L01 Overview

Visual Tasks 从 Low-Level 到 High-Level: Sensation → Processing → Perception → Cognition.

- **Data acquisition**: RGB 相机、深度相机、LiDAR 等. 得到照片、视频、点云、mesh 等.
- **Low-Level**: Processing 和特征提取. 图片降噪、去模糊、去水印、Edge/Corner 等.
- **Mid-Level**: 会对现实世界开始进行推断分析. 比如 3D 重建、全景照片合成.
- **High-Level**: 主要特征是会进行 semantic 级别的表征和解释. 比如场景内容理解、AR.

L02 Classic Methods

卷积定理: 卷积后的傅里叶变换 = 傅里叶变换的乘积.

导数定理: 可以预先计算 $\frac{d}{dx}g$, 则 $\frac{d}{dx}(f*g)$ 等价于 $f*(\frac{d}{dx}g)$.

Key-point Detect 的要求: Repeatability、Saliency、Accurate localization、Sufficient number.

对 Flat region、Edge、Corner 的定义: 考察将窗口移动 (u, v) 后窗口内强度的改变量. 特别地对于 Corner 而言, 向任何方向移动都应该有显著的强度变化.

$$E(u,v)=\sum_{x,y} w(x,y)\left[I(x+u,y+v)-I(x,y)\right]^2$$

对上述公式中的中括号内使用一阶 Taylor 近似, 得到关于 u 和 v 的二次型. 记二次型矩阵为 M, 则

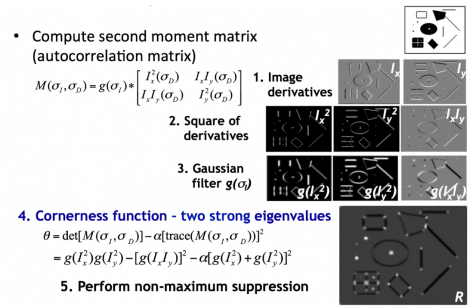
$$M=\begin{bmatrix}\sum I_x^2 & \sum I_xI_y\\ \sum I_xI_y & \sum I_y^2\end{bmatrix}=R^{-1}\begin{bmatrix}\lambda_1 & 0\\ 0 & \lambda_2\end{bmatrix}R \quad (\lambda_1\ge 0,\ \lambda_2\ge 0)$$

如果 λ1 和 λ2 都很大, 说明这是 Corner. 因为这样说明存在两个正交的方向, 窗口在这两个方向偏移后 E 都有显著变化!

不过考虑到正交对角化仍然具有一定的计算量, 使用快速近似:

$$\theta=\det(M)-\alpha\,\mathrm{trace}(M)^2=\lambda_1\lambda_2-\alpha(\lambda_1+\lambda_2)^2$$

步骤总结:

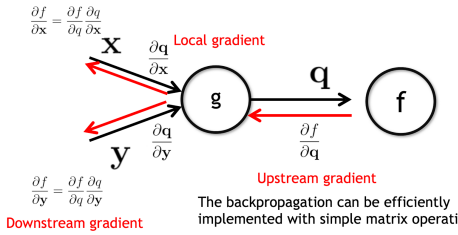


注意第 4 步中相当于为原图逐点计算了一个 θ. 如果用 numpy 实现, Ix Iy 直接是矩阵即可. 当然如果窗口大小不是 1x1, 那么 Ix Iy 要改成 ΣIx ΣIy, 用全 1/n 核卷积即可.

Harris Corner 的性质: 对平移和旋转 (Gauss or 1x1 才行吧…) 具有不变性, 但对缩放不具有不变性!

Line Fitting: LSE (法线方程 or SVD 技巧 (取 v_n)), RANSAC (Random Sample Consensus(共识)), RANSAC 抽 k 次全 fail 的概率 $(1-w^n)^k$. 选取 k 足够大即可. Hough 变换: 对于落入直线 y=mx+n 的一堆 (xi, yi), 为了求出 m 和 n, 以 m 和 n 为坐标轴绘制一堆直线 $y_i=mx_i+n$.

L03 Deep Learning I



Convolution layer: summary

Common settings:

K = (powers of 2, e.g. 32, 64, 128, 512
- F = 3, S = 1, P = 1
- F = 5, S = 1, P = 2
- F = 5, S = 2, P = ? (whatever fits)
- F = 1, S = 1, P = 0

Let's assume input is W_1 x H_1 x C
Conv layer needs 4 hyperparameters:
- Number of filters **K**
- The filter size **F**
- The stride **S**
- The zero padding **P**

This will produce an output of W_2 x H_2 x K
where:
- $W_2 = (W_1 - F + 2P)/S + 1$
- $H_2 = (H_1 - F + 2P)/S + 1$

Number of parameters: F^2CK and K biases

L04 Deep Learning II

General definition of equivariance:

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

Here A is an operation, T_A and S_A are their transformation function in the space of X and $\phi(X)$.

CNN: Parameter sharing => Equivariance to Translation

Data Preprocessing

否则试想如果所有 data 都是正的, 则梯度符号相同, 考虑 2 个参数, 则只能在第一三象限前进, 如果真实最优在第四象限, 则会 zig-zag!

Weight Initialization

· Xavier Initialization: $W=\frac{1}{\sqrt{Din}}N(0,1)$. 【推导: 希望 y=Wx 满足 Var(x)==Var(y), 假设 iid, 则 Var(y)=DinVar(x)Var(wi), 可知 wi 的方差必须是 1/Din. 】

· He Initialization: $W=\sqrt{\frac{2}{Din}}N(0,1)$. 这是对 ReLU 有效的. (对 ReLU 使用 Xavier 是不行的!)

对于卷积层而言, Din = filter_size² · n_input_channels.

SGD: 条件数过高时抖动、局部极值、mini-batch 的 noise.

解决: Momentum (一阶), 或者 Adam (一阶和二阶).

