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.

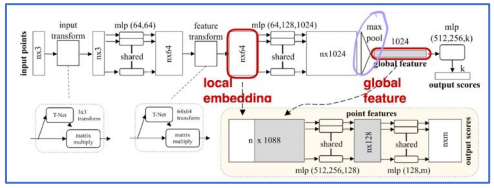
· **SDF**: Signed Distance Field. Marching Cude 算法.

L09 3D Deep Learning

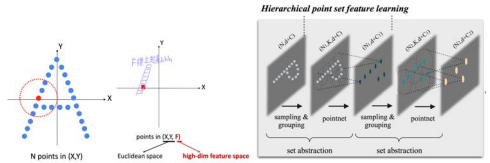
· **3D CNN**: 4D kernel；计算量太大；体素的稀疏问题（椅子这个才占用 2.41% 确实有点反直觉!)

· **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.

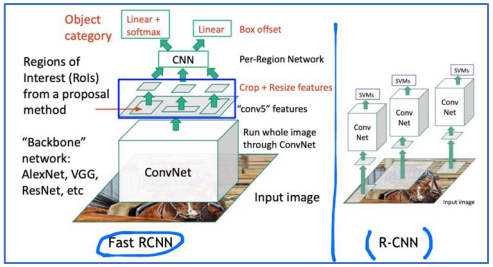


L10 Detection & Segmentation

· **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).

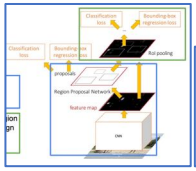
· **R-CNN**: 所谓 **Region Proposal**: 通过某种方法先提取出一些 **RoI** (Regions of Interest, ~2k 个)，然后每个区域 Reshape (插值) 到 224×224，使用 ConvNet，最后用 SVM 给出分类，并用回归给出相对 RoI 的 (dx, dy, h, w). 但还是太慢了，因为一张图就要进行 2000 次计算!

· **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

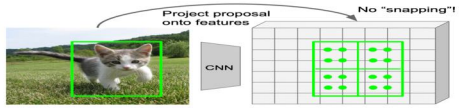
· 对 Proposal 进行 **NMS**. 算法描述: D 是结果集, B 初始为所有 Proposals. 从 B 中挑出 Confidence 最大的那个 prop 加入到 D，然后移除所有 B 中和 prop 的 IoU 超过阈值的 proposals. 重复操作直到 B 成为空集.

· **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 $\mathbf{q}_1 = w_1 + x_1\mathbf{i} + y_1\mathbf{j} + z_1\mathbf{k}$, $\mathbf{q}_2 = w_2 + x_2\mathbf{i} + y_2\mathbf{j} + z_2\mathbf{k}$

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)

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

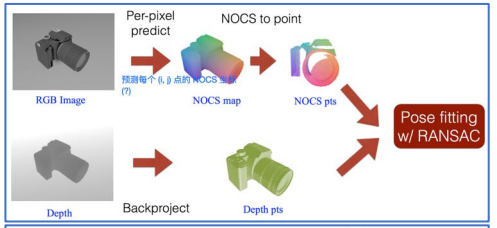
Shape-Match Loss for symmetric objects (symmetric)

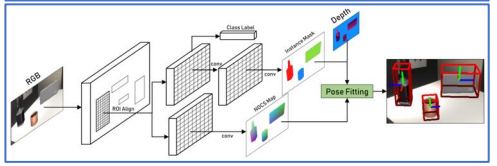
$$\mathrm{SLoss}(\tilde{\mathbf{q}}, \mathbf{q}) = \frac{1}{2m}\sum_{\mathbf{x}_1\in\mathcal{M}} \min_{\mathbf{x}_2\in\mathcal{M}} \|R(\tilde{\mathbf{q}})\mathbf{x}_1 - R(\mathbf{q})\mathbf{x}_2\|^2$$

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.

