sampling Strategy:Uniform Sampling. 计算每个表面的面积, 以面积为权独立同分布地选取三角形. 在三角形里如何均匀选取呢? 仿射变换形成直角三角形, 再拼接成矩形.<sup>20</sup>

但是这样取样之后, 在取样量不算非常大的时候, 经常会出现不太均匀的情形. 此时我们可以采用 Farthest Point Sampling(FPS). 目标是选取  $N$  个点使其两两距离求和最大. 但这是一个  $\mathcal{NP}$ hard 的问题. 我们采取一个近似的贪心算法. 我们先均匀选取 10000 个点. 在这 10000 个点最远的 1024 个点成为组合优化问题. 但仍然比较困难. 我们先确定一个点, 最后依次选取最远的点. 我们也只是希望点云看起来比较均匀, 因此没必要一定取得数学上的最优解.

---

<sup>20</sup>2022 笔者内心 OS: 直接拼接成平行四边形如何?2024 笔者注: 对三角形均匀采样可以在正方形 (平行四边形) 中采样然后对角线截半.

除此之外, 点云的距离度量成为一个问题, 这也是无序带来的问题之一. 我们希望找到一个 permutation invariant 的度量: Chamfer distance.<sup>21</sup>

$$d_{CD} = \sum_{x \in S_1} \min_{y \in S_2} \|x - y\|_2 + \sum_{y \in S_2} \min_{x \in S_1} \|x - y\|_2 \quad (95)$$

对于每个单项, 称为 uni chamfer distance. 在一个点云是另一个子集的时候有用.

另一个度量是 Earth Mover's distance<sup>22</sup>. 与 CD 不同的是, 它要求两个点云数量相同, 且每个点必须找到互不重复的对应.<sup>23</sup>

$$d_{EMD}(S_1, S_2) = \min_{\phi: S_1 \rightarrow S_2} \sum_{x \in S_1} \|x - \phi(x)\|_2 \quad (96)$$

CD 对于取样情况不太敏感, 而 EMD 则比较敏感. 比如同样对于 Stanford bunny, 如果一个点云多集中在头部, 另一个比较均匀, 则 CD 变化不大而 EMD 变换显著. 由于点云是 surface+sampling, 因此如果对于取样有要求, 应该使用 EMD.

### 14.3 CD vs EMD

区别在于 CD 对于采样不敏感, 而 EMD 对于采样敏感. 同时 EMD 需要两个点云点的数量相同.

也就是如果 mesh 一样但是 sample 不一样, 那么 EMD 会很大

### 14.4 Implicit Shape

另一种表示 3D 数据的方法是隐式表达, 即 SDF(signed distance function). 简单地说, 这个函数表示空间中点到物体表面的距离, 内部为负, 外部为正. 其数学定义为: 设  $\Omega$  为度量空间  $X$  的子集,  $d$  为  $X$  的度量, 则 SDF 定义为

$$f(x) = \begin{cases} -d(x, \partial\Omega) & \text{if } x \in \Omega \\ d(x, \partial\Omega) & \text{if } x \in \Omega^c \end{cases} \quad (97)$$

其中  $d(x, \partial\Omega) \stackrel{\text{def}}{=} \inf_{y \in \partial\Omega} d(x, y)$ .

另外, SDF 还满足 Eikonal equation, 即

$$\|\nabla F\| = 1. \quad (98)$$

这点不难理解, 因为沿与距离最短的点的连线方向的方向导数之模为 1, 而函数本身即表达距离, 不可能只前进一个单位的长度, 距离变化超过一单位, 因此这也是最大值.

implicit representation 的形式非常具有潜力, 因为它可以很容易地与神经网络结合. 因此我们下一个主题就是 3D 的 Deep learning.

<sup>21</sup>在某些文献当中, 式 95 有时也会带平方. 但是如果带平方, 则三角不等式不成立.

<sup>22</sup>在 WGAN 中使用. 2024 笔者注.

<sup>23</sup>Earthmover, 中文直译为推土机. EMD 距离用于衡量 (在某一特征空间下) 两个多维分布之间的 dissimilarity, 它的计算基于著名的运输问题. 但是精确求解 EMD 亦非易事, 调用的 package 也多为近似算法.

## 15 3D Deep Learning

对于点云来说, 直接拉成 vector 显然是不行的. 因为点云本身无序. 有一些直接的处理方法: 转换成 voxel grid(从点云到表面, 再重建 voxel), 或者 2D 投影 (这属于是越活越回去了). 我们想要的 network 需要对  $N!$  种不同的点云顺序具有不变性.

一种想法: 排序赋予序关系如何?

我们用 Permutation Invariant 的函数:Symmetric Function. 也就是

$$f(x_1, x_2, \dots, x_n) \equiv f(x_{\pi_1}, x_{\pi_2}, \dots, x_{\pi_n}), \forall x \in \mathbb{R}^N \quad (99)$$

实际上, 这样的函数随处可见. 比如均值, 极值等. 文献 [?] 、 PointNet.<sup>24</sup>

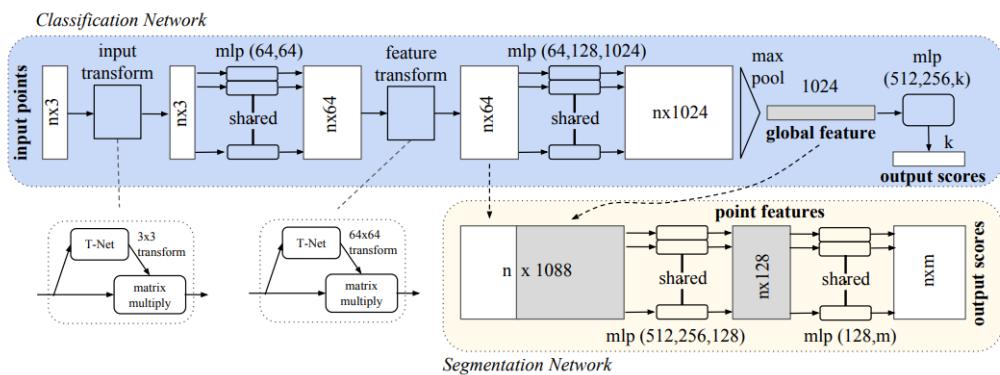


Figure 2. **PointNet Architecture.** The classification network takes  $n$  points as input, applies input and feature transformations, and then aggregates point features by max pooling. The output is classification scores for  $k$  classes. The segmentation network is an extension to the classification net. It concatenates global and local features and outputs per point scores. “mlp” stands for multi-layer perceptron, numbers in bracket are layer sizes. Batchnorm is used for all layers with ReLU. Dropout layers are used for the last mlp in classification net.

图 37: PointNet 的结构图

做 segmentation: 将每个点自身的特征 (local) 和 global feature 结合, 过 MLP 就能进行分类.  
诸多挑战: Resolution,Occlusion,Noise,Registration.

PointNet 对于 Resolution 和 Noise 比较 robust. 在 ModelNet40 上, 去掉 50% 的数据, Furthest 的准确率只下降了 2%. 能够如此稳定的关键在于 max 函数, 在连续取样时, 一些点的丢失可以用周围的点来弥补. 而且如果 pointnet 只有 1024 个 feature, 那么它最多也只会选取 1024 个点, 可能去掉的点根本就没有用到.

PointNet 的不足

一步直接 maxpooling, 没有 local context. 因此有了 PointNet++. 它仍是目前不追求过高性能时对 point cloud 处理的首选方法. 即设定一定的范围, 将某个点及其周围的点过 MLP, 再 maxpooling. 等同于在 image 里进行 conv. 当然, 求 neighbour 可能比较花时间. 另外, 所有的 neighbour 都通过了同样的 MLP, 毕竟在点云里也无法和图像一样, 区分哪个在什么方向.

此外, pointnet++ 怎么进行分辨率下降呢? 2d 中是 pooling, 而 pointnet 则进行 grouping, 从前而借.

<sup>24</sup>王老师:“虽然我不在这个组里, 但 PointNet 这篇论文的诞生我也是亲眼目睹过的, ICCV 的投稿截止在 11 月, 9 月底大家还不知道投什么, 于是有人说要不试试 PointNet 吧. 写到最后只有两个周调参了, 还是打不过 voxel, 于是加了 T-transform. 我们这里不讲, 因为现在几乎没有用这个的了”.(笑)

在恢复的时候, 添加的点没有 feature, 如何处理 (2D 可以进行 Deconv, 自然具有特征)? 添加 interpolate, 新的点选择与它最近的三个点, 按照距离反比为权插值获得特征.

3D: voxel Net->sparse conv, 屏蔽没有值的 voxel, Hash 有值的坐标, 从 NVIDIA,cuda 的层次写起.

Sparse conv: kernel 是空间各向异性 (spatial anisotropic) 的. 因此它的 acc 能够达到惊人的 70% 以上, 而 point cloud 无论如何添加特性, 最多也在 64% 左右. 而且因为可被标签, 使用更加方便. 但是它的分辨率受限.

Point cloud: 高分辨率, 鲁棒性, 但是表现略差.

### 15.1 PointNet++ V.S. Conv

PointNet++ 相当于卷积网络的先做  $1*1$  conv 然后  $n*n$  MaxPooling, 都是变少了点的数目, 但是每一个点的特征变多了

区别在于: PointNet++ 对于一个局部部分, 不同的点是各向同性的, 但是卷积网络对于一个局部是不同的, 因为一个卷积核不同位置的权不一样.

具体表现在: 如果把一个邻域内的点更换位置, 那么 PointNet++ 的结果不变, 但是卷积网络的结果会变.

这是由于两个原因:

1. 卷积网络的卷积核是  $3*3$  的, 而 PointNet++ 的 MLP 是全局的, 相当于  $1*1$  的 CONV, 而  $1*1$  的 CONV 很容易想到是各向同性的, 如果把一个邻域内的点更换位置, 那么每个点的 MLP 结果不变.
2. 卷积网络的 MaxPooling 不是全局的, 而 PointNet++ 的 MaxPooling 是全局的, 因此如果把一个邻域内的点更换位置, 那么卷积网络的结果会变, 而 PointNet++ 的结果不变, 因为怎么变换位置都是取最大值.