SGD

$$x_{t+1}=x_t-\alpha\nabla f(x_t)$$

```
while True:
  dx = compute_gradient(x)
  x -= learning_rate * dx
```

SGD+Momentum

$$v_{t+1}=\rho v_t+\nabla f(x_t)$$
$$x_{t+1}=x_t-\alpha v_{t+1}$$

```
vx = 0
while True:
  dx = compute_gradient(x)
  vx = rho * vx + dx
  x -= learning_rate * vx
```

- Build up "velocity" as a running mean of gradients
- Rho gives "friction"; typically rho=0.9 or 0.99

Adam:

```
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

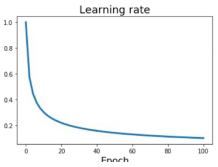
Bias correction

AdaGrad / RMSProp

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

Adam with beta1 = 0.9, beta2 = 0.999, and learning_rate = 1e-3 or 5e-4 is a great starting point for many models!

LR Schedule:



Step: Reduce learning rate at a few fixed points. E.g. for ResNets, multiply LR by 0.1 after epochs 30, 60, and 90.

Cosine: $\alpha_t=\frac{1}{2}\alpha_0(1+\cos(t\pi/T))$

Linear: $\alpha_t=\alpha_0(1-t/T)$

Inverse sqrt: $\alpha_t=\alpha_0/\sqrt{t}$

但初始 lr 太高可能会 loss 爆炸, 可以用 Linear Warmup.

训练 CNN 的两大难题 (train 时 underfit / test 时 overfit).

解决 underfit: BN (Batch Norm) 和 Skip link;

BN 的总结如下: 其中 N 是 Batch Size.

Input: $x: N\times D$

Learnable scale and shift parameters: $\gamma,\beta: D$

Learning $\gamma=\sigma$, $\beta=\mu$ will recover the identity function!

γ和β决定了这个正态函数在什么节点、以什么斜率被 ReLU 处理

$$\mu_j=\frac{1}{N}\sum_{i=1}^{N}x_{i,j}$$ Per-channel mean, shape is D

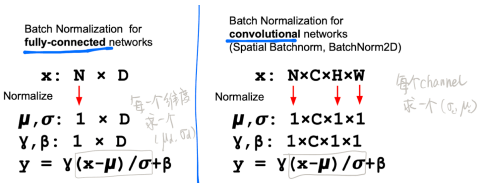
$$\sigma_j^2=\frac{1}{N}\sum_{i=1}^{N}(x_{i,j}-\mu_j)^2$$ Per-channel var, shape is D

$$\hat{x}_{i,j}=\frac{x_{i,j}-\mu_j}{\sqrt{\sigma_j^2+\varepsilon}}$$ Normalized x, Shape is N x D

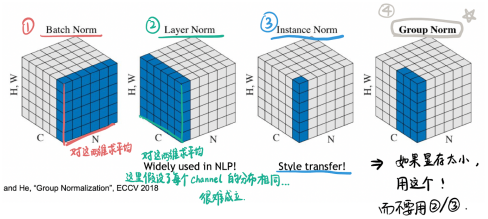
$$y_{i,j}=\gamma_j\hat{x}_{i,j}+\beta_j$$ Output, Shape is N x D

在测试时, σ 和 μ 可以用恒定值. 这个恒定值可以在训练阶段通过 ρμ+(1-ρ)μ 来学得, 仅用于预测.

对比 FC 和 Conv 的 BN:



BN 的变种:



Group Norm 当 Batch size 非常小的时候, 或者每个 Batch 的相关度太高 (correlated) 时, 会 outperform Batch Norm.

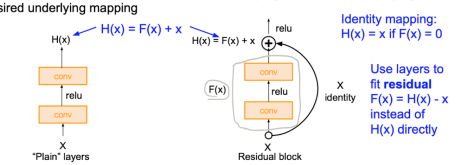
Batch Norm 一般插入在 FC/Conv 后, 但在 nonlinear 前. 比如介于 FC 和 tanh 之间.

· BN 的 Pros: 让深层网络非常容易训练, 改善了梯度流动, 允许更高的 lr, 更快收敛, 对初始化更 robust, 作为训练时的正则化, 测试时零开销 (可以和 Conv 融合).

· BN 的 Cons: 在 Train 和 Test 阶段具有不同的表现, 是 bug 的常见来源!

ResNet: 用到 Skip Link / Residual Link 的技巧

Solution: Network layers to fit a residual mapping instead of directly trying to fit a desired underlying mapping



Skip Link 可以让 Loss Landscape 变得平滑从而容易优化.

L05 Deep Learning III

Generalization gap: the difference between a model's performance on training data and its performance on unseen data drawn from the same distribution.

如果分布不同, 叫 Domain Gap. (e.g. 猫猫 train 野猫 test)

缓解 Generalization Gap 的思想: 平衡 data variability 和 model capacity.

· 从 data 角度: Data Augmentation、Batch Norm...

· 从 model 角度: Regularization、Dropout... (Slides 说这两种一般仅用于大型 FC Layer, 而上面两种则总可以使用.)

Softmax: sigmoid 的高维推广. 一般来说用到 exp(x), 不过一般形式是 exp(βx), 其中 β 如果是 ∞ 则退化为 argmax.

KL-Divergence:

$$D_{\mathrm{KL}}(P\parallel Q)=\sum_{x\in\mathcal{X}}P(x)\log\left(\frac{P(x)}{Q(x)}\right).$$