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

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

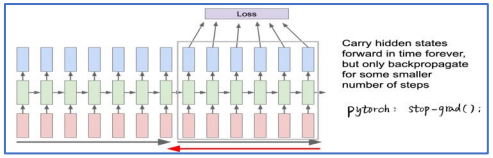
$$\begin{bmatrix} \sum I_xI_x & \sum I_xI_y \\ \sum I_xI_y & \sum I_yI_y \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} \sum I_xI_t \\ \sum I_yI_t \end{bmatrix}$$

When is This Solvable?

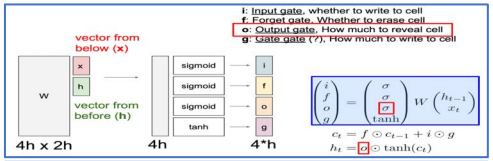
- $\mathbf{A}^T\mathbf{A}$ should be invertible
- $\mathbf{A}^T\mathbf{A}$ should not be too small due to noise
  - eigenvalues λ_1 and λ_2 of $\mathbf{A}^T\mathbf{A}$ should not be too small
- $\mathbf{A}^T\mathbf{A}$ should be well-conditioned
  - λ_1/λ_2 should not be too large (λ_1 = larger eigenvalue)

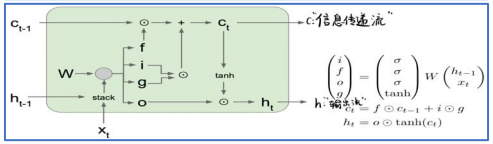
L12: Temporal Data Analysis

· 带截断的 **RNN** Backpropagation.

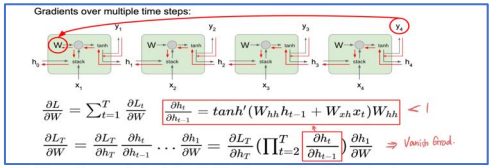


· **LSTM**:





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.



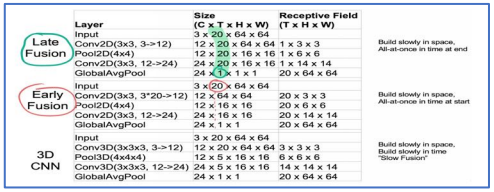
RNNs allow a lot of flexibility in architecture design
Vanilla RNNs are simple but don't work very well
Common to use LSTM or GRU: their additive interactions improve gradient flow
Backward flow of gradients in RNN can explode or vanish. Exploding is controlled with gradient clipping. Vanishing is controlled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

L13 Video Analysis

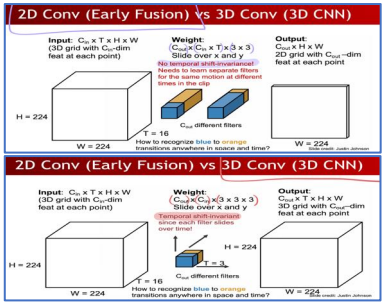
· **Late Fusion**: 首把每一帧对应的 2D 图片经过 CNN 映射为一个高维 Feature，然后把所有帧的 HighDimFeat 组合到一起，喂给 MLP 给出最后的预测. 这里「组合」可以是 Concat，也可以是 AvgPool. 【Late Fusion 的问题在于：很难比较 low level 图片中的 motion 在帧与帧之间的差别】

· **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.

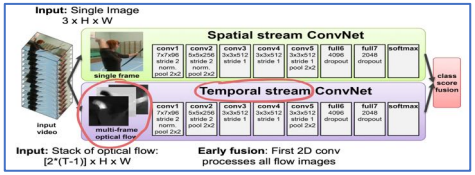
· 三者的总结. 注意 build 是指 Receptive Field 的 build.



