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.

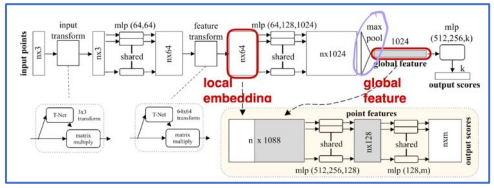
· **SDF**: Signed Distance Field. Marching Cude 算法.

L09 3D Deep Learning

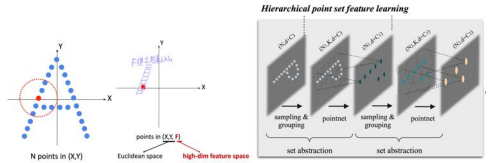
· **3D CNN**: 4D kernel；计算量太大；体素的稀疏问题（椅子这个才占用 2.41% 确实有点反直觉!)

· **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.

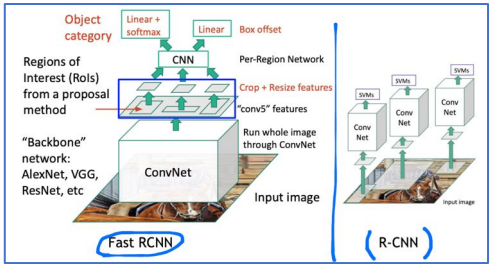


L10 Detection & Segmentation

· **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).

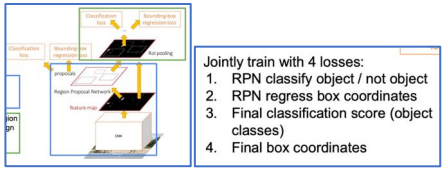
· **R-CNN**: 所谓 **Region Proposal**: 通过某种方法先提取出一些 **RoI** (Regions of Interest, ~2k 个)，然后每个区域 Reshape (插值) 到 224×224，使用 ConvNet，最后用 SVM 给出分类，并用回归给出相对 RoI 的 (dx, dy, h, w). 但还是太慢了，因为一张图就要进行 2000 次计算!

· **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.



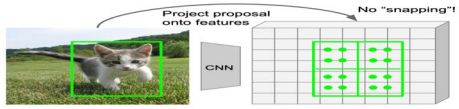
· 对 Proposal 进行 **NMS**. 算法描述: D 是结果集, B 初始为所有 Proposals. 从 B 中挑出 Confidence 最大的那个 prop 加入到 D，然后移除所有 B 中和 prop 的 IoU 超过阈值的 proposals. 重复操作直到 B 成为空集.

· **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.

---

1. $p(z)$ 已知, 是一个高维标准正态分布
2. $p_\theta(x|z)$ 函数族已知, 一个均值和方差来源于神经网络的正态分布
3. $q_\phi(z|x)$ 函数族已知, 一个均值和方差来源于神经网络的正态分布