参数量:  $C*C'$  V.S.  $3*3*3*C*C'$

## 16 Object Detection and Instance Segmentation

### 16.1 任务简介

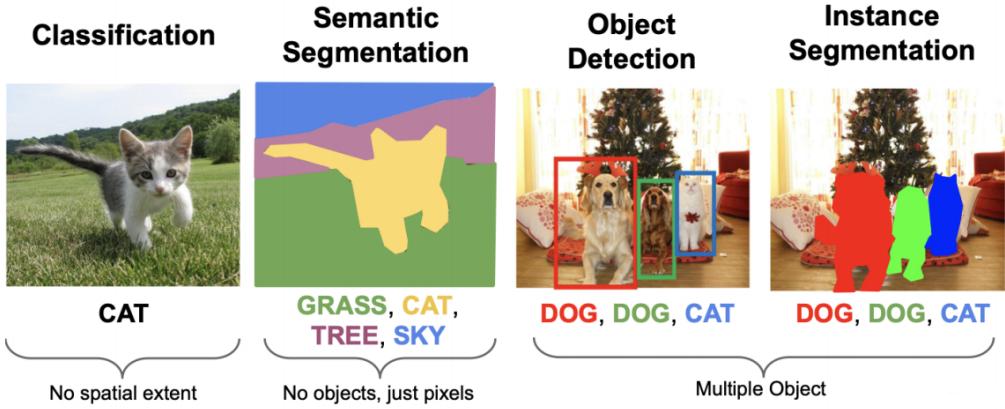


图 38: 分类任务

如上的任务当中, 最左侧的是分类, 即一张图片里只有一个待分类的事物. 随后是语义分割, 即将代表不同语义的像素区分开来, 每个像素都要有一个输出, 它不区分不同的个体. 随后是目标检测, 它区分不同个体, 但并不一定每个像素都被一个 bounding box 包围. 最后在此之上可以继续进行语义的分割.

我们先从 Object Detection 中最简单的 Single Object 说起. 它的目标是定位和分类, 网络输出是 2D 的 bounding box, 且是 axis aligned 的.<sup>25</sup>确定 bounding box 需要四个参数  $x, y, w, h$ , 即 bbox 左上角的坐标和大小.<sup>26</sup>

总的来说, 单目标检测可以被划分成两个任务:(对图片的) 分类和 (对 bbox) 的回归. 如下图:

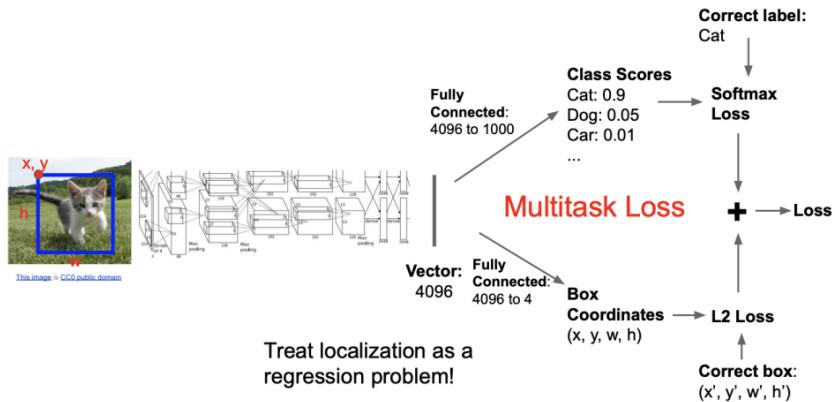


图 39: 单目标检测的网络概念图

<sup>25</sup>到了三维的时候, 由于阶数的提升, axis-aligned bounding box 中有非常多的部分并不属于这个物体, 因此可能需要旋转. 但是对 2D 来说, 这并不成为问题.

<sup>26</sup>如果我们用八个点的坐标作为表示, 那么显然有很多冗余. 尤其在高维的情形, 如何找到冗余尽可能少而能够便利地表达某一对象的方法是非常重要的. 在后面的旋转部分, 我们还会看到这一点.

假设我们得到了图片的特征向量, 那么可以通过 MLP 变成分类概率的向量, 然后使用交叉熵进行度量; 然后我们对图片 MLP 到四个数值, 然后用  $L^2$  loss 进行回归<sup>27</sup>.

这是一个 Multitask Loss, 即分类 loss 和 bounding box 的 loss 之和, 可能产生竞争. 它们应该以何种比例叠加? 比如分类的 loss 最大概在  $\log N$  的量级, 但是对 bounding box 来说, 如果采用  $L^2$ , 差 5 个 pixel 那就是 25. 而  $e^{25}$  是一个非常大的数字. 即使 loss 的量级差不多, gradient 的量级也不一定是相同的.

除此之外, 这个网络还有一个特殊性质: 对于不同图片, 可能需要不同数量的输出. 这在以前的 Neural Network 是无法做到的. 传统的方法是用 sliding window 进行遍历检测, 但这也自然而然带来一个问题: 你怎么知道 window 多大呢? 后来的传统方法采用了 Region Proposal 的方法, 即先提出一些可能是物体的 bbox, 在其中进行检测. 后来将这些 proposal 称为 Region of Interest(RoI).

## 16.2 Region-based CNN

第一个深度学习的相关工作 [?]:R-CNN(Region-based CNN). 它的大致流程是: 先提出一些 proposal, 然后通过 SVM 进行分类, 以及对 bbox 的回归, 回归的输出是  $(dx, dy, dh, dw)$ , 即 bbox 应该进行的调整. 为了处理不同大小的 bbox 输入, RCNN 将所有的 bbox 覆盖的部分统一变换到 224\*224 的大小.

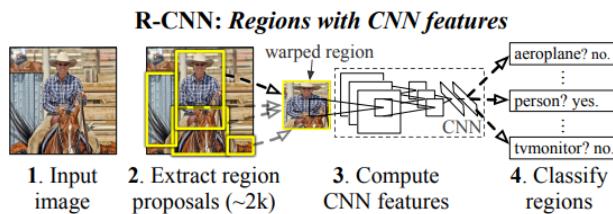


图 40: RCNN 分类部分

训练时我们会有很多的 RoI, 但 ground truth(后文简写成 gt) 的数量显然要少得多. 那么如何确定一个 RoI 被分成哪一类, 以及向哪个 gt 进行回归呢? 首先, 如果一个 bbox 与任何一个 gt 的交都小于某一阈值, 那么就将它单独分类成 background, 此时我们不再关心其 regression 的情况; 如果一个 bbox 同时包含了多个 gt, 那么可以计算它与这些 gt 的 IoU, 将 IoU 最大的那个 gt 作为其分类和回归的对象. 这里要注意的是, 我们不关注 background 的回归是必须采取的行为, 因为如果要监督一个 background 的回归, 那么它的 loss 是非常大的, 会直接破坏其他 RoI 的训练.

这个工作的两个问题: 首先 proposal 可能太多了, 在测试时速度太慢. 其次若给出的 proposal 可能有缺损, 将其单独提取出来之后的部分无法获知周围的信息, 几乎不可能准确对 bbox 进行回归.

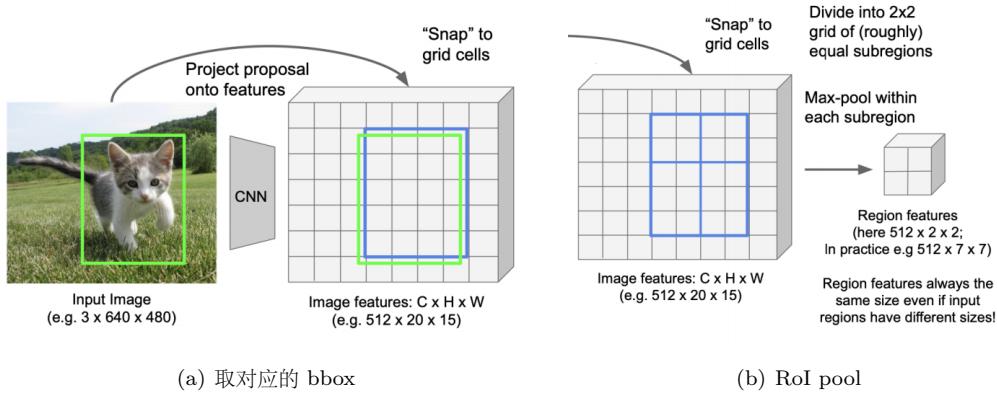


图 41: Fast RCNN

### 16.3 Fast R-CNN

随后提出的 Fast R-CNN 解决了上面的两个问题. Fast RCNN 仍然采用传统方法获得 RoI, 随后将整个图片过 CNN, 再在 feature map 上取出对应的 bbox. 图像在卷积的过程中分辨率会减小, 因此在 feature map 上取的时候也要对应地缩小 RoI, 如图 41(a)所示, 缩小过程中不在格点上的顶点被吸附到最近邻的格点上.

这样取得的 RoI 大小仍然不统一, 原论文采用了一种 max pooling 的方法, 将形式各异的 RoI 分割成合乎要求的子块, 对每个子块求最大值获得期望的形状, 如图 41(b)所示. 图中为方便将最终的大小画成了  $2^*2$ , 实际上为  $7^*7$ .

这样做的好处是, 可以将多出来的 RoI 数量这一维度作为 Batch 的一部分, 从而实际上并不会明显扩大工作量, 因为最后的 RoI 分辨率比较小. 以  $7^*7$  为例, 1000 个 RoI 总共约 50k 个数据, 与  $224^*224$  大小的原图相仿. 另外, 经过 conv 后, 感受野也增大了, 也就不会产生上文提到的被裁剪而缺少上下文的问题了.Fast RCNN 相比 RCNN 取得了可观的速度提升. 下图左侧, Fast RCNN 的训练时间只有 RCNN 的约十分之一, 且测试速度显著提高. 右图红色为不包含传统方法获得 RoI, 仅对 RoI 进行处理的时间, 可以看出时间的限制在 RoI 的提出上了.