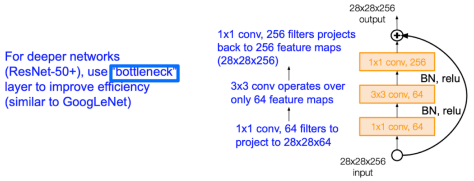
其中 P 被视为 reference / ground truth prob., 而 Q 则是预测的 prob. 这样上式可以推出 Cross-Entropy Loss (H(P, Q)).

$$D_{KL}(P\|Q)=\underbrace{-\sum_{x\in\mathcal{X}}P(x)\log Q(x)}_{H(P,Q)}-\underbrace{(-\sum_{x\in\mathcal{X}}P(x)\log P(x))}_{H(P)}$$

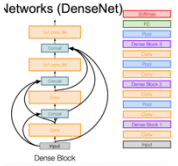
VGG-Net: Small Filter & Deeper Network

· Receptive Field 的概念: 3 个 3x3 conv filter 的 receptive field 是 7x7, 但相比一个 7x7 核不仅少了参数, 还增加了 non-Linear.

ResNet 的 **BottleNeck** 技巧



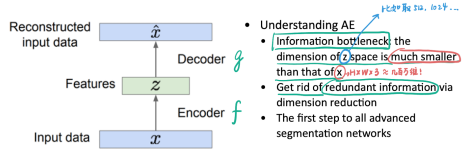
Dense-Net: Dense Block 内部的层之间全连接, 缓解梯度消失, 鼓励特征重用.



MobileNet 的所谓 Depth-Wise Conv: 假设有 C 个 channel, 那么现在每个 filter 不再是 nxnxC, 而是只是 nxnx1, 即每个 channel 对应一个 filter! 这样减轻了 C 倍计算量.

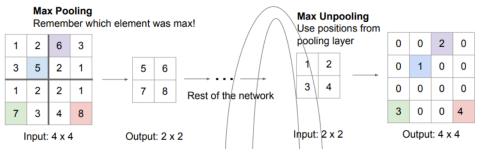
L06 Deep Learning IV

Auto Encoder

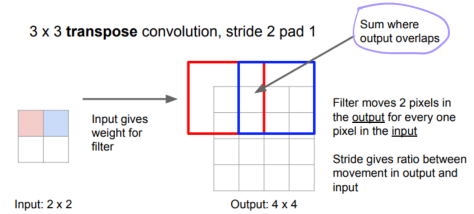


此处: Reconstruction loss $\triangleq \|\hat{x}-x\|^2$

Max unpooling:



Learnable unsampling: **Transposed Convolution:**



纯 Conv 的 Semantic Segmentation 中 Bottle-Neck 的意义: 很小的内存占用+极大的 receptive field (global context).

UNet: 轮廓信息难以 encode 进 bn, 使用 skip-link.

$$I(X)=\sum_{v\in V} X_v * Y_v .$$

$$U(X)=\sum_{v\in V}(X_v+Y_v-X_v*Y_v) .$$

· 评估 segmentation 的 metric: IoU (Intersection over Union). 以及 Soft IoU:

$$L_{IoU}=1-IoU=1-\frac{I(X)}{U(X)} .$$

L07 3D Vision

$$P'=\begin{bmatrix}\alpha & -\alpha\cot\theta & u_o & 0\\ 0 & \frac{\beta}{\sin\theta} & v_o & 0\\ 0 & 0 & 1 & 0\end{bmatrix}\begin{bmatrix}x\\ y\\ z\\ 1\end{bmatrix}$$

f = focal length
u_o, v_o = offset (Or C_x, C_y)
α, β → non-square pixels
θ = skew angle

K has 5 degrees of freedom!

综合 Intrinsic 和 Extrinsic 的变换, 记整体变换为 M, 则有 $M_{3\times4}=K_{3\times3}\,[R_{3\times3}\ T_{3\times1}]$:

记 M 的行向量为 m_1, m_2, m_3, 记世界坐标系下点坐标为 P_w, 则映射后得到的 2 维点为 $(\frac{m_1P_w}{m_3P_w}, \frac{m_2P_w}{m_3P_w})$.

Weak Projective Camera: 对 Projective 的简化. 变换矩阵的对比如下图所示:

Projective (perspective) Weak perspective

$$M=K[R\quad T]=\begin{bmatrix}\mathbf{A}_{2\times3} & \mathbf{b}_{2\times1}\\ \mathbf{v}_{1\times3} & 1\end{bmatrix}\rightarrow M=\begin{bmatrix}\mathbf{A} & \mathbf{b}\\ \mathbf{0} & 1\end{bmatrix}$$

Calibration problem

一对已知的 pi Pi 对可以给出 2 行方程. 由于 M 的自由度为 5 + 3 + 3 = 11, 所以需要 6 对.

[Eq. 1] $\begin{bmatrix}u_i\\ v_i\end{bmatrix}=\begin{bmatrix}\mathbf{m}_1\mathbf{P}_i\\ \mathbf{m}_3\mathbf{P}_i\\ \mathbf{m}_2\mathbf{P}_i\\ \mathbf{m}_3\mathbf{P}_i\end{bmatrix}$

一对 correspondence 可得到2个方程.

$$u_i=\frac{\mathbf{m}_1\mathbf{P}_i}{\mathbf{m}_3\mathbf{P}_i}\rightarrow u_i(\mathbf{m}_3\mathbf{P}_i)=\mathbf{m}_1\mathbf{P}_i\rightarrow u_i(\mathbf{m}_3\mathbf{P}_i)-\mathbf{m}_1\mathbf{P}_i=0$$

$$v_i=\frac{\mathbf{m}_2\mathbf{P}_i}{\mathbf{m}_3\mathbf{P}_i}\rightarrow v_i(\mathbf{m}_3\mathbf{P}_i)=\mathbf{m}_2\mathbf{P}_i\rightarrow v_i(\mathbf{m}_3\mathbf{P}_i)-\mathbf{m}_2\mathbf{P}_i=0$$

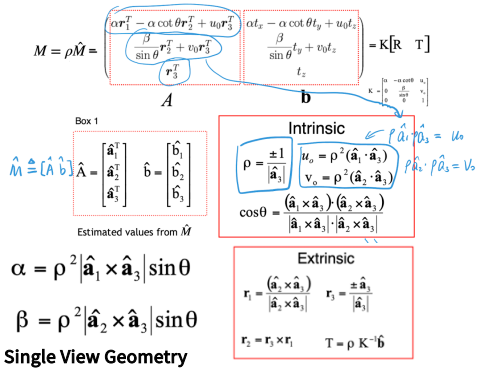