当然 $p(z)$ 是高维标准正态分布这个假设不是很合理, 这也是 VAE 的一个 limitation, 但是在一定程度上是可以接受的

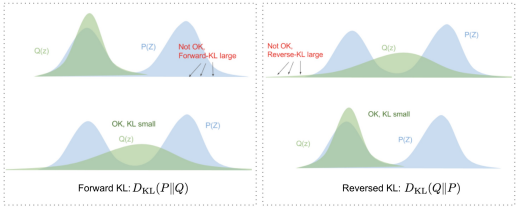
$$\begin{aligned}
\mathrm{D}_{\mathrm{KL}}\left(q_\phi(\boldsymbol{z}\mid\boldsymbol{x})\|p_\theta(\boldsymbol{z}\mid\boldsymbol{x})\right) &= \int q_\phi(\boldsymbol{z}\mid\boldsymbol{x})\log\frac{q_\theta(\boldsymbol{z}\mid\boldsymbol{x})}{p_\theta(\boldsymbol{z}\mid\boldsymbol{x})}\mathrm{d}\boldsymbol{z} \\
&= \int q_\phi(\boldsymbol{z}\mid\boldsymbol{x})\log\frac{q_\theta(\boldsymbol{z}\mid\boldsymbol{x})p_\theta(\boldsymbol{x})}{p_\theta(\boldsymbol{z},\boldsymbol{x})}\mathrm{d}\boldsymbol{z} \\
&= \int q_\phi(\boldsymbol{z}\mid\boldsymbol{x})\left(\log\left(p_\theta(\boldsymbol{x})\right)+\log\frac{q_\theta(\boldsymbol{z}\mid\boldsymbol{x})}{p_\theta(\boldsymbol{z},\boldsymbol{x})}\right)\mathrm{d}\boldsymbol{z} \\
&= \log\left(p_\theta(\boldsymbol{x})\right)+\int q_\phi(\boldsymbol{z}\mid\boldsymbol{x})\log\frac{q_\theta(\boldsymbol{z}\mid\boldsymbol{x})}{p_\theta(\boldsymbol{z},\boldsymbol{x})}\mathrm{d}\boldsymbol{z} \\
&= \log\left(p_\theta(\boldsymbol{x})\right)+\int q_\phi(\boldsymbol{z}\mid\boldsymbol{x})\log\frac{q_\phi(\boldsymbol{z}\mid\boldsymbol{x})}{p_\theta(\boldsymbol{x}\mid\mathrm{d}\boldsymbol{z})p_\theta(\boldsymbol{z})}\mathrm{d}\boldsymbol{z} \\
&= \log\left(p_\theta(\boldsymbol{x})\right)+E_{\boldsymbol{z}\sim q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left(\log\frac{q_\phi(\boldsymbol{z}\mid\boldsymbol{x})}{p_\theta(\boldsymbol{z})}-\log\left(p_\theta(\boldsymbol{x}\mid\boldsymbol{z})\right)\right) \\
&= \log\left(p_\theta(\boldsymbol{x})\right)+\mathrm{D}_{\mathrm{KL}}\left(q_\phi(\boldsymbol{z}\mid\boldsymbol{x})\|p_\theta(\boldsymbol{z})\right)-\mathbb{E}_{\boldsymbol{z}\sim q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left(\log\left(p_\theta(\boldsymbol{x}\mid\boldsymbol{z})\right)\right)
\end{aligned} \tag{110}$$

将上式重写成

$$\log\left(p_\theta(\boldsymbol{x})\right)-\mathrm{D}_{\mathrm{KL}}\left(q_\phi(\boldsymbol{z}\mid\boldsymbol{x})\|p_\theta(\boldsymbol{z}\mid\boldsymbol{x})\right)=-\mathrm{D}_{\mathrm{KL}}\left(q_\phi(\boldsymbol{z}\mid\boldsymbol{x})\|p_\theta(\boldsymbol{z})\right)+\mathbb{E}_{\boldsymbol{z}\sim q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left(\log\left(p_\theta(\boldsymbol{x}\mid\boldsymbol{z})\right)\right) \tag{111}$$

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 $q_\phi(\boldsymbol{z}\mid\boldsymbol{x})$ 和 $p(\boldsymbol{z})$ 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

$$\mathcal{L}_{\theta,\phi}=-RHS=\mathrm{D}_{\mathrm{KL}}\left(q_\phi(\boldsymbol{z}\mid\boldsymbol{x})\|p_\theta(\boldsymbol{z})\right)-\mathbb{E}_{\boldsymbol{z}\sim q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left(\log\left(p_\theta(\boldsymbol{x}\mid\boldsymbol{z})\right)\right) \tag{112}$$



Non-Saturating Loss functions

判别器梯度上升:

$$\max_{\theta_d}\left[\mathbb{E}_{x\sim p_{data}}\log D_{\theta_d}(x)+\mathbb{E}_{z\sim p(z)}\log(1-D_{\theta_d}(G_{\theta_g}(z)))\right] \tag{121}$$

生成器梯度下降:

$$\min_{\theta_g}\mathbb{E}_{z\sim p(z)}\log(1-D_{\theta_d}(G_{\theta_g}(z))) \tag{122}$$

但是优化这个生成器目标函数效果不好, 因为 $\log(1-x)$ 这个函数在接近 0 的地方梯度相对较小, 所以会导致刚开始优化的时候梯度很小, 优化效果不好.