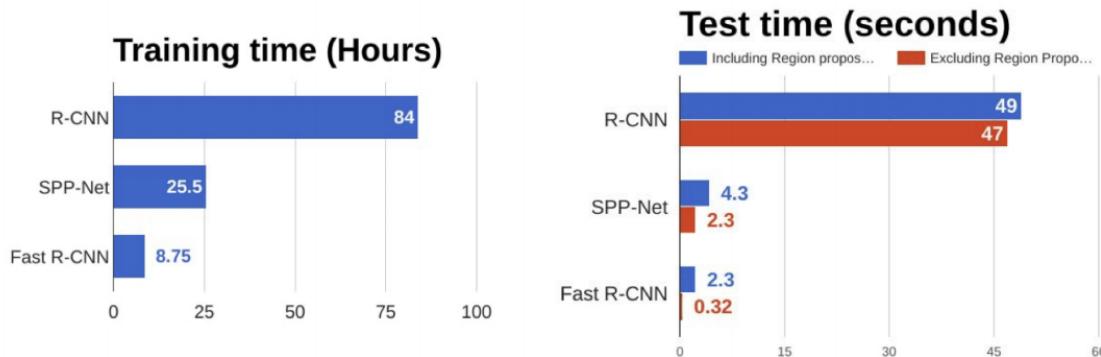


图 42: RCNN v.s. Fast RCNN

<sup>27</sup>注意  $L^2$  norm 和  $L^2$  loss 的区别. 前者是模长, 后者是平方和. 当  $L^2$  norm 作为损失函数时, 被称为 rooted mean squared error(RMSE), 而另一个则是 mean squared error.L2 在接近收敛的时候, 梯度也小, 不容易越过. 但是 L2 在较大的时候, 梯度很大, 容易导致神经网络训练效果不好, 而且还容易出现 NaN.RMSE 因为开根号的问题, 在接近收敛时反而可能出现问题.soft L1: 结合两者.

## 16.4 Faster R-CNN

经过 R-CNN 和 Fast R-CNN 的积淀, Ross B. Girshick 在 2016 年提出了新的 Faster RCNN. 在结构上,Faster RCNN 已经将特征抽取 (feature extraction),proposal 提取,bounding box regression,classification 都整合在了一个网络中,使得综合性能有较大提高,在检测速度方面尤为明显.

Faster R-CNN 引入 Anchor box 的概念. 在 Fast R-CNN 中, 我们已经得到了 feature map, 而其中的每个 pixel 都可能含有不同形状的一个或多个物体, 直接将取每个 pixel 作为 RoI 是不合理的.Faster R-CNN 让每个 pixel 都给出一些不同形状的 anchor(如图 43所示). 这个部分被称为 Region Proposal Network(RPN).

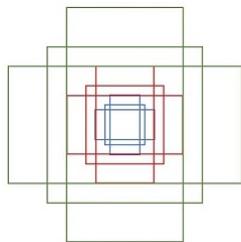


图 43: 不同形状的 anchor

当然,这样会获得数千个乃至更多的 bbox, 而一张图片里一般并没有这么多, 而这里我们并没有 gt 可供参考, 因此我们还要进行 NMS. 具体来说, 先将这些 bbox 进行分类预测, 按照它们的分类进行分组, 随后对每个类型的分组内取分类概率最高的 RoI, 将同组之内和它 IoU 大于某个 threshold 的 RoI 全部去除. 这样做的原理是将某一类别概率最高的作为标准, 与其 IoU 较大的则认为是圈出了同一个物体, 全部去除; 对于剩余的 RoI 重复这一操作.<sup>28</sup>

当然,Faster R-CNN 之中还有非常多的细节, 比如 anchor 如何取定, 如何确定训练 RPN 的 positive/negative samples, 如何参数化 bbox 回归的过程等, 限于篇幅限制无法一一展开, 读者可参考 [?].

Faster R-CNN 中包含了四种 loss:RPN 对是否是物体的二分 loss,RPN regress box loss, final classification score, final box coordinates, 调参的过程自然是非常复杂的, 但是网络的效果也是显而易见的 (如图 44).Faster R-CNN 的出现, 使得目标检测不再是学术界的 toy model, 而是真正进入了实用领域, 促进了安保等行业的发展.

## 16.5 two-stage detector and one-stage detector

如上图,Faster R-CNN 是一个分两步的目标检测网络. 有人提出: 既然在第一步里也做了 bbox 的 refine, 那么能否去掉第二步呢? 这就诞生了 single-stage detectors, 代表有 YOLO 系列, 它的特点就是非常快, 目前能达到 120 fps. 总体来说,two stage 准确率占优, 而 one stage 速度更快.

## 16.6 Evaluation Metric: mAP

无论是一步还是两步, 都需要面临的问题: 我们如何评估预测结果? 假如我们有一张图, 有 20 个 ground truth bounding box, 我们显然不可能要求其完全相符. 我们可以定义一个 IoU threshold. 首先其输出的类别要对, 其次  $\text{IoU} > \text{threshold}$ . 另外一方面, 如果我乱猜了 5000 张, 显然也是不行

<sup>28</sup>举例来说, 假设猫分类的 RoI 中概率最大者圈住了一只猫的绝大多数, 因而以 90% 的 confidence 认为是猫, 则其他与它 IoU 大于 0.5 的 RoI 很可能只圈住了这只猫的半边身子, 所以把它们都去掉. 而 IoU 较小的可能是其他的猫. 当然, 这样做也存在一些问题, 比如有一个和它非常接近的 RoI 因为某种原因识别为狗, 那么这样做就不能去除这样的 RoI, 因为此处 NMS 只对同类的 RoI 进行操作. 后来也有工作同时预测 RoI 与 IoU(即“预测的预测”), 并证明这样做效果更优.

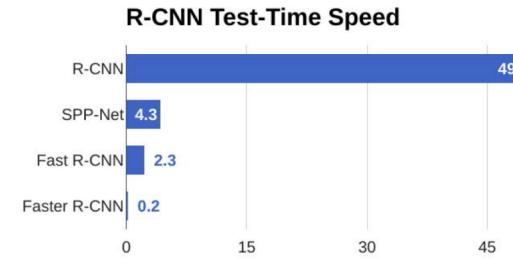


图 44: 几种网络的速度比较

这是边注. 这段话测试一下  
边注的分段和位置.  
庚信平生最萧瑟, 暮年诗赋动江关.

$$a = b$$

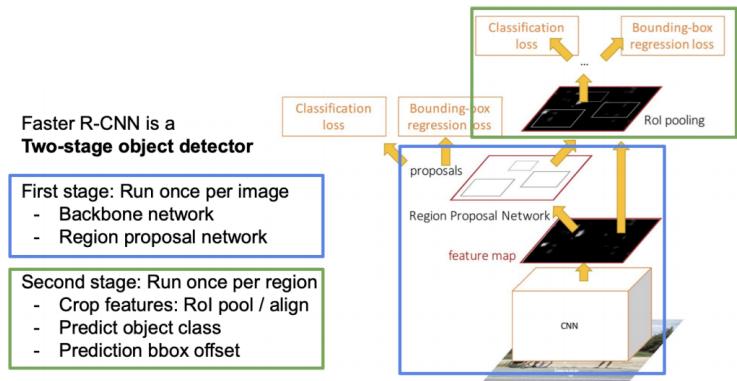


图 45: Two Stage 示意

的.trade-off between recall & precision. 因此提出度量:AP. 即 Average Precision. 即 precision-recall 图像下的面积. 它先选出某个种类, 然后按照 prob 排序, 逐个增加. 这样 precision 下降,recall 提升.mAP 就是所有不同 category and/or IoU threshold.

Object detection 变量非常多. 若要准确度, 则 Faster R-CNN. 若要快速:YOLO. 但目前这一领域, 工业界已经占据了统治地位.

## 17 Instance Segmentation

### 17.1 Mask R-CNN

在目标检测上再进一步, 输出哪个 pixel 属于哪个 segmentation.

两种方法:bottom-up, top-down. 目前在 2D 中前者较好, 因为 bbox 已经做得很好. 后者在 3D 中有用.

Top-Down Approach:Mask R-CNN<sup>29</sup>

首先, 经过 RoI pooling 之后, 分辨率可能下降. 但这是大家都知道的.

最重要的在于, 何恺明指出, 我们不能进行最近邻的吸附, 否则会不匹配, 这是不可能被学习到的. 因此何恺明使用双线性插值进行处理,RoI align.

Ablation Study on RoI Align.AP at 75 提升比 AP50 还高, 这是因为这样的方法对于高精度影响更大. 此外, 加入 align 后,bounding box 的表现也提升了. 额外的信息量.synergy.(what is mask?)

### 17.2 3D Object Detection and Instance Segmentaiton

3D object detection 部分过于复杂, 只做了解, 不再详细记录.

---

<sup>29</sup>何恺明是如何让这个看起来大家都看不起的工作拿到了 ICCV 2017 best paper 呢?

## 18 Pose and Motion

### 18.1 Beyond Detection: Pose

Pose 是物体刚性运动的表征. 二维 bbox 拥有四个自由度. 三维有六个. 若有转动角  $\theta$ , 则为七个.

rotation 包含物体的朝向信息. 定义六维物体姿态 (6d object pose) 为 3 平动 (translation), 3 转动 (rotation).

旋转矩阵  $\mathbf{R}$  满足  $\mathbf{R}^\top \mathbf{R} = \mathbf{I}$  且  $\det \mathbf{R} = 1$ . 它属于群  $\text{SO}(3)$ <sup>30</sup>.

旋转矩阵只有三个自由度, 但却包含九个元素, 这使得神经网络难以预测. 我们需要其他的表达方式.

### 18.2 Euler Angle

$$R_x(\alpha) := \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \alpha & -\sin \alpha \\ 0 & \sin \alpha & \cos \alpha \end{bmatrix} \quad R_y(\beta) := \begin{bmatrix} \cos \beta & 0 & \sin \beta \\ 0 & 1 & 0 \\ -\sin \beta & 0 & \cos \beta \end{bmatrix} \quad R_z(\gamma) := \begin{bmatrix} \cos \gamma & -\sin \gamma & 0 \\ \sin \gamma & \cos \gamma & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (100)$$

当然, 具体使用的时候, 有不同的 convention: 比如, 按照什么顺序依次旋转轴? 旋转是使用变换前的轴, 还是变换后的轴?