$$\begin{cases}-u_1(\mathbf{m}_3\mathbf{P}_1)+\mathbf{m}_1\mathbf{P}_1=0\\ -v_1(\mathbf{m}_3\mathbf{P}_1)+\mathbf{m}_2\mathbf{P}_1=0\\ \vdots\\ -u_n(\mathbf{m}_3\mathbf{P}_n)+\mathbf{m}_1\mathbf{P}_n=0\\ -v_n(\mathbf{m}_3\mathbf{P}_n)+\mathbf{m}_2\mathbf{P}_n=0\end{cases}\rightarrow \underbrace{\mathbf{P}}_{\text{known}}\ \underbrace{\mathbf{m}}_{\text{unknown}}=0 \quad [\text{Eq. 4}]$$

Homogenous linear system

$$\mathbf{P}\overset{\text{def}}{=}\begin{pmatrix}\mathbf{P}_1^T & \mathbf{0}^T & -u_1\mathbf{P}_1^T\\ \mathbf{0}^T & \mathbf{P}_1^T & -v_1\mathbf{P}_1^T\\ \vdots & & \\ \mathbf{P}_n^T & \mathbf{0}^T & -u_n\mathbf{P}_n^T\\ \mathbf{0}^T & \mathbf{P}_n^T & -u_n\mathbf{P}_n^T\end{pmatrix}_{2n\times12}^{1\times4}\qquad \boldsymbol{m}\overset{\text{def}}{=}\begin{pmatrix}\mathbf{m}_1^T\\ \mathbf{m}_2^T\\ \mathbf{m}_3^T\end{pmatrix}_{12\times1}^{4\times1}$$

考虑到存在平凡解 m = 0, 将问题转化为约束 |m| = 1 下最小化 |Pm| 的问题. 这等价于 SVD 中求最后一个 v_n! 即 M = v_n.

但首先这样的 M 是单位化的, 真实的 M 还会相差某个 ρ 倍. 另外问题是如何反解出每个参数 α、β、θ、u0、v0、r1、r2、r3、T. 结论为:



$$\alpha=\rho^2|\hat{\mathbf{a}}_1\times\hat{\mathbf{a}}_3|\sin\theta$$

$$\beta=\rho^2|\hat{\mathbf{a}}_2\times\hat{\mathbf{a}}_3|\sin\theta$$

Single View Geometry

· 两条直线 l, l' 的交点 x = l × l'.

· 2D「**无穷远点**」: x_∞ [x1 x2 0]. 「**无穷远线**」: l_∞ = [0 0 1], 显然所有 x_∞ 都位于 l_∞ 上. 3D 中的「**无穷远点**」: x_∞ = [x1 x2 x3 0]. Image plane 上的「**Vanishing Point**」v: x_∞ 经过 M 投影到 Image plane 上的点. 如果不考虑 Extrinsics, 则有 v = Kd, d = 单位化的 K 逆 v. 其中 d 是方向向量. 关于某个平面的「**Vanishing Line**」: 同一个平面上的 Vanishing points 都会落在这上面.

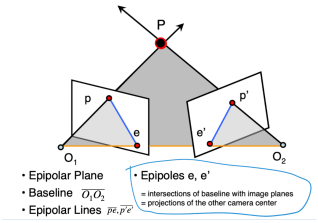
· 两个 Vanishing Points v1 v2 对应的平行线的夹角:

$$\cos\theta=\frac{\mathbf{v}_1^T\omega\,\mathbf{v}_2}{\sqrt{\mathbf{v}_1^T\omega\,\mathbf{v}_1}\sqrt{\mathbf{v}_2^T\omega\,\mathbf{v}_2}}\quad\left(\text{其中}\ \omega=(KK^T)^{-1}\right)$$

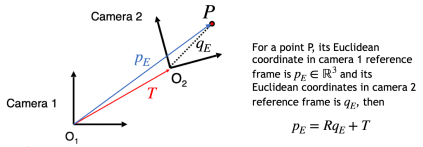
[Eq. 28]

如果已知 θ = 90°, 则 $v_1^T\omega v_2=0$. 如果找到多对这样的 v1 v2, 就可以解出 ω, 进而使用 Cholesky 分解得到 K.

Epipolar Geometry



以下: World Space 就是 O_1 的 Camera Space. $M=K[I\ 0]$, 而 $M'=K'[R^T\ -R^TT]$. 其中 T 和 R 的定义和示意图如下:



For a point P_i its Euclidean coordinate in camera 1 reference frame is $p_E \in \mathbb{R}^3$ and its Euclidean coordinates in camera 2 reference frame is q_E, then

$$p_E=Rq_E+T$$

T = $\mathbf{O_2}$ in the camera 1 reference system

R is the rotation matrix such that a vector **p'** in the camera 2 is equal to **R p'** in camera 1.

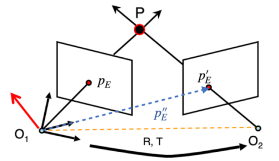
Epipolar Constraints: 对于空间中的点 P, 其在两个 Image Plane 上成像的 p 和 p' 之间的约束关系.

两种描述: Essential Matrix 和 Fundamental Matrix. 二者有关系式 $F=K^{-1^T}EK'^{-1}$. 前者使用 3D Euclidean 坐标系, 给出了 $p_E^TEp_E'=0$ 的约束; 后者使用 2D Image Plane 上的 Homogeneous 坐标系, 给出了 $p^TFp'=0$ 的约束.

叉乘的矩阵表示:

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

E 的推导: 如图, p_E 是 O1 下的 3D 坐标; p'_E 是 O2 下的 3D 坐标. 而 p'_E 在 O1 下的坐标为 $p''_E=Rp_E'+T$.



- $p_E \in \mathbb{R}^3$ is the Euclidean coordinate in the camera 1 reference frame
- $p'_E \in \mathbb{R}^3$ is the Euclidean coordinate in the camera 2 reference frame
- Then, the Euclidean coordinate of p' in the camera 1 reference frame, $p''_E=Rp'_E+T$