所以可以进行凹凸性反转, 对生成器的目标函数进行改进, 使得生成器的目标函数在刚开始优化的时候梯度更大, 优化效果更好. 于是就有了下面的目标函数:

生成器梯度上升:

$$\max_{\theta_g}\mathbb{E}_{z\sim p(z)}\log(D_{\theta_d}(G_{\theta_g}(z))) \tag{123}$$

除此之外, 点云的距离度量也成为一个问题, 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. [21]

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

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[22]. 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.[23]

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

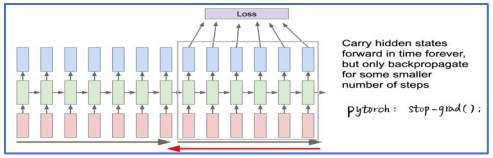
CD 对于取样情况不太敏感, 而 EMD 则比较敏感. 比如同样对于 Stanford bunny, 如果一个点云多集中在头部, 另一个比较均匀, 则 CD 变化不大而 EMD 变换显著. 由于点云是 surface+sampling, 因此如果对于取样有要求, 应该使用 EMD.

2. NMS. 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. [27]

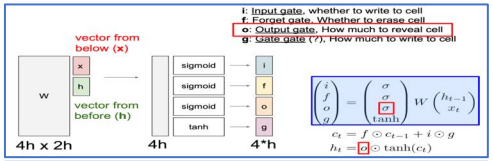
[27] 举例来说, 假设猫分类的 RoI 中概率最大者圈住了一只猫的绝大多数, 因而以 90% 的 confidence 认为是猫, 则其他与它 IoU 大于 0.5 的 RoI 很可能只圈住了这只猫的半边身子, 所以把它们都去掉. 而 IoU 较小的可能是其他的猫. 当然, 这样做也存在一些问题, 比如有一个和它非常接近的 RoI 因为某种原因识别为狗, 那么这样做就不能去除这样的 RoI, 因为此处 NMS 只对同类的 RoI 进行操作. 后来也有工作同时预测 RoI 与 IoU(即" 预测的预测"), 并证明这样做效果更优.

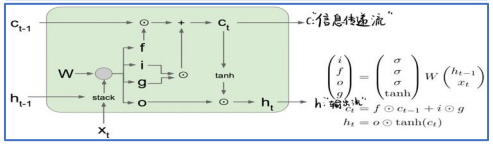
L12: Temporal Data Analysis

· 带截断的 **RNN** Backpropagation.

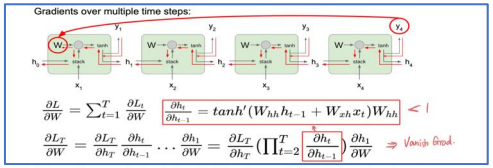


· **LSTM**:





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.



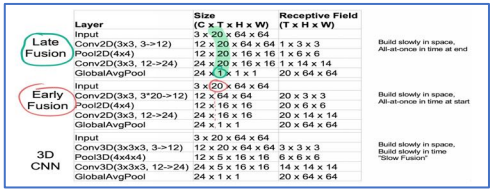
RNNs allow a lot of flexibility in architecture design
Vanilla RNNs are simple but don't work very well
Common to use LSTM or GRU: their additive interactions improve gradient flow
Backward flow of gradients in RNN can explode or vanish. Exploding is controlled with gradient clipping. Vanishing is controlled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

L13 Video Analysis

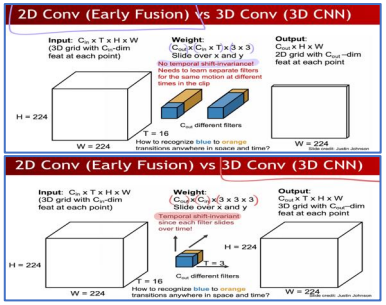
· **Late Fusion**: 首把每一帧对应的 2D 图片经过 CNN 映射为一个高维 Feature，然后把所有帧的 HighDimFeat 组合到一起，喂给 MLP 给出最后的预测. 这里「组合」可以是 Concat，也可以是 AvgPool. 【Late Fusion 的问题在于：很难比较 low level 图片中的 motion 在帧与帧之间的差别】

· **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.

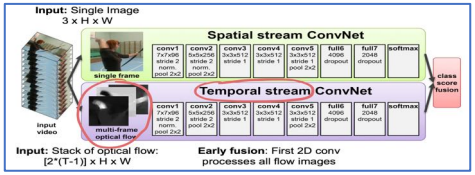
· 三者的总结. 注意 build 是指 Receptive Field 的 build.