但两个操作复合时, 并不能简单将两个角相加.

### 18.3 Axis Angle

寻找瞬时转轴和角度,  $\mathbf{e}, \theta$ . 得到  $\theta = \theta \mathbf{e}$ , 三个变量可以取任意值.

应用 Rodrigues' rotation formula, 可以将 axis-angle 转变为 rotation matrix:

$$\mathbf{R} = \mathbf{I} + (\sin \theta) \mathbf{K} + (1 - \cos \theta) \mathbf{K}^2 \quad (101)$$

其中  $\mathbf{K} = [\mathbf{e}]_\times$ .

Axis Angle 非常好地表示了旋转的特征, 且其  $\theta$  是不随坐标系选取而变化的. 但是它仍然存在问题: 当我们获知两个 AA 的  $\theta, \mathbf{K}$  时, 其复合也不能由它们简单运算得到.  $\text{SO}(3)$  在李群, 李代数当中有很漂亮的形式, 以及与 AA 的联系.

### 18.4 Quaternion

Quaternion 即四元数, 表达形式是由一个实部和三个虚部组成:

$$q = w + xi + yj + zk. \quad (102)$$

<sup>30</sup>SO 的含义是 Special Orthogonal, 前者代表行列式为 1.

其中

$$\begin{aligned}\mathbf{i} * \mathbf{i} &= -1 \\ \mathbf{j} * \mathbf{j} &= -1 \\ \mathbf{k} * \mathbf{k} &= -1 \\ \mathbf{i} * \mathbf{j} &= -\mathbf{j} * \mathbf{i} = \mathbf{k} \\ \mathbf{j} * \mathbf{k} &= -\mathbf{k} * \mathbf{j} = \mathbf{i} \\ \mathbf{k} * \mathbf{i} &= -\mathbf{i} * \mathbf{k} = \mathbf{j}\end{aligned}\tag{103}$$

运算律:

$$\begin{aligned}\mathbf{q}_1 * \mathbf{q}_2 &= (w_1 w_2 - x_1 x_2 - y_1 y_2 - z_1 z_2) \\ &\quad + (w_1 x_2 + x_1 w_2 + y_1 z_2 - z_1 y_2) \mathbf{i} \\ &\quad + (w_1 y_2 - x_1 z_2 + y_1 w_2 + z_1 x_2) \mathbf{j} \\ &\quad + (w_1 z_2 + x_1 y_2 - y_1 x_2 + z_1 w_2) \mathbf{k}\end{aligned}\tag{104}$$

共轭: $q = w - i - j - k$ . 性质为

$$\|\mathbf{q}\| = \sqrt{\mathbf{q} * \mathbf{q}'} = \sqrt{w^2 + x^2 + y^2 + z^2}\tag{105}$$

若模为 1, 则是单位四元数. $\mathbf{q}^{-1} = \mathbf{q}'$ . 乘法满足结合律但不满足交换律.

如何表达旋转? scalar+vector 的表达方式:

$$\mathbf{q} = (s, \mathbf{v})\tag{106}$$

得到

$$\mathbf{q}_1 * \mathbf{q}_2 = (s_1 s_2 - \mathbf{v}_1 \cdot \mathbf{v}_2, s_1 \mathbf{v}_2 + s_2 \mathbf{v}_1 + \mathbf{v}_1 \times \mathbf{v}_2)\tag{107}$$

一个单位四元数对应一个旋转. $s = \cos \frac{\theta}{2}, \mathbf{v} = e \sin \frac{\theta}{2}$ .

对于一个向量  $\mathbf{x}$ , 其运算方式为将  $\mathbf{x}$  补成四元数  $\mathbf{x} = (0, \mathbf{x})$ , 求  $\mathbf{x}' = \mathbf{q} \mathbf{x} \mathbf{q}^{-1}$ . 计算复合只需要四元数相乘.

四元数实际上是一个四维空间上的超球面  $S^3$ . 很遗憾, 它也不是欧式的.

How to Estimate Rotation use Neural Network?

方法一: Use a neural network to regress a rotation representation.<sup>31</sup>

方法二: Predict object coordinate or correspondence and then solve rotation.

Orthogonal Procrustes Problem:

$$\hat{\mathbf{A}} = \underset{\mathbf{A} \in \mathbb{R}^{p \times p}}{\operatorname{argmin}} \|\mathbf{M} - \mathbf{N}\mathbf{A}\|_F^2 \quad \text{subject to} \quad \mathbf{A}^T \mathbf{A} = \mathbf{I},\tag{108}$$

The solution can be expressed in terms of the SVD of a special matrix<sup>32</sup>.

$$\mathbf{M}^T \mathbf{N} = \mathbf{U} \mathbf{D} \mathbf{V}^T, \text{ then } \hat{\mathbf{A}} = \mathbf{V} \mathbf{U}^T\tag{109}$$

SVD is very sensitive to outliers. For fitting rotations, we need to use RANSAC.

How many pairs of 3D-3D correspondence do we need for hypothesis generation? 2 pairs(如果连线不与转轴平行).

<sup>31</sup>在此祝愿助教 Jiayi Chen 的论文被顺利接收.

<sup>32</sup>这里为了保证行列式为 1, 可以令  $\hat{\mathbf{A}} = \mathbf{U} \Lambda \mathbf{U}^T$ , 其中  $\Lambda = \operatorname{diag} \{1, 1, \det \mathbf{V} \mathbf{U}^T\}$

## 19 Instance-Level 6D Object Pose Estimation

Instance-level: a small set of known instances.

Pose is defined for each instance according to their CAD model.

Input: RGB/RGBD. 如果有相机内参, 那么没有 D 也可以. 有 D 可以做得更好.

2D center localization. 先预测 2d 图片的中心位置和深度. 随后利用相机内参得到 translation.

PoseCNN: Translation Estimation:Voting. 每个 pixel 给出一个指向中心的向量, 得到 center.

PoseCNN: Rotation Estimation. RoI?

loss:  $\mathcal{L}(\mathbf{q}, \mathbf{q}^*)$ . 我们发现  $\mathbf{q}$  和  $-\mathbf{q}$  在旋转意义上是相同的,double coverage. 因此一种可行的 regression loss 是取两者的最小值.

PoseCNN 则采用了另一种 loss:

$$\text{PLoss}(\tilde{\mathbf{q}}, \mathbf{q}) = \frac{1}{2m} \sum_{\mathbf{x} \in \mathcal{M}} \|R(\tilde{\mathbf{q}})\mathbf{x} - R(\mathbf{q})\mathbf{x}\|^2 \quad (110)$$

对称性:(表示旋转的等价类)

$$\text{SLoss}(\tilde{\mathbf{q}}, \mathbf{q}) = \frac{1}{2m} \sum_{\mathbf{x}_1 \in \mathcal{M}} \min_{\mathbf{x}_2 \in \mathcal{M}} \|R(\tilde{\mathbf{q}})\mathbf{x}_1 - R(\mathbf{q})\mathbf{x}_2\|^2 \quad (111)$$

PoseCNN 的 translation 表现尚可, 但是 rotation 的表现一般, 这受限于四元数的性能.

6D pose 要求已知物体的 cad 模型, 这在现实中不太可能.

category-level 6D pose. 希望能够泛化, 输入 3d 输出 6d pose, Without the need to use CAD model.

王鹤老师的论文:Normalized Object Coordinate Space for Category-Level 6D Object Pose and Size Estimation,CVPR2019 oral.

Detecting and estimating 6D pose and 3D size of previously unseen objects from certain categories from RGBD images.

为什么要 depth 呢? 因为对于未知的物体来说, 仅有 rgb 而没有 depth 是无法确定其大小的. 有了 depth 和相机内参, 才能消除 scale 的不确定性.

问题的主要难点是 rotation 的估计. 前面我们看到 PoseCNN 即使对于已知的物体, 做得也相当不好.

间接法.Representation: Normalized Object Coordinate Space(NOCS)

简而言之, 我们需要对一张图片的像素预测其在 CAD model 中的位置. 你可能会问: 不是没有 CAD model 吗? 在此我们建立了一个 reference space: NOCS.

step 1:rotation Normalization:align object orientations. 将所有物体对齐成同样的姿态, 如马克杯的方向都向左, 此时旋转矩阵为 0.

Step 2 (translation normalization): zero-center the objects. 对于新物体, 将其紧 bbox 的中心作为原点.

Step 3 (scale normalization): uniformly normalize the scales. 将 bbox 的对角线长度设置为 1. 这样所有的都可以放入一个对角线长为 1 的正方体里了.NOCS = Reference frame.

NOCS = Reference frame transformation from NOCS to camera space.

为什么有内参  
没有深度也是可以的呢?  
因为这里我们是 Instance-level 的姿态估计, 换言之我们已经有了这个物体的形状参数, 其大小规格也是已知的. 理论上我们甚至可以不停地试  $\mathbf{R}, \mathbf{t}$  使得转换后的形状与照片符合.

这里我们隐含了一个假设, 即我们可以在没有其 CAD 的情形下讨论其朝向. 如马克杯的把手.

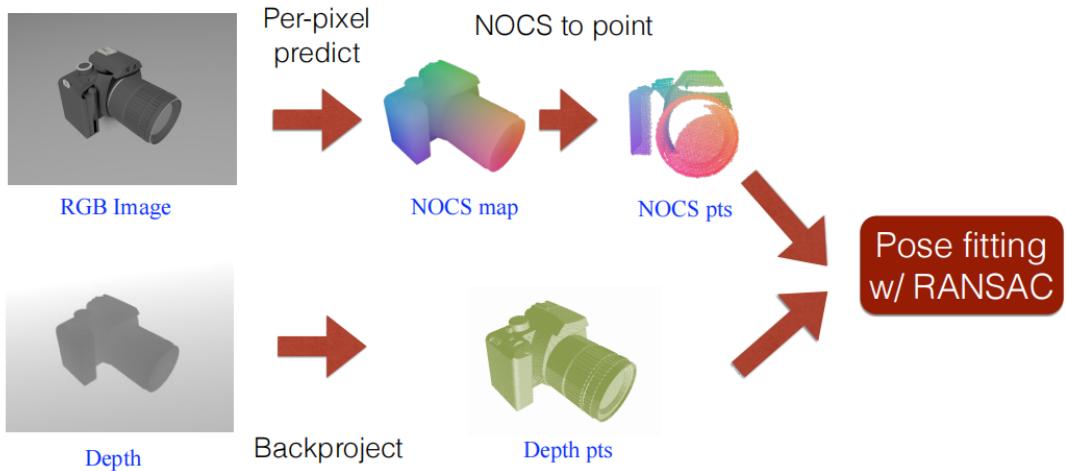


图 46: From Image to NOCS map to Pose.