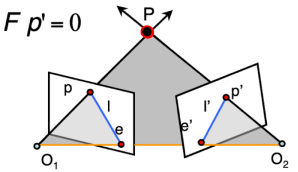
为了求 Epipolar Plane, 可以求它的法向量 n. 只需要让 O_1O_2 和 p_E'' 叉乘即可, 即 $n=T\times(Rp'_E+T)=T\times Rp'_E$.

写成矩阵的形式即 $n=[T_\times]Rp'_E$. 再注意到 O_1p_E 垂直于 n, 因此 $p_E\cdot[T_\times]Rp'_E=0$, 即 $p_E^T([T_\times]R)p'_E=0$. 括号里即 E.

F 的推导: 设 p_E 在 Image1 下的 Homogeneous 坐标为 p, p'_E 在 Image2 下的 Homogeneous 坐标为 p'. 则有 $p=Kp_E$ 和 $p'=K'p'_E$. 把这两个代入到刚才的 $p_E^T([T_\times]R)p'_E=0$, 得到 $(K^{-1}p)^T([T_\times]R)(K'^{-1}p')=0$, 即 $p^T(K^{-1})^T([T_\times]R)(K'^{-1})p'=0$. 记中间那一坨为 F, 则 $p^TFp'=0$.

F 的性质:

$$p^T\cdot F\,p'=0$$



- l = F p' is the epipolar line associated with p'
- l'= F^Tp is the epipolar line associated with p
- F e' = 0 and F^Te = 0
- F is 3x3 matrix; 7 DOF
- F is singular (rank two)

Stereo System

$$\frac{u-u'}{f}=\frac{B}{z}\qquad u-u'=\frac{B\cdot f}{z}=\text{disparity}\qquad [\text{Eq. 1}]$$

Note: Disparity is inversely proportional to depth

Arranged by PkuCuipy @ Github

L08 3D Sensors

· **ToF (Time of Flight)**：分为 iToF (indirect, i.e. phase) 和 dToF (direct, i.e. time). iPhone FaceID 是 **Structured light**, 一个投射特定 Pattern，一个识别; iPad LiDAR 是 dToF.

· **3D Representation**: Regular form (多角度图片、深度图、体素) v.s. Irregular form (点云、Mesh、F(x)=0).

· **Point Cloud**：它不是 surface representation，而是在 surface 上 sampling (Uniform / Farthest Point).

· **Point Cloud 间距离**: Chamfer Distance & Earth Mover Distance. 前者是逐点找最近的对方点，对采样不敏感；后者要求「一一对应」的意义下最小，对采样的随机性敏感.

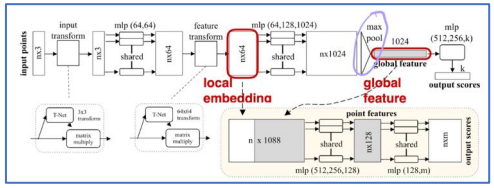
· **SDF**: Signed Distance Field. Marching Cude 算法.

L09 3D Deep Learning

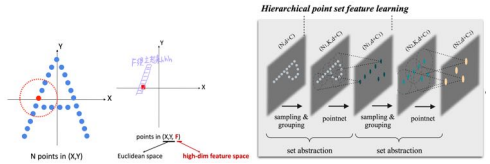
· **3D CNN**: 4D kernel；计算量太大；体素的稀疏问题（椅子这个才占用 2.41% 确实有点反直觉!)

· **Sparse Conv**：中心点非 0 的地方才做卷积.

· **PointNet**: Local Embeddings 和 Global Feature；所谓 Critical Point (真正对 Global feat 有贡献的点) 问题在于只能学到要么「单点」要么「全局」，没有「局部 context」.



· **PointNet++:** 对多个局部区域分别使用 PointNet，然后得到更少但带有更高维度 feature 的点.

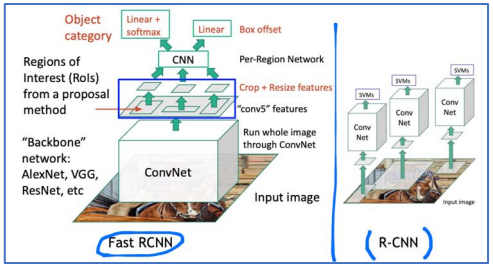


L10 Detection & Segmentation

· **Object Detection:** 单个物体: Classification (类别) + Regression (位置, 4 ToF). 不定数目物体：需要后处理，只靠 nn 做不到. 最 naive 想法: Sliding-Window (计算代价太高).

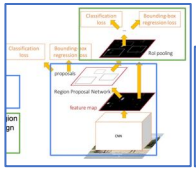
· **R-CNN**: 所谓 **Region Proposal**: 通过某种方法先提取出一些 **RoI** (Regions of Interest, ~2k 个)，然后每个区域 Reshape (插值) 到 224×224，使用 ConvNet，最后用 SVM 给出分类，并用回归给出相对 RoI 的 (dx, dy, h, w). 但还是太慢了，因为一张图就要进行 2000 次计算!

· **Fast R-CNN**: 先对整张图 CNN 得到 Feature Map，**然后对原图进行 RoI**，然后把 RoI 对应到 Feature Map 的相应区域，对这些区域进行 Crop 和 Resize.



· **Faster R-CNN**: 除改用 **RPN (Region Proposal Network)** 以外均与 Fast R-CNN 一致. RPN 的原理：对每个像素位置都取 K 个 "Anchor Box"，用 ConvNet 预测每个 AB 是否是一个 object 以及 Bounding Box (x, y, h, w) of 这个 object.

· 称之为**两阶段的 Detector**: 【第一阶段】: Backbone CNN + RPN; 【第二阶段】: 对每个 RoI，进行 Feature Crop、预测类别、预测 bounding box.



Jointly train with 4 losses:
1. RPN classify object / not object
2. RPN regress box coordinates
3. Final classification score (object classes)
4. Final box coordinates

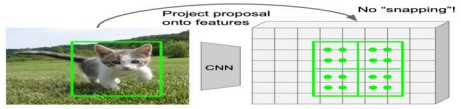