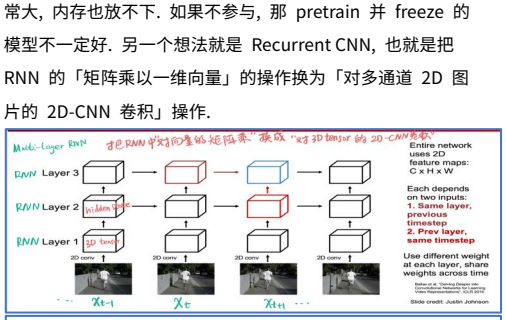
· Early Fusion 中的 2D CNN (左) 与 3D CNN (右) 的对比:



· **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.

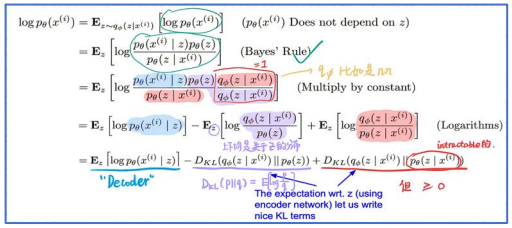


L14 Generative Model

· **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.

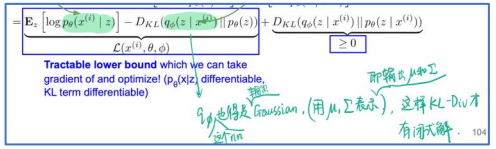
· PixelRNN/CNN 的好处在于可以显式给出密度，且易于优化，并且效果蛮好的 (和 VAE 相比). 缺点是二者都很慢! (Pixel RNN 在训练和推断都很慢, Pixel CNN 在训练时可以一定程度并行加速，但推断时必须串行因此仍然很慢!)

· **VAE** 对图片 x 的概率密度的建模如下:



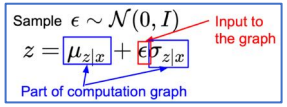
· 第一项由 Decoder nn 的采样给出估计，越大表示 decoder 重建得越对；第二项是两个高斯分布之间的 KL 散度，具有解析解，它越小表示 z 的 latent distribution 越接近 N(0, 1)；最后一项是 intractable 的，但永远 ⩾ 0.

· 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)



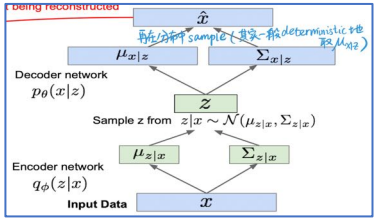
· 在训练时，输入数据 x 首先经过 Encoder 网络 qΦ(z|x) 给出由 μ 和 Σ 表征的正态分布，这样就可以计算第二项的散度 (这个散度越小越好)；然后在这个分布进行多次 z 的采样，求出相应的第一项那个期望的估值 (这一项越大越好).

· 对 z 的「采样」操作可以转写为可导的形式:

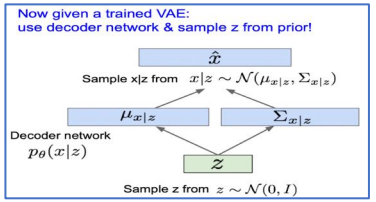


· VAE 优点：可解释的 Latent Space；学出来的 q(z|x) 可以给出特征表示，对于其他任务可能有帮助. 缺点：只能优化一个 LowerBound，生成图比较 blurry.

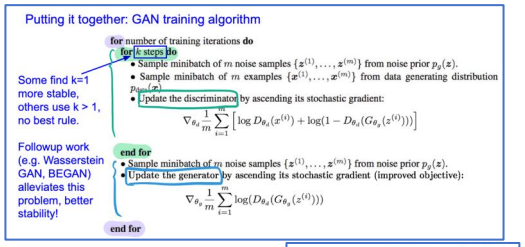
· **训练时的 VAE**:



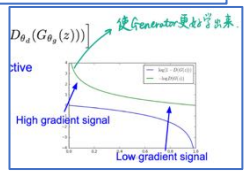
· **Generation Time 的 VAE:**



· **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