### 19.1 Beyond Object Pose

human/hand pose estimation. 人体可以按照关节活动, 并不是刚体.

## 20 Motion

Today let's focus on motions between two consecutive frames!

Optical Flow 光流.

图片的亮的部分在两帧之间的表象运动.

几个假设: 亮度相对稳定, 小移动, 一个点的运动与其邻居相似.

$$\begin{aligned} I(x+u, y+v, t) &\approx I(x, y, t-1) + I_x \cdot u(x, y) + I_y \cdot v(x, y) + I_t \\ I(x+u, y+v, t) - I(x, y, t-1) &= I_x \cdot u(x, y) + I_y \cdot v(x, y) + I_t \\ \text{Hence, } I_x \cdot u + I_y \cdot v + I_t &\approx 0 \rightarrow \nabla I \cdot [uv]^T + I_t = 0 \end{aligned} \quad (112)$$

那么, 这个方程足够解出所有  $(u, v)$  吗? 我们有  $n^2$  个方程, 但有  $2n^2$  个未知数, 因此不够.

The Aperture Problem. 单纯从图像来看, 运动可能并不完整. Barberpole Illusion. 沿着线的方向不容易观测, 垂直的容易被观察到.

更多约束: Spatial coherence constraint. 1981 年 Lucas 和 Kanade 提出了假设在每个 pixel 的  $5 \times 5$  window 当中 flow 相同.

$$\begin{bmatrix} I_x(\mathbf{p}_1) & I_y(\mathbf{p}_1) \\ I_x(\mathbf{p}_2) & I_y(\mathbf{p}_2) \\ \vdots & \vdots \\ I_x(\mathbf{p}_{25}) & I_y(\mathbf{p}_{25}) \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} I_t(\mathbf{p}_1) \\ I_t(\mathbf{p}_2) \\ \vdots \\ I_t(\mathbf{p}_{25}) \end{bmatrix} \quad (113)$$

即  $\mathbf{A}_{25 \times 2} \mathbf{d}_{2 \times 1} = \mathbf{b}_{25 \times 1}$

得到

$$\mathbf{A}^\top \mathbf{A} \mathbf{d} = \mathbf{A}^\top \mathbf{b} \quad (114)$$

什么时候可解?

1. 可逆
2. 特征值不能太小
3. 良态

FlowNet: 最简单的想法: 两张三通道图 merge 在一起, 卷. dense regression. early fusion.

或者: 分别提取 feature. 两个网络 share weight. 然后结合到一起. middle fusion.

过早 fusion 会使得问题空间变大. 过完 fusion 会使得微观细节缺失.

这和我们之前的 Harris Corner Detector 非常相似. 光流当中最容易被捕捉的也是 corner.corner 与光流紧密相关.

## 21 RNN

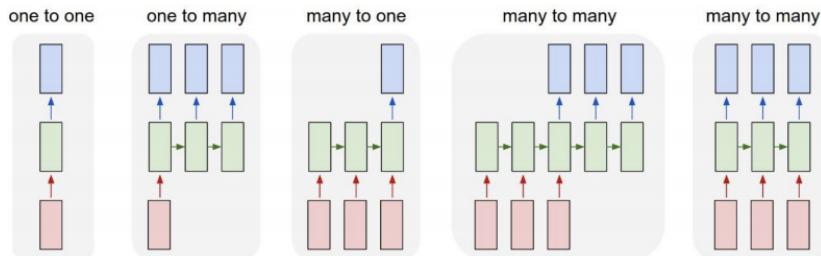


图 47: 各种输入输出方式

我们之前的网络, 都是 one-to-one, 一个输入, 一个输出.

one-to-many: 例如看图说话. image->sequence of words

many-to-one. 例如动作预测.

many-to-many. 例如 video captioning.

此外还有 offline 和 online 的处理. 前者全部看完, 后者边看边输出.

Recurrent Neural Network.

$$\begin{cases} h_t = f_W(h_{t-1}, x_t) \\ y_t = f_{W_{hy}}(h_t) \end{cases} \quad (115)$$

Vanilla RNN:

$$\begin{cases} h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t) \\ y_t = W_{hy}h_t \end{cases} \quad (116)$$

计算图, 各个方向流向 W.

## 21.1 Characer-Level Language Model

sample:greedy? weighted sampling, 根据字母的权, 但可能选到冷僻的, 比如 hh.? 更高级:beam search. 每次选三个, 两层三叉树, 选取最后的 top3?

实际上将 h 和 x 结合的时候, 常常先将 x 进行 embedding,  $W[h,x] \rightarrow W[h, g(x)]$ . 比如, hidden layer 可能之后 512 维, 但是输入如果是词向量, 那么可能有几万维, 非常不均衡. 因此将 embedding 到合适的维度. 此外, word embedding 已经有现成的处理.

BP 时, 不同位置的 W 被 loss 调用了不同次. 这个操作开销极大.

解决方法:truncated BP. 截断. 比如 FP 时是从 0 到 t, BP 只算从 t 到 t-6.

pytorch 的命令:stop\_gradient.

RNN 在长程记忆里会出现问题. delta t 一般被称为 sequence length. 如果过短则相关性不强, 过长则 cost 过多.

RNN tradeoff.

multilayer:2 层即可

## 21.2 Vanilla RNN Gradient Flow

tanh 的梯度恒小于 1, 梯度消失.

gradient clipping.

## 21.3 LSTM

信息传递的一部分放入  $c$ ?

cell state 是 long-term memory. 我们知道对于普通的 tanh, 往 01 之间映射, 那么  $h_t$  和很早之前的  $h_i$  之间的联系就比较微弱了. 而 LSTM 中的  $c$  则一直把信息保留 ( $f$  也是经过 sigmoid 激活的, 在 01 之间, 可以选择记住或者遗忘), 最后  $h$  将  $c$  进行处理.

核心结构:old info 和 new info 的加和. 若  $f=1$ , 则就是 skip link. 换言之  $c$  之间有梯度的旁路.

## 22 Video analysis

video=2d+time

数据量太过庞大.4D.

提取特征, 最后放在一起. 直接放在一起太大. 逐帧分析? 不管时间维度. 比如跑步录像. 但如果两脚都着地? 可能误判.maxpool? 抹去序关系反而不正确.

3D CNN? 将时间视作第三个空间维度. 但问题在于: 这对感受野的要求比较大. 在视频当中, 一件事情的效应可能在相当长的时间之后才能体现, 但 CNN 也不可能 cover 任意长的时间序列. 也就是说, 在视频处理当中, 空间范围和时间范围地位并不对等.

early fusion: 所有信息在第一步混在一区提取.3D CNN 在各个维度感受野的增长比较均匀.

C3D: The VGG of 3D CNNs. 第一次 pool 没有在时间维上处理, 不希望提取得太早.

3d 的 kernel size 比较敏感, $5^3 > 11^2$ . 移动多了一个维度, 计算代价更大. 得益于其网络的精良设计,C3D 的 sports-1m 的 top5 acc 还是达到了 84%.

人类识别运动的关键:Motion, 不是 pixel.

Use Both Motion and appearance: two-stream fusion network. 它使用经典算法获得 flow 输入神经网络. 神经网络分为两支, 一支做空间, 一支做时间. 时间 (flow) 这一支使用了 early fusion, 因为相对于 RGB, 光流的信息已经比较清晰.

Modeling Long-Term Temporal Structure. 我们希望处理序列,RNN 如何? aggregation(聚合).

CNN 和 RNN 一起训练计算代价太大. 因此先 train CNN, 不向其传递梯度, 只训练 RNN. 但这样 CNN 与 RNN 可能优化目标不同. 原来 RNN 是独立的网络吗?

end to end training: 端到端训练. 所有 optimization variable 同时被优化.

Recurrent convolutional Network.

## 23 Generative Models

discriminative vs generative.

Objectives:

learn  $p_{model}(x)$  that approximates  $p_{data}(x)$ . 换言之假设  $\text{image} \in \mathcal{X} = \mathcal{R}^{3 \times H \times W} \sim p_{data}$ . 我们想要习得这个分布.

隐式和显式.

本门课接触三个生成模型:pixelRnn, , gan

expilcity density model, or Fully Visible Belief Network (FVBN)

假设我们的隐式概率模型是  $p(x) = p(x_1, \dots, x_n)$ . 这一步没有引入任何信息. 应用链式法则可得:

$$p(x) = \prod_{i=1}^n p(x_i | x_1, \dots, x_{i-1}) \quad (117)$$

什么样的神经网络能够处理这样的连续的条件概率?(显然我们希望获得一个 shared 网络)

RNN. 但这样有问题: 首先网络太大, 其次如果要截断梯度, 语义不明.

CNN. 只依赖局部的 pixel.train 的时候可并行计算, 因为像素已经存在. 但生成仍然缓慢: 必须按照顺序生成.

说实话, 效果不咋地...

优点: 可显示表达, 容易优化, 采样好 (? 相对). 缺点: 慢

但是凭什么认为只和上面的像素相关呢?

### 23.1 Approximate density:Variational Autoencoder

变分自动编码器?

PixelRNN 实际上只是非常低维的. 因为只有少数取值.

VAEs:

$$p_\theta(x) = \int p_\theta(z)p_\theta(x | z)dz \quad (118)$$

如何进行学习?

L2 loss 和高斯噪声? 为什么输出会比较糊.