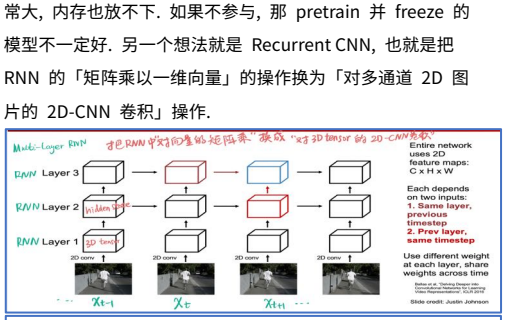
· Early Fusion 中的 2D CNN (左) 与 3D CNN (右) 的对比:



· **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.

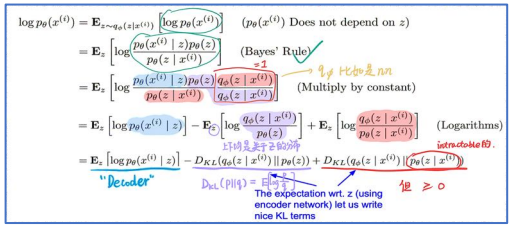


L14 Generative Model

· **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.

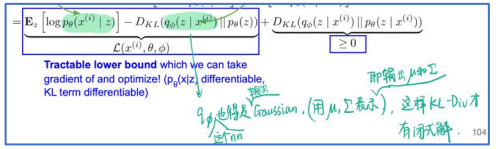
· PixelRNN/CNN 的好处在于可以显式给出密度，且易于优化，并且效果蛮好的 (和 VAE 相比). 缺点是二者都很慢! (Pixel RNN 在训练和推断都很慢, Pixel CNN 在训练时可以一定程度并行加速，但推断时必须串行因此仍然很慢!)

· **VAE** 对图片 x 的概率密度的建模如下:



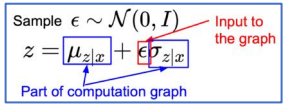
· 第一项由 Decoder nn 的采样给出估计，越大表示 decoder 重建得越对；第二项是两个高斯分布之间的 KL 散度，具有解析解，它越小表示 z 的 latent distribution 越接近 N(0, 1)；最后一项是 intractable 的，但永远 ⩾ 0.

· 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)



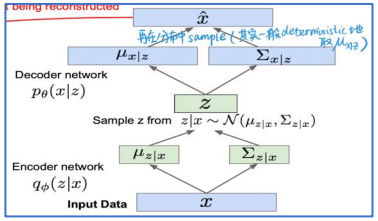
· 在训练时，输入数据 x 首先经过 Encoder 网络 qΦ(z|x) 给出由 μ 和 Σ 表征的正态分布，这样就可以计算第二项的散度 (这个散度越小越好)；然后在这个分布进行多次 z 的采样，求出相应的第一项那个期望的估值 (这一项越大越好).

· 对 z 的「采样」操作可以转写为可导的形式:

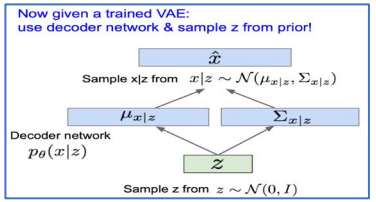


· VAE 优点：可解释的 Latent Space；学出来的 q(z|x) 可以给出特征表示，对于其他任务可能有帮助. 缺点：只能优化一个 LowerBound，生成图比较 blurry.

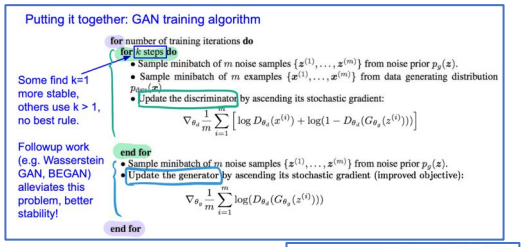
· **训练时的 VAE**:



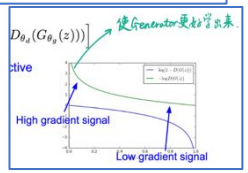
· **Generation Time 的 VAE:**



· **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