· 对 Proposal 进行 **NMS**. 算法描述: D 是结果集, B 初始为所有 Proposals. 从 B 中挑出 Confidence 最大的那个 prop 加入到 D，然后移除所有 B 中和 prop 的 IoU 超过阈值的 proposals. 重复操作直到 B 成为空集.

· **Evaluation**: 给定 IoU 后的 **PR 曲线**和 **mAP**: AP = Average(Precision(Recall)). **十一点法**: Recall 取 [0, 0.1, 0.2, ..., 1.0]. mAP 的所谓 "m" 可以省略，仅仅表示【如果有多个类别，再对这些类别的 AP 求 mean】.

· **Instance Segmentation:** 分为 Top-down 和 Bottom-up 两类方法，前者就是说先找 BBox，然后再预测 Mask；后者是先 Gather 相似的像素，然后给这个集合预测类别标签.

· **Mask R-CNN**: 一种 Top-down 方法，单纯是在 R-CNN 的最后再加上一个 Mask Prediction Network.

· **RoI Align:** RoI Pool 的问题在于，"Snap" 到整数网格的行为会导致系统误差！改为用 **RoI Align**: 使用双线性插值.



3D Detection & Segmentation: BBox -> Frustum.

L11 Pose & Motion

· **Euler Angle** 把旋转矩阵 R 表示为 Rz(α) · Ry(β) · Rx(γ).

· **Axis Angle** 表示为转轴 (axis) **e** 和角度 (angle) θ.

$$K = [e]_\times = \begin{bmatrix} 0 & -e_z & e_y \\ e_z & 0 & -e_x \\ -e_y & e_x & 0 \end{bmatrix}$$
$$R = I + (\sin\theta)\mathbf{K} + (1-\cos\theta)\mathbf{K}^2$$

· **Quaternion:** 表示为 q = w + xi + yj + zk，这里 w 称为实部，向量 v=(x, y, z) 称为虚部. 满足 $i^2 = j^2 = k^2 = -1$，反称性 ij = -ji, 以及 ij = k, jk = i, ki = j.

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

设
$$\begin{aligned} \mathbf{q}_1 &= w_1 + x_1\mathbf{i} + y_1\mathbf{j} + z_1\mathbf{k} \\ \mathbf{q}_2 &= w_2 + x_2\mathbf{i} + y_2\mathbf{j} + z_2\mathbf{k} \end{aligned}$$
则
$$\begin{aligned} \mathbf{q}_1 * \mathbf{q}_2 &= (w_1w_2 - x_1x_2 - y_1y_2 - z_1z_2) \\ &+ (w_1x_2 + x_1w_2 + y_1z_2 - z_1y_2)\mathbf{i} \\ &+ (w_1y_2 - x_1z_2 + y_1w_2 + z_1x_2)\mathbf{j} \\ &+ (w_1z_2 + x_1y_2 - y_1x_2 + z_1w_2)\mathbf{k} \end{aligned}$$

· **共轭**: q' = w - xi - yj - zk

· **模长**: ||q|| = sqrt(q · q') = sqrt($w^2 + x^2 + y^2 + z^2$)

· **Unit Quaternion**: 满足模长为 1，从而 q 逆 == q'.

· 四元数具有结合律，但不具有交换律!

· q = w + xi + yj + zk 可以**表示为向量** (s, **v**)，其中 s = w, **v** = [x, y, z]. (类似于复数 a+bi 表示为 (a, b)). 于是上述乘法法则可以写为 (s1, **v1**) · (s2, **v2**) = (s1s2 − **v1** · **v2**, s1**v2** + s2**v1** + **v1**×**v2**).

· 之所以拆成这样写，是为了方便从 Axis-Angle 表示转换到 Quaternion 表示: Axis-angle 表示法的 (**e**, θ) 对应于四元数表示法的 q=(s, **v**)，其中 s = cos(θ/2), **v** = **e** sin(θ/2).

· 如何用 q 旋转向量 **x**? 将 **x** 看成实部为 0 的四元数 (0, **x**)，则 q**x**q' 的虚部对应于旋转后的向量（且实部一定是 0).

· **多次旋转的组合**: 注意到 q2(q1**x**q1')q2' = (q2q1)**x**(q1'q2'). 因此 q2q1 可以表征这两个旋转的总体旋转.

· 如何预测一个旋转？【方式 1】直接对旋转的某种表示法进行 Regression. 【方式 2】预测 Camera Coordinates 下的每个点的 Model Coordinates (所以前提是这个物体的 CAD 模型是已知的)，然后 Fit 一个 Rotation.

· Fit 的过程对应于 **Orthogonal Procrustes Problem:**

The **orthogonal Procrustes problem** is a matrix approximation problem that can be stated as follows: for $\mathbf{M} \in \mathbb{R}^{n\times p}$ and $\mathbf{N} \in \mathbb{R}^{n\times p}$, solve

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

这个问题存在解析解:

If we have the SVD: $\mathbf{M}^T\mathbf{N} = \mathbf{U}\mathbf{D}\mathbf{V}^T$, then $\widehat{\mathbf{A}} = \mathbf{V}\mathbf{U}^T$.

【证明思路】Frobenius 范数可以表为 Frob(A) = tr(A'A)，再利用 tr(AB) = tr(BA)，以及正交矩阵的对角元素最大为 1.