看起来在前面的 AE 当中, 我们只需要 decoder 部分就可以完成生成. 事实果真如此吗? 如果只有 decoder, 那么 z 服从何种分布完全不了解 (总不能是均匀分布吧). 需要它的分布容易 sample, 不要分布太广 (太过稀疏不利于网络学习).

两者都容易算, 但是其积分不容易计算. 蒙特卡洛, 维数太高.

我们此前接触的网络都是判别模型,given x, 判定 y,  $P(X|Y)$ . 上节课介绍的生成模型, 所有的变量就是 X, 我们学习  $P(X)$  或者  $P(X|Y)$ (if labels are available). 本学期我们准备介绍的 PixelRNN/CNN,VAE and GAN.

那么为什么要选用高斯分布呢? 缺失没有一种原则说明哪种分布更好.

$$\begin{aligned}
\log p_\theta(x^{(i)}) &= \mathbf{E}_{z \sim q_\phi(z|x^{(i)})} [\log p_\theta(x^{(i)})] \quad (p_\theta(x^{(i)}) \text{ Does not depend on } z) \\
&= \mathbf{E}_z \left[ \log \frac{p_\theta(x^{(i)}|z)p_\theta(z)}{p_\theta(z|x^{(i)})} \right] \quad (\text{Bayes' Rule}) \\
\text{We want to} \\
\text{maximize the} \\
\text{likelihood} &= \mathbf{E}_z \left[ \log \frac{p_\theta(x^{(i)}|z)p_\theta(z)}{p_\theta(z|x^{(i)})} \frac{q_\phi(z|x^{(i)})}{q_\phi(z|x^{(i)})} \right] \quad (\text{Multiply by constant}) \\
&= \mathbf{E}_z [\log p_\theta(x^{(i)}|z)] - \mathbf{E}_z \left[ \log \frac{q_\phi(z|x^{(i)})}{p_\theta(z)} \right] + \mathbf{E}_z \left[ \log \frac{q_\phi(z|x^{(i)})}{p_\theta(z|x^{(i)})} \right] \quad (\text{Logarithms}) \\
&= \mathbf{E}_z [\log p_\theta(x^{(i)}|z)] - D_{KL}(q_\phi(z|x^{(i)}) \| p_\theta(z)) + D_{KL}(q_\phi(z|x^{(i)}) \| p_\theta(z|x^{(i)}))
\end{aligned}$$

Decoder network gives  $p_\theta(x|z)$ , can  
 compute estimate of this term through  
 sampling (need some trick to  
 differentiate through sampling).

This KL term (between  
 Gaussians for encoder and  $z$   
 prior) has nice closed-form  
 solution!

$p_\theta(z|x)$  intractable (saw  
 earlier), can't compute this KL  
 term :( But we know KL  
 divergence always  $\geq 0$ .

图 48: 推导过程

## 23.2 重学 VAE

VAE 如果有统计学习的基础, 可以更容易地理解.

VAE 来自 Auto Encoder. 我们的生成模型有一个 (latent space?) 中的向量  $z$ . 一般取先验分布为  $\mathcal{N}(\mathbf{0}, \mathbf{I})$ . 所有的函数都需要支持概率输出. 这与 AE 不同, 后者一般只输出一个值. 那么如何从一个值变成分布呢? 将输出经过一个 Probabilistic model. 即, 你认定了它服从某种分布之后, 再转换为概率分布.(比如, 你认为它是高斯噪声, 那么这就是你的概率模型, 也就意味着你默许了生成图片中的高斯噪声. 当然, 你也可以选择其他概率模型.)

这里我们默许了  $z \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$

$$p_\theta(x) = \int p_\theta(z)p_\theta(x|z)dz \quad (119)$$

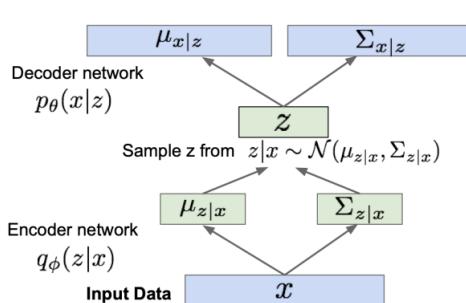
核心区别: 估计 mu, sigma

如何计算积分?intractable. 那么, MC 方法可以吗? 就是:

$$\log p(x) \approx \log \frac{1}{k} \sum_{i=1}^k p_\theta(x|z^{(i)}), \text{ where } z^{(i)} \sim p(z) \quad (120)$$

但是, 我们很难在 latent 空间中精准地找到某张照片对应的  $z$ , 因此取样的绝大多数结果都是 0, 效果必定差.

### Variational Autoencoder (VAE)



另一种方式: 利用 Bayes 公式, 如果能够引入后验概率  $p_\theta(z|x)$ ,

$$p_\theta(x) = \frac{p_\theta(x,z)}{p_\theta(z|x)} = \frac{1}{p_\theta(z|x)} p(z)p_\theta(x|z) \quad (121)$$

但是我们并没有从根本上解决问题. 现在, 我们希望学一个神经网络近似  $p_\theta(z|x)$ , 也就是学习一个  $q_\phi(z|x)$  来近似  $p_\theta(z|x)$ . 最后我们的网络结构如图. 如果是 AE, 那么就没有方差一项了, 唯一对应, 没有随机性. 这里, 注意  $q_\phi$  只是对  $p$  的近似.

那  $p_\theta(x|z)$  为啥不能学?

图 49: VAE 结构

$$\begin{aligned}
\log p_\theta(x^{(i)}) &= \mathbf{E}_{z \sim q_\phi(z|x^{(i)})} [\log p_\theta(x^{(i)})] \quad (p_\theta(x^{(i)}) \text{ Does not depend on } z) \\
&= \mathbf{E}_z \left[ \log \frac{p_\theta(x^{(i)}|z)p(z)}{p_\theta(z|x^{(i)})} \right] \quad (\text{Bayes' Rule}) \\
&= \mathbf{E}_z \left[ \log \frac{p_\theta(x^{(i)}|z)p(z)}{p_\theta(z|x^{(i)})} \frac{q_\phi(z|x^{(i)})}{q_\phi(z|x^{(i)})} \right] \quad (\text{Multiply by constant}) \\
&= \mathbf{E}_z [\log p_\theta(x^{(i)}|z)] - \mathbf{E}_z \left[ \log \frac{q_\phi(z|x^{(i)})}{p_\theta(z)} \right] + \mathbf{E}_z \left[ \log \frac{q_\phi(z|x^{(i)})}{p_\theta(z|x^{(i)})} \right] \quad (\text{Logarithms}) \\
&= \mathbf{E}_z [\log p_\theta(x^{(i)}|z)] - D_{KL}(q_\phi(z|x^{(i)}) \| p(z)) + D_{KL}(q_\phi(z|x^{(i)}) \| p_\theta(z|x^{(i)})) \quad (122)
\end{aligned}$$

我们试图最大化下界, 它被称为 Evidence Lower BOund(ELBO). 第一项最大化, 也就是让网络输出的分布的 theta 接近真实的分布 (均值接近, 方差小, 那么第一项就大! 另外注意上式的  $x^{(i)}$  并不是网络输出, 而是输入.) 第二项最大化, 也就是希望 q 与  $p(z)$  接近. 这是因为 VAE 必须知道  $z$  的分布情况, 否则生成的时候很难将  $z$  取值在概率密集的部分. 当然, 这和我们真正的目的还是有所偏差, 因为 loss 的要求是将每个  $x_i$  的生成的  $z$  都是 Gaussian, 但我们实际的意图是将所有  $x_i$  构成的集合生成的  $z$  的集合满足 Gaussian..

方差变大?

因此, VAE 的 loss 非常有趣, 它的  $\phi$  看似只出现在第二项, 但实际上还出现在第一项的  $z$  里面, 因此我们将  $z$  写成  $z = \mu_{z|x} + \epsilon\sigma_{z|x}$ , 从而有梯度可以回传. 另外, ELBO 的计算也是 intractable 的, 因为第一项的  $\mathbf{E}_z$  这一步就计算不出来, 我们直接扔掉期望, 取它自己作为 MC 的估计.

那你怎么不一开始就这么干?

我们说, 两个被估计的量分别是  $\log(\mathbf{E}_z [p_\theta(x^{(i)}|z)])$  和  $\mathbf{E}_z \log p_\theta(x^{(i)}|z)$ , 前者的  $z \sim p$ , 后者则是 q, 后者实际上更加集中. 而且 log 和期望的次序也交换了.

实际上我们有时直接令  $\Sigma_{x|z} = \mathbf{I}$ . 因为 loss 当中, 形式为

$$\exp^{-\left(\frac{x-\mu}{\sigma}\right)^2} \quad (123)$$

最后一个问题: 这东西跟变分 (variational) 究竟有什么关系? 要说起变分, 我们不得不先讲泛函. 泛函简单地说就是函数的函数, 或广义的函数. 变分, 是指自变量函数发生的变化, 为与自变量的变化  $dx$  区分, 我们一般用  $\delta$  表示. 例如我们想要求解空间中  $A, B$  两点之间最短的曲线, 设任意一条曲线为  $f(x, y)$ , 其长度为

$$l = \int_A^B f(x, y) ds. \quad (124)$$

当自变量函数  $f$  发生微小变化  $\delta f$  时, 长度也发生了  $\delta l$  的变化. 运用 EL 方程, 我们可以求得

$$\frac{\delta l}{\delta f} = 0 \quad (125)$$

时的函数  $f$ .

同时想起了我学得极差的数理方法和更差的数学分析

在 VAE 当中, 我们实际上也是希望找到  $q_\theta$  来近似  $p_\theta$ , 也是变分的过程. 但是实际上我们并没有使用任何变分法的技术, 因为我们已经加入了先验的知识进行参数化, 从而将搜索空间限制为  $\theta$  上, 而非无限维的函数空间.

