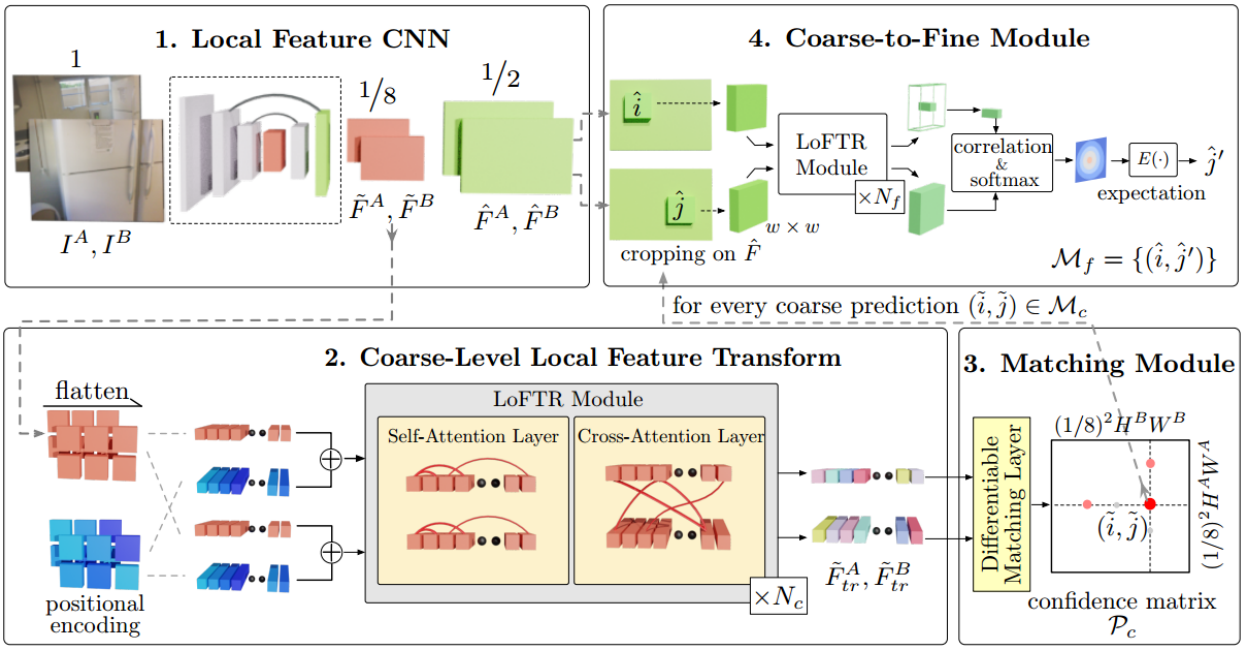
LoFTR是将transformer引入到Feature matching的里程碑. 下面我将用尽量通俗的方式讲解他的原理和网络结构. 首先整体的网络结构如下图一所示:



图一: LoFTR总体网络结构

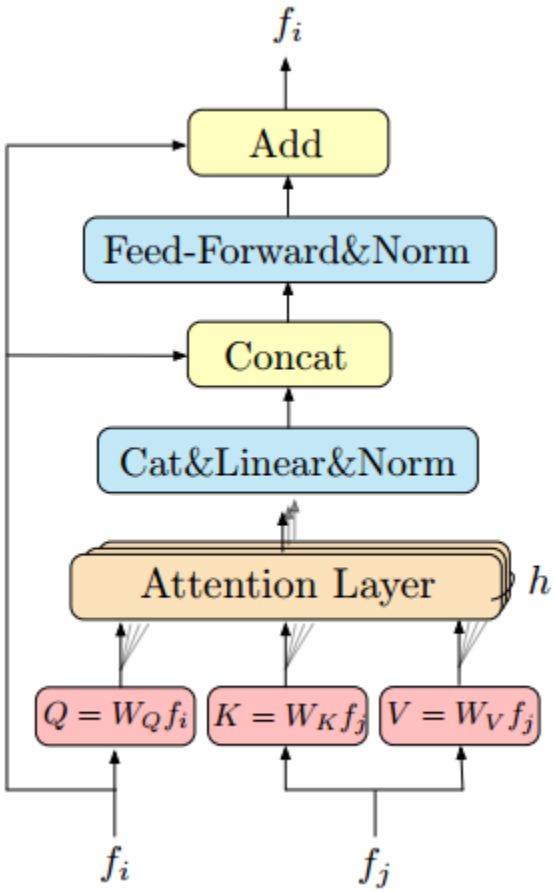
LoFTR的backbone是经典的Resnet + Feature Pyramid Network. 作者认为CNN网络拥有translation equivariance和locality两个特性非常适合局部特征点的提取. 简单地说, translation equivariance就是物体在两张图片中做了细小的平移, 最后经过ResNet输出的特征图变化很小或者没有变化, 这点主要是通过CNN网络中的pooling layer实现的. Locality是CNN网络的特性, ResNet输出的不同层的特征图虽然感受野不一样, 但是都是局部的感受野, 专注于提取局部信息, 相对地, Transformer是处理全局信息, 对局部信息提取不如CNN.