· **Instance-Level Pose Estimation**: Instance-Level 即这些物品都是已知的，有 CAD 模型，Pose 也是基于 CAD 定义的. 输入是 RGB / RGBD，预测 6D Pose. 代表模型: **PoseCNN**. 它的 Loss 定义比较神奇，是先把预测出的 q 转为 R，然后对所有点 Apply 这个 R，然后以点云之间的差异作为 Loss. 对于普通物体 (如汽油瓶) 和对称物体 (如可乐瓶) 具有不同的 Loss:

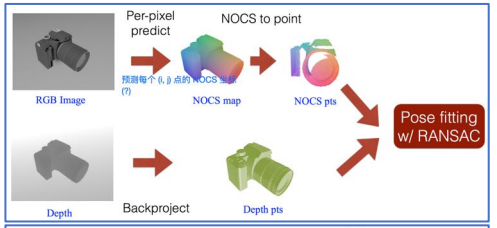
Pose Loss (non-symmetric)
$$\mathrm{PLoss}(\tilde{\mathbf{q}}, \mathbf{q}) = \frac{1}{2m}\sum_{\mathbf{x}\in\mathcal{M}} \|R(\tilde{\mathbf{q}})\mathbf{x} - R(\mathbf{q})\mathbf{x}\|^2$$

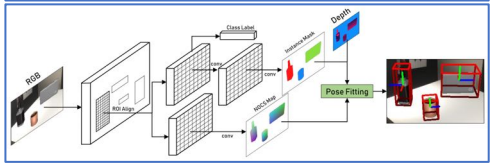
Shape-Match Loss for symmetric objects (symmetric)
$$\mathrm{SLoss}(\tilde{\mathbf{q}}, \mathbf{q}) = \frac{1}{2m}\sum_{\mathbf{x}_1\in\mathcal{M}} \min_{\mathbf{x}_2\in\mathcal{M}} \|R(\tilde{\mathbf{q}})\mathbf{x}_1 - R(\mathbf{q})\mathbf{x}_2\|^2$$

局限性：需已知 CAD，可控环境中较有用，难以泛化一般物体.

· **Category-Level**: 即可以泛化到一类物体. 不再需要 CAD model. **NOCS**: Normalized Object Coordinate Space: 第一步是对齐物体朝向、第二步中心归零、第三步缩放到单位方块内.

· Pose 预测的流程：首先要求 Input 是 RGBD 的! 然后: RGB 图逐像素预测每个 (i, j) 的 NOCS 坐标，这样得到点云 1; 然后对 Depth 图使用 BackProjection 算法得到点云 2. 对这两个点云进行 Pose Fitting.





Motion

· 仅讨论 **Optical Flow** 方法估计 Motion.

· 三个**基本假设**: 亮度一致、小运动量、局部一致性. 根据亮度不变假设，有 $I(x, y, t-1) = I(x + u(x, y), y + v(x, y), t)$. 注意到等式右边可以根据 Taylor 公式展成等式左边再加上一些项，于是「再加上的那些项」就应该为 0. 即 $[Ix, Iy, It] \bullet [u, v, 1] = 0$，其中前面三个是亮度 I 的梯度分别在 x, y, t 方向的分量. 为了可解和稳定，使用 5×5 的窗口列出 25 行等式，然后用最小二乘法求解. 这就是 **Lucas-Kanade** 算法.

$$\begin{bmatrix} I_x(\mathrm{p}_1) & I_y(\mathrm{p}_1) \\ I_x(\mathrm{p}_2) & I_y(\mathrm{p}_2) \\ \vdots & \vdots \\ I_x(\mathrm{p}_{25}) & I_y(\mathrm{p}_{25}) \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} I_t(\mathrm{p}_1) \\ I_t(\mathrm{p}_2) \\ \vdots \\ I_t(\mathrm{p}_{25}) \end{bmatrix} \quad \underset{25\times2}{A} \quad \underset{2\times1}{d} = \underset{25\times1}{b}$$

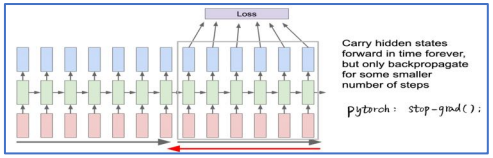
Least squares solution for d given by $(A^TA)\ d = A^Tb$

$$\begin{bmatrix} \sum I_xI_x & \sum I_xI_y \\ \sum I_xI_y & \sum I_yI_y \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} \sum I_xI_t \\ \sum I_yI_t \end{bmatrix}$$

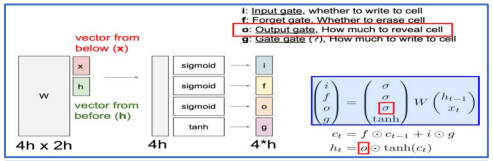
When is This Solvable?
- $\mathbf{A}^T\mathbf{A}$ should be invertible
- $\mathbf{A}^T\mathbf{A}$ should not be too small due to noise
  - eigenvalues λ_1 and λ_2 of $\mathbf{A}^T\mathbf{A}$ should not be too small
- $\mathbf{A}^T\mathbf{A}$ should be well-conditioned
  - λ_1/λ_2 should not be too large (λ_1 = larger eigenvalue)

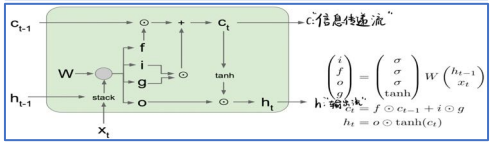
L12: Temporal Data Analysis

· 带截断的 **RNN** Backpropagation.

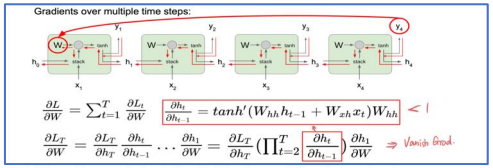


· **LSTM**:





i 用 sigmoid 激活，取值 (0, 1)，起到 "how much" 的作用; g 用 tanh 激活，取值 (-1, +1)，起到 "what (to write)" 作用.



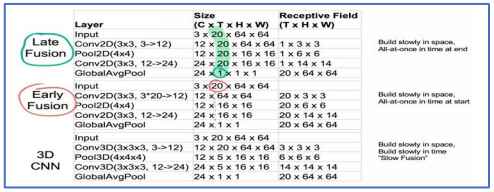
RNNs allow a lot of flexibility in architecture design
Vanilla RNNs are simple but don't work very well
Common to use LSTM or GRU: their additive interactions improve gradient flow
Backward flow of gradients in RNN can explode or vanish. Exploding is controlled with gradient clipping. Vanishing is controlled with additive interactions (LSTM)