### 23.3 VAE

前面我们曾经讲过 AutoEncoder 的概念, 其实它可以视为一种降维手段, 将原数据通过一定处理 (如多层神经网络) 获得其另一种表示 (或称编码), 这个过程就是 encode, 然后需要时可以进行解码恢复. 它本质上可以视为学习两个映射:

$$\begin{aligned}\phi : \mathcal{X} &\rightarrow \mathcal{F} \\ \psi : \mathcal{F} &\rightarrow \mathcal{X} \\ \phi, \psi = \arg \min_{\phi, \psi} &\| \mathcal{X} - (\psi \circ \phi) \mathcal{X} \|^2\end{aligned}\tag{126}$$

在这一节我们要讲述的 VAE 也是一种 AutoEncoder, 只不过它引入了一些概率上的内容, 编码解码不再是一一对应的, 而是服从一些概率分布. 具体来说, 我们有一些对  $\mathbf{X}$  观测而得到的量  $\mathbf{x}_i, i = 1, 2, \dots, n$ , 我们希望将其进行编码之后, 得到由另一些变量表示的形式, 后者被称为隐变量 (latent variable). 然后我们希望再从这些隐变量中恢复得到服从  $\mathbf{X}$  分布的量. 也就是说, 我们希望从  $\mathbf{X}$  的一些观测值出发, 试图用一些其他的变量  $\mathbf{Z}$  (即隐变量) 描述这个量的概率分布, 这样我们就间接习得了  $\mathbf{X}$  的分布  $p(\mathbf{X})$ , 同时我们还要给出  $\mathbf{Z}$  服从的分布  $q(\mathbf{Z})$ , 这样我们就可以依据分布  $q$  选取另外的  $\mathbf{Z}$ , 然后通过解码来生成一些新的  $\mathbf{x}_j$ , 后者仍然服从  $\mathbf{X}$  的分布, 但是我们此前没有见过的, 此时解码器就变成了一个关于  $\mathbf{X}$  的生成模型.

这里的  $\mathbf{X}, \mathbf{Z}$  都可能是随机向量.

我们整理一下上面的内容, 假设所有已知数据  $\mathbf{x}$  来自一个未知的概率分布  $P(\mathbf{x})$ , 我们希望用一组参数  $\theta$  来确定一个参数分布  $p_\theta(\mathbf{x})$  来拟合  $P(\mathbf{x})$ . 我们假定  $\mathbf{x}$  与另一些隐变量  $\mathbf{z}$  有关, 那么依据边缘分布和条件分布的相关性质可知

$$p_\theta(\mathbf{x}) = \int_{\mathbf{z}} p_\theta(\mathbf{x}, \mathbf{z}) d\mathbf{z} = \int_{\mathbf{z}} p_\theta(\mathbf{x} | \mathbf{z}) p_\theta(\mathbf{z}) d\mathbf{z}\tag{127}$$

这里  $p(\mathbf{z})$  一般被称为先验分布, 一般取其为标准正态分布. 而  $p_\theta(\mathbf{z} | \mathbf{x})$  则被称为后验分布. 编码器学习的就是  $p_\theta(\mathbf{z} | \mathbf{x})$ , 因为它代表了如何从  $\mathbf{x}$  转换到隐变量. 解码器学习的则是  $p_\theta(\mathbf{x} | \mathbf{z})$ .

正因为  $\mathbf{z}$  是先验的, 所以  $p_\theta(\mathbf{z})$  也可以写成  $p(\mathbf{z})$ , 因为它没有参数.

上式并不能解决我们的问题. 尽管我们确定了先验分布, 而且  $p_\theta(\mathbf{x} | \mathbf{z})$  可由解码器学习, 但上式的积分是难以计算的, 因为实际操作中我们不可能遍历  $\mathbf{z}$  所在的高维空间. 所以我们试图用贝叶斯公式曲线救国:

$$p_\theta(\mathbf{x}) = \frac{p_\theta(\mathbf{x} | \mathbf{z}) p(\mathbf{z})}{p_\theta(\mathbf{z} | \mathbf{x})}\tag{128}$$

那么  $p_\theta(\mathbf{z} | \mathbf{x})$  又怎么获得呢? 这其实是编码器负责学习的分布, 因此我们用编码器来学习它. 我们记编码器学习的分布为  $q_\phi(\mathbf{z} | \mathbf{x})$ .

这可能就是神经网络之道:  
凡是不容易算不好表示的东西, 通通丢给神经网络去学习...

那么如何度量学习的好坏呢? 我们知道两个分布的差异可以用 KL-divergence 度量.

最后我们用一句话来概括 VAE 的工作, 然后进入形式化的推导: 在 VAE 当中, 输入数据  $\mathbf{X}$  是来自于一个特定的先验参数分布  $p_\theta(\mathbf{x})$ , 随后我们同时训练 encoder 和 decoder, 使得在我们学习的后验分布  $q_\phi(\mathbf{z} | \mathbf{x})$  和真实后验分布  $p_\theta(\mathbf{z} | \mathbf{x})$  的 KL 散度  $D_{KL}(q_\phi \| p_\theta)$  作为度量之下的重建误差最小.

## 23.3.1 ELBO

计算学习的后验分布  $q_\phi(z | x)$  和真实后验分布  $p_\theta(z | x)$  的 KL 散度  $D_{KL}(q_\phi \| p_\theta)$  得到:

$$\begin{aligned}
 D_{KL}(q_\phi(z | x) \| p_\theta(z | x)) &= \int q_\phi(z | x) \log \frac{q_\phi(z | x)}{p_\theta(z | x)} dz \\
 &= \int q_\phi(z | x) \log \frac{q_\phi(z | x)p_\theta(x)}{p_\theta(z, x)} dz \\
 &= \int q_\phi(z | x) \left( \log(p_\theta(x)) + \log \frac{q_\phi(z | x)}{p_\theta(z, x)} \right) dz \\
 &= \log(p_\theta(x)) + \int q_\phi(z | x) \log \frac{q_\phi(z | x)}{p_\theta(z, x)} dz \\
 &= \log(p_\theta(x)) + \int q_\phi(z | x) \log \frac{q_\phi(z | x)}{p_\theta(x | dz)p_\theta(z)} dz \\
 &= \log(p_\theta(x)) + E_{z \sim q_\phi(z | x)} \left( \log \frac{q_\phi(z | x)}{p_\theta(z)} - \log(p_\theta(x | z)) \right) \\
 &= \log(p_\theta(x)) + D_{KL}(q_\phi(z | x) \| p_\theta(z)) - \mathbb{E}_{z \sim q_\phi(z | x)} (\log(p_\theta(x | z)))
 \end{aligned} \tag{129}$$

将上式重写成

$$\log(p_\theta(x)) - D_{KL}(q_\phi(z | x) \| p_\theta(z | x)) = -D_{KL}(q_\phi(z | x) \| p_\theta(z)) + \mathbb{E}_{z \sim q_\phi(z | x)} (\log(p_\theta(x | z))) \tag{130}$$

左侧第一项是我们希望最大化的输出的概率, 第二项是希望最小化的分布差异, 综合起来应该最大化左侧. 我们再来看右侧, 第一项是  $q_\phi(z | x)$  和  $p(z)$  之间的 KL 散度, 最小化这一项说明我们希望后验分布也符合正态. 最后一项最大化则是希望我们的解码器预测更加准确. 使用优化理论的常用手段, 我们将损失函数定义为

$$\mathcal{L}_{\theta, \phi} = -RHS = D_{KL}(q_\phi(z | x) \| p_\theta(z)) - \mathbb{E}_{z \sim q_\phi(z | x)} (\log(p_\theta(x | z))) \tag{131}$$

我们希望求得

$$\theta^*, \phi^* = \arg \min_{\theta, \phi} \mathcal{L}_{\theta, \phi} \tag{132}$$

现在我们来看看这两项具体如何运算. 第一项, 由于我们假定后验分布也是高斯分布, 两个高斯分布的 KL 散度有闭式解

$$D_{KL}(p(z | x) \| q(z)) = \frac{1}{2} \sum_{k=1}^d \left( \mu_{(k)}^2(x) + \sigma_{(k)}^2(x) - \ln \sigma_{(k)}^2(x) - 1 \right) \tag{133}$$

对于后验分布, 我们有

$$q(x | z) = \frac{1}{\prod_{k=1}^D \sqrt{2\pi \tilde{\sigma}_{(k)}^2(z)}} \exp \left( -\frac{1}{2} \left\| \frac{x - \tilde{\mu}(z)}{\tilde{\sigma}(z)} \right\|^2 \right) \tag{134}$$

得到

$$-\ln q(x | z) = \frac{1}{2} \left\| \frac{x - \tilde{\mu}(z)}{\tilde{\sigma}(z)} \right\|^2 + \frac{D}{2} \ln 2\pi + \frac{1}{2} \sum_{k=1}^D \ln \tilde{\sigma}_{(k)}^2(z) \tag{135}$$

这里我们常常令  $\sigma_i = 1$ , 即协方差矩阵为单位阵. 否则对于上面这一项, 网络可以通过一直增大方差的方式来减小 loss. 此时损失函数就变为 MSE.

因为我们假定后验分布的隐变量彼此独立, 即  $\Sigma = \text{diag}\{\sigma_1^2, \dots, \sigma_d^2\}$  因此多维的情形可以由一维计算后求和. 计算过程略.

算了, 还是看参考文献 [?].

## 24 GAN

GAN: 我劝 PixelCNN 和 VAE, 先把生成的这个理念先搞懂. 你 VAE 什么的都在搞概率论, 他能搞吗? 搞不了, 没这个能力知道吗?

minimax game:

$$\min_{\theta_g} \max_{\theta_d} \left[ \mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log \left( 1 - D_{\theta_d} \left( G_{\theta_g}(z) \right) \right) \right] \quad (136)$$

交替训练. 防止梯度相似?

但是有一点值得注意: 上式中  $\log(1 - x)$  这一函数在  $x$  接近 0 时梯度非常小. 而训练之初, discriminator 可以轻易辨别出 generator 生成的图片为假, 而 generator 的梯度又太小, 于是就形成了 discriminator 把 generator 按在墙角使劲打, 后者还跑不出去.

总之, 这个 loss 对于 generator 是很不利的.

Non-Saturating Loss functions.

如何衡量网络的表现? 很难有客观的度量.