输入两张图片 I^A 和 I^B, 经过Resnet+FPN处理后, 作者提取第四层的特征图 $\tilde{F}^A$ 和 $\tilde{F}^B$ (尺寸为原图的⅛, 256 channels)进行coarse-level local feature transform. 这个coarse-level local feature transform 其实就是带有self-attention和cross-attention的transformer. 首先作者把每张特征图按像素展开变成一行, 然后给每个像素(每个像素为256 element vector)加上不同的positional encoding去记录特征图里每个像素的位置. 然后把带有positional encoding的特征图像素输入LoFTR Module. 进入LoFTR module后, 输入的两个特征图会进入self-attention layer各自进行self-attention, 紧接着self-attention的两个输出会进入cross-attention layer互相进行cross-attention(注意A对于B的cross attention不同于B对于A的cross attention). 这个self-attention & cross attention的过程会进行 N_c 次. 无论是

self-attention layer还是cross-attention layer都不会改变输入的dimension, 也就是说整个过程中, 输入输出dimension始终一致. 最终LoFTR module输出 $\tilde{F}^A_{tr}$ 和 $\tilde{F}^B_{tr}$, 对应两个原始的特征图 $\tilde{F}^A$ 和 $\tilde{F}^B$.

什么是self-attention和cross attention呢?

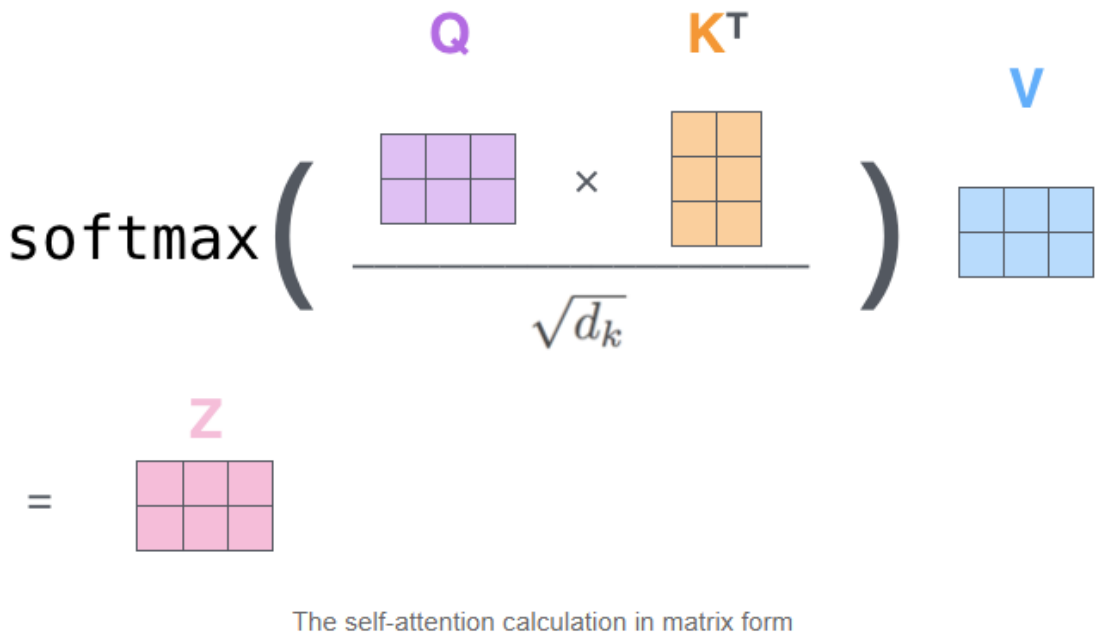
我们来讲解一下attention机制吧, self-attention机制的transformer最早出现于Vaswani的paper "Attention Is All You Need" [1]. Cross-attention和self-attention原理基本一致, 下面图二展示了self-attention和cross-attention的基础过程:



图二: self-attention和cross-attention机制的基础过程

对于self-attention来说, f_i 和 f_j 都来自于同一个feature map(经过positional encoding), 可以是 $\tilde{F}^A$ 或者 $\tilde{F}^B$. 输入了feature map后, self-attention layer里面会有三个矩阵, query matrix(W_Q), key matrix(W_K)以及value matrix(W_V), 这三个矩阵乘以输入的feature map就会分别生成query, key和value(注意query, key, value也是矩阵, 并不是标量). 得到的query乘以key transpose就可以得到score. 我们先对score除以根号dk(dk为value每一行的长度), 然后对score的每一行取softmax, 就能得到权

重矩阵. 然后用这个权重矩阵乘以value就可以得到attention layer的输出Z. 具体计算过程如下图三所示 :



The self-attention calculation in matrix form

图三: self-attention具体计算过程, Q代表query, K代表key, V代表value, Z代表输出

上面为一个self-attention head的计算过程. 实际的implementation中我们需要用到多个attention-head, 也就是multi-head attention. 我们可以把这个过程类比成CNN里用多个kernel来进行convolution, 每个kernel提取不同的信息. 同理每个attention head的query matrix(W_Q), key matrix(W_K)以及value matrix(W_V)拥有不同的值, 分别用来提取不同的信息. 每个attention-head都重复着图三的计算过程, 唯一的区别是query, key, value的长度要除以attention-head的数量. 比如只有一个attention-head的时候, query, key和value的dimension都是N*256(N对应多少个像素点), 现在有8个attention-head, 那么query, key和value的长度就变成N*32(通过改变W_Q, W_K, W_V的dimension来改变Q, K, V的dimension), 然后最终的输出就是把所有Z连接起来(concatenate)恢复原来长度(比如8个N*32的Z连接起来恢复成8*256). 这个最终的输出会经过一个feed-forward network, 也就是两层的MLP, 到此一个self-attention layer才算完成. 需要注意的是两张图片的特征图都需要各自进行self-attention, 得到的这两个输出会作为cross-attention layer的输入.

有一点值得一提的是, 为了防止gradient vanish, 我们需要在attention layer里面用到类似于ResNet的skip connection, 实现residual learning. 在vanilla transformer中, 我们通常会把输入的feature map和attention-head的输出相加(记住attention-head不改变输入的dimension), 然后normalize. 但是这里作

者直接把attention-head输出和feature map输入拼接(concatenate)在一起. 比如说attention-head输出的dimension是N*256, 输入的feature map也是N*256, 那么concatenate后就变成了N*512的矩阵. 为了不改变最终输出的dimension, 作者利用MLP层, 输入N*512, 重新输出N*256的矩阵, 保证输出传给下一个attention layer的时候dimension不变.

Cross-attention计算过程和self-attention一样, 唯一不同的是图二的输入 f_i 和 f_j 不再来自同一张特征图, 而是来自两张特征图, $\tilde{F}^A$和$\tilde{F}^B$, 或者$\tilde{F}^B$和$\tilde{F}^A$(经过self-attention layer的特征图). 正如我前面提过, $\tilde{F}^A$对$\tilde{F}^B$做cross-attention得到的结果和$\tilde{F}^B$对$\tilde{F}^A$做cross-attention的结果是不一样的, 也就是说不符合交换律. 当我们需要A图对于B图的cross-attention时, f_i 对应 $\tilde{F}^A$, f_j 对应 $\tilde{F}^B$, 反之亦然. 每个cross-attention layer我们都需要做A对于B的cross-attention以及B对于A的cross-attention. 这两个cross-attention会生成两个输出, 作为下一个self-attention layer的输入, 依次循环. 这个self-attention layer + cross-attention layer的过程会循环Nc次. 不过每次循环的self-attention layer和cross-attention layer的参数是独立的, 为了理解我们其实可以把他当做是Nc个完全独自的self-attention layer+cross-attention layer, 也就是Nc个独立的LoFTR module. 作者的implementation中Nc取值为4.