L13 Video Analysis

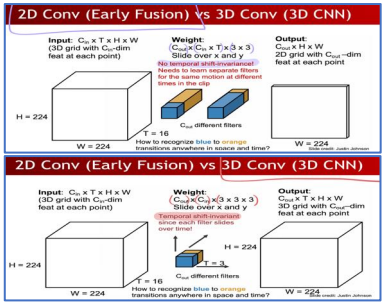
· **Late Fusion**: 首把每一帧对应的 2D 图片经过 CNN 映射为一个高维 Feature，然后把所有帧的 HighDimFeat 组合到一起，喂给 MLP 给出最后的预测. 这里「组合」可以是 Concat，也可以是 AvgPool. 【Late Fusion 的问题在于：很难比较 low level 图片中的 motion 在帧与帧之间的差别】

· **Early Fusion**: 把 T×3×H×W 的视频，看成一个 H×W×(T×3) 的具有 3T 个通道的「图片」，然后对它使用 **2D CNN**. 或者也可以把视频看成 (H×W×T)×3 的 (3 为通道数)，然后使用 **3D CNN**. 2D CNN 的问题在于只用一个 Layer 来处理所有时间可能不太行. 考虑用 3D CNN 在时间维度「慢慢地」获取时间维度的帧与帧之间的信息.

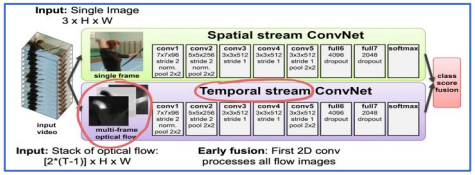
· 三者的总结. 注意 build 是指 Receptive Field 的 build.



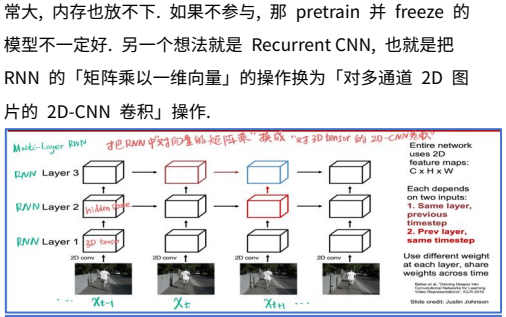
· Early Fusion 中的 2D CNN (左) 与 3D CNN (右) 的对比:



· **Two-Stream Fusion**: 即 Spatial 和 Temporal 的融合到一起用于训练，前者就是视频本身，后者比如 optical flow. (因此也有说法叫 Appearance + Motion).



· **Recurrent CNN**: 把 CNN 和 RNN 结合的 naive 想法是对每一帧先用 CNN 等输出一维向量，然后作为 x 喂给 RNN. 这里首先 CNN 是否要参与梯度反向传播？如果要，那么开销会非常大，内存也放不下. 如果不参与，那 pretrain 并 freeze 的模型不一定好. 另一个想法就是 Recurrent CNN，也就是把 RNN 的「矩阵乘以一维向量」的操作换为「对多通道 2D 图片的 2D-CNN 卷积」操作.

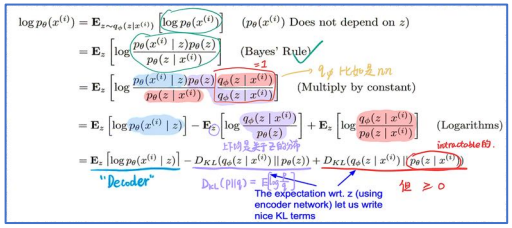


L14 Generative Model

· **Explicit** density vs **Implicit** density. 区别在于能否输出一个 probability，还是只能 sample 但给不出 prob. 前者包括可精确计算的 (**tractable**) 的 **PixelRNN / PixelCNN** 和只能近似计算概率密度的 VAE. 后者包括 GAN.

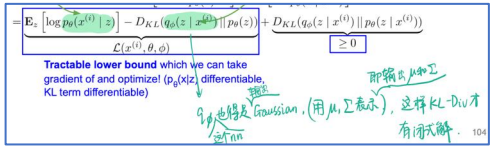
· PixelRNN/CNN 的好处在于可以显式给出密度，且易于优化，并且效果蛮好的 (和 VAE 相比). 缺点是二者都很慢! (Pixel RNN 在训练和推断都很慢, Pixel CNN 在训练时可以一定程度并行加速，但推断时必须串行因此仍然很慢!)

· **VAE** 对图片 x 的概率密度的建模如下:



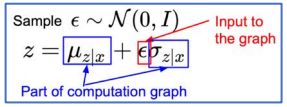
· 第一项由 Decoder nn 的采样给出估计，越大表示 decoder 重建得越对；第二项是两个高斯分布之间的 KL 散度，具有解析解，它越小表示 z 的 latent distribution 越接近 N(0, 1)；最后一项是 intractable 的，但永远 ≥ 0.

· 前两项放在一起称为 **E**vidence **L**ower **Bo**und (ELBO). 这里 ELBO 其实仍是 intractable 的，但这里我们选择对第一项使用 Monte Carlo 进行估计使之 tractable. (可证现在的 MC 的方差比较小可接受，而一开始用 MC 的话 Var 很大，不可用)



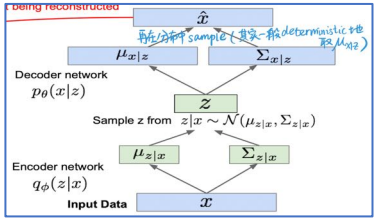
· 在训练时，输入数据 x 首先经过 Encoder 网络 qΦ(z|x) 给出由 μ 和 Σ 表征的正态分布，这样就可以计算第二项的散度 (这个散度越小越好)；然后在这个分布进行多次 z 的采样，求出相应的第一项那个期望的估值 (这一项越大越好).

· 对 z 的「采样」操作可以转写为可导的形式:

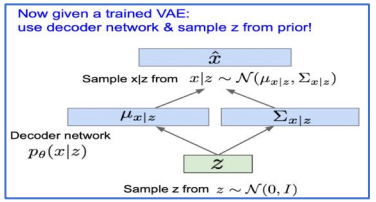


· VAE 优点：可解释的 Latent Space；学出来的 q(z|x) 可以给出特征表示，对于其他任务可能有帮助. 缺点：只能优化一个 LowerBound，生成图比较 blurry.

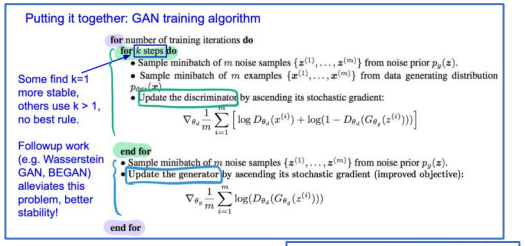
· **训练时的 VAE**:



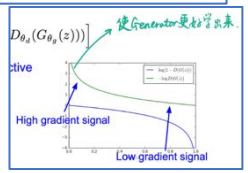
· **Generation Time 的 VAE:**



· **GAN**



· 这里 Generator 的目标函数之所以不是 -log(1 - D(G(z))) 是因为这样的话 0 附近的梯度太小了，训练最开始时时 Generator 网络会寸步难行.



· Mode drop (只生成一类人)/collapse(只生成一张图)...

· **FID**: 用把图片编码成向量的网络 (CNN / InceptionNet 等) 分别作用于生成图片集合和真实图片集合，然后看成两个高斯分布，然后按照如下公式计算 FID: (第一项关注「真不真」，第二项关注「全不全，即考虑了发生 Mode drop 的情形」)

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

Pros:
- Beautiful, state-of-the-art samples!

Cons:
- Trickier / more unstable to train
- Can't solve inference queries such as p(x), p(z|x)

VAE	GAN
• Blurry	• More realistic
• Full coverage of the data	• Only penalize fake and therefore can suffer from mode collapse
• Support approximate inference	• Can't infer probability

Arranged by PkuCuipy @ Github