Qualitative GAN Generator Evaluation

主观度量.Nearest neighbors: to detect overfitting, generated samples are shown next to their nearest neighbors in the training set. 这也是我们之前 interpolation 要求渐变的原因, 防止你只学了个记忆功能. 如果是这样, 那么连续变化会有非常明显的不连续.

User study

Mode drop and mode collapse: Over datasets with known modes (e.g. a GMM or a labeled dataset), modes are computed as by measuring the distances of generated data to mode centers.

mode collapse: 假如对所有  $z$  都输出同一张 gt 呢? 我们并没有规定生成所有类型的图片, 假如网络只生成了一小部分类型 (drop), 或者只生成一张图 (mode collapse). 猫捉老鼠:g 总是只生成一张,d 总是说这是假的.(mode chasing) 根源:VAE 要求所有类型都为真, 而 GAN 则只需要保证“我生成的”图片真. 这也是 GAN 发展早期最棘手的问题之一.

Quantitative Measurement: FID

FID 实际上就是将 gt 和生成图片都转换到一个特征空间, 随后认为其符合正态, 度量其统计参数.

$$FID(r, g) = \|\mu_r - \mu_g\|_2^2 + \text{Tr} \left( \Sigma_r + \Sigma_g - 2 (\Sigma_r \Sigma_g)^{\frac{1}{2}} \right) \quad (137)$$

FID measure is sensitive to image distortions. From upper left to lower right: Gaussian noise, Gaussian blur, implanted black rectangles, swirled images, salt and pepper noise, and CelebA dataset contaminated by ImageNet images.



既关心真不真,  
也关心全不全.

GAN 在语义上的加减: 生成好的图片的网络, 其隐变量也必然学习了一些好的特征.latent space structure 也必然很好.

## A Condition Number

问题的条件数是数值分析中常见的概念, 是导数的一种推广. 简单来说, 就是对于输入发生微小变化时, 输出的变化程度的大小. 如果一个问题的条件数较小, 那么就称其是良置的 (well conditioned), 否则称为病态的 (ill conditioned).

考虑一个线性系统  $\mathbf{Ax} = \mathbf{b}$ , 那么若  $\det \mathbf{A} \neq 0$ , 则  $\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$ . 若输入变化  $\delta\mathbf{x}$ , 则输出变化为  $\mathbf{A}\delta\mathbf{x}$ . 考虑相对变化之比的上界:

## B Transformation in $\mathbb{R}^n$

在这一小节中我们简单介绍各种变换.

**等距变换**顾名思义, 就是保持距离的变换. 在其基本形式当中可以表示为平移加旋转, 这也是我们已经接触过的. 矩阵表示为

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0}^\top & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad (138)$$

其中的  $\mathbf{R}$  为旋转矩阵, 也是正交矩阵.

**相似变换**是指形状不变, 但可以改变大小和位置的变换. 直观地说就是等距变换加上缩放. 矩阵表示为

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{S}\mathbf{R} & \mathbf{t} \\ \mathbf{0}^\top & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}, \quad \mathbf{S} = \begin{bmatrix} s & 0 \\ 0 & s \end{bmatrix} \quad (139)$$

其中的  $\mathbf{S}$  表示缩放. 它保持边的长度比例和角度不变. 理解仿射变换的一种有益方法是将线性变换  $\mathbf{A}$  视作旋转和非均匀缩放的组合. 这是因为我们可以对  $\mathbf{A}$  进行 SVD:

$$\mathbf{A} = \mathbf{U}\Sigma\mathbf{V}^\top \quad (140)$$

由于  $\mathbf{U}, \mathbf{V}^\top$  都是正交矩阵, 可以视作旋转, 而  $\Sigma = \text{diag}\{\sigma_1, \sigma_2\}$  则可以视为非均匀缩放.

**仿射变换**则是一种保持了点, 线和平行性的变换. 它可以表示为一个线性变化加一次平移. 也就是

$$T(\mathbf{v}) = \mathbf{A}\mathbf{v} + \mathbf{t} \quad (141)$$

同样在齐次坐标下我们可以写作

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{A} & \mathbf{t} \\ \mathbf{0}^\top & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad (142)$$

只不过这里的矩阵  $\mathbf{A}$  可以代表任意的线性变换.

**射影变换**则只保留了将线映射成线, 而不保证平行性. 它表示为

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{A} & \mathbf{t} \\ \mathbf{v} & b \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad (143)$$

不难看出它包含了上述所有的变换种类, 添加了额外的自由度  $\mathbf{v}$ . 注意并不总能够通过缩放矩阵使得  $b = 1$ , 因为  $b$  可能为 0. 当  $b \neq 0$  时, 射影变换可以做如下分解:

$$\mathbf{H} = \mathbf{H}_S \mathbf{H}_A \mathbf{S}_P = \begin{bmatrix} s\mathbf{R} & \mathbf{t}/v \\ \mathbf{0}^\top & 1 \end{bmatrix} \begin{bmatrix} \mathbf{K} & \mathbf{0} \\ \mathbf{0}^\top & 1 \end{bmatrix} \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{v}^\top & b \end{bmatrix} = \begin{bmatrix} \mathbf{A} & \mathbf{t} \\ \mathbf{v} & b \end{bmatrix} \quad (144)$$

其中

$$\mathbf{A} = s\mathbf{RK} + \frac{1}{v}\mathbf{tv}^\top \quad (145)$$

且  $\mathbf{K}$  是满足  $\det \mathbf{K} = 1$  的归一化上三角矩阵. 如果限定  $s$  的符号, 它是唯一的, 只需做一次 QR 分解即可得到.

不难看出这个分解是将射影变换分解为相似变换  $\mathbf{H}_S$ , 仿射变换  $\mathbf{H}_A$  和一个有约束的透视变换(特殊的射影变换) $\mathbf{H}_P$  组成.

在射影变换下, 四个点的交比仍然保持不变. 四个点  $P_1, P_2, P_3, P_4$  的交比定义为

$$\text{cross ratio} = \frac{\|P_3 - P_1\| \|P_4 - P_2\|}{\|P_3 - P_2\| \|P_4 - P_1\|} \quad (146)$$

## C DOF and rank in essential matrix and fundamental matrix

所谓矩阵的自由度, 实际就是指矩阵中有多少个元素可以独立变化. 例如, 一个  $m \times n$  的矩阵在不加任何限制的情况下有  $mn$  个自由度, 而对于  $n$  阶上三角矩阵, 其自由度为  $n(n + 1)/2$ .

从三维的物体投影成二维的图像, 这个过程其实就是射影变换. 在射影空间当中的单应矩阵  $\mathbf{H}$ (可以理解为我们的投影变换矩阵) 天然少一个自由度, 因为自原点出发位于同一条线上的两个点, 在射影之后无法区分, 所以  $\mathbf{H} \sim \alpha\mathbf{H}$ .

测地线距离:geodesic distance.

## D QR Decomposition

矩阵的 QR 分解就是将矩阵分解为正交矩阵和上三角矩阵的乘积, 它可以对任意形状的矩阵进行. 常用的方法有 Gram–Schmidt process, Givens rotaitons 和 Householder reflections 等. 我们用最容易理解的施密特正交化方法来推导方阵的情形.

我们先将分解后的形式写出:

$$\begin{bmatrix} \mathbf{a}_1 & \mathbf{a}_2 & \cdots & \mathbf{a}_n \end{bmatrix} = \begin{bmatrix} \mathbf{e}_1 & \mathbf{e}_2 & \cdots & \mathbf{e}_n \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & \cdots & r_{1n} \\ 0 & r_{22} & \cdots & r_{2n} \\ 0 & 0 & \ddots & \vdots \\ 0 & 0 & \cdots & r_{nn} \end{bmatrix} \quad (147)$$

也就是满足

$$\mathbf{a}_i = \sum_{j=1}^i r_{ji} \mathbf{e}_j \quad (148)$$

由此可以定出

$$\begin{aligned} \mathbf{u}_1 &= \mathbf{a}_1, & \mathbf{e}_1 &= \frac{\mathbf{u}_1}{\|\mathbf{u}_1\|} \\ \mathbf{u}_2 &= \mathbf{a}_2 - \text{proj}_{\mathbf{u}_1} \mathbf{a}_2, & \mathbf{e}_2 &= \frac{\mathbf{u}_2}{\|\mathbf{u}_2\|} \\ &\vdots \\ \mathbf{u}_n &= \mathbf{a}_n - \sum_{j=1}^{n-1} \text{proj}_{\mathbf{u}_j} \mathbf{a}_n, & \mathbf{e}_n &= \frac{\mathbf{u}_n}{\|\mathbf{u}_n\|} \end{aligned} \quad (149)$$

以及

$$r_{ij} = \langle \mathbf{e}_i, \mathbf{a}_j \rangle \quad (150)$$

我们考虑一个方阵  $\mathbf{P}$ , 其副对角线上的元素为 1, 其余为 0. 不难验证  $\mathbf{P}\mathbf{P}^\top = \mathbf{I}, \mathbf{P} = \mathbf{P}^\top = \mathbf{P}^{-1}$ . 左乘矩阵  $\mathbf{P}$  会使得矩阵上下翻转, 右乘会使得矩阵左右翻转. 将一个上三角矩阵上下翻转后左右翻转, 即变为下三角矩阵. 记  $\tilde{\mathbf{A}} = \mathbf{PA}$ , 对  $\tilde{\mathbf{A}}^\top$  进行 QR 分解, 得到

$$\tilde{\mathbf{A}}^\top = \tilde{\mathbf{Q}} \tilde{\mathbf{R}} \quad (151)$$

由此得到

$$\mathbf{A} = \mathbf{P} \tilde{\mathbf{R}}^\top \tilde{\mathbf{Q}}^\top = (\mathbf{P} \tilde{\mathbf{R}}^\top \mathbf{P}) (\mathbf{P} \tilde{\mathbf{Q}}^\top) \stackrel{\text{def}}{=} \mathbf{R} \mathbf{Q} \quad (152)$$

同样的, 对于  $\mathbf{A}^\top$  进行 QR 分解就可以得到  $\mathbf{A}$  的 LQ 分解, 可以用同样的方法得到 QL 分解. 不过在使用时要注意:  $\mathbf{P}$  不一定是旋转矩阵.