经过Nc个LoFTR module后, 我们得到了两个输出$\tilde{F}^A_{tr}$和$\tilde{F}^B_{tr}$. 然后我们通过下面equation(1)得到score matrix S:

$$\mathcal{S}(i,j) = \frac{1}{\tau} \cdot \langle \tilde{F}^A_{tr}(i), \tilde{F}^B_{tr}(j) \rangle \tag{1}$$

Score matrix的第i行第j列可以看做是A特征图第i个像素和B特征图第j个像素的点积(乘以一个normalization系数). 我们知道点积经常被用来求cosine distance, 因此点积可以一定程度上反应两个像素点的相似度. 得到Score matrix后我们通过dual softmax去得到probability matrix P:

$$\mathcal{P}_c(i,j) = \mathrm{softmax}\left(\mathcal{S}(i,\cdot)\right)_j \cdot \mathrm{softmax}\left(\mathcal{S}(\cdot,j)\right)_i \tag{2}$$

对于S的(i,j), 我们首先对第i行做softmax, 然后对第j列做softmax, 然后分别取(i,j)在两次softmax的值相乘, 就能得到probability matrix里的(i,j). Probability matrix的第i行我们可以理解为A特征图的第i个像素和B特征图的每一个像素的匹配概率. 相似地, Probability matrix的第j列我们可以理解为B特征图的第j个像素和A特征图的每个像素的匹配概率.

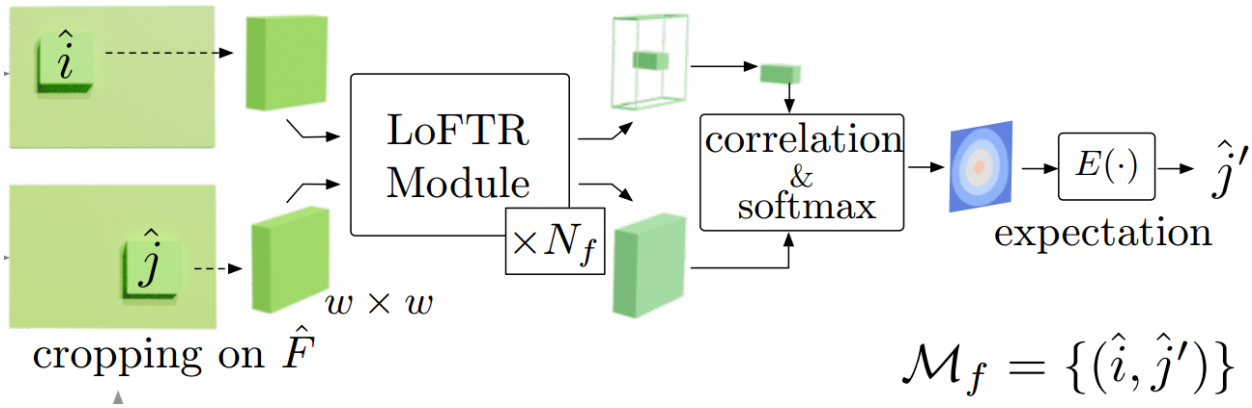
得到probability matrix下一步就是做匹配点提取了. 作者根据equation(3)来进行匹配点的提取:

$$\mathcal{M}_c = \{(\tilde{i}, \tilde{j}) \mid \forall (\tilde{i}, \tilde{j}) \in \mathrm{MNN}(\mathcal{P}_c),\ \mathcal{P}_c(\tilde{i}, \tilde{j}) \geq \theta_c\} \tag{3}$$

如果(i,j)想要成为匹配点, 首先 probability matrix的(i,j)值要大于theta_c. 作者在实际implementation中这个值取了0.2. 其次i和j要是mutual nearest neighbor. 这是什么意思呢? 其实就是A特征图的像素i和B特征图的像素j是最匹配的, 同时B特征图的像素j也和A特征图的像素i最匹配. 通俗地说就是你喜欢对方还不行, 要对方也喜欢你才行. 严谨地说就是(i,j)在probability matrix第i行是最大值, 同时在第j列也是最大值. 到此我们就完成了coarse-level local feature transform.

既然有coarse level, 根据过往经验, 后面肯定还有fine level. 如果你有这样的想法就说明你猜对了. 成功提取匹配点后, 我们进入下一步coarse to fine module. 我们为什么叫前一个步骤叫coarse呢? 因为他是在ResNet第四层的feature map上进行的, 此时的feature map每边大小是原大小的⅛, 这一层虽然perceptive field比较大, 但是失去了很多位置信息, 但是位置信息对于特征点提取太关键了, 因此在这一层取得的match不可以作为最后的特征点匹配结果. 接下来我们要在ResNet第一层的feature map(边长为原图1/2)里进行coarse to fine module, 目的是把获得的匹配点的模糊位置精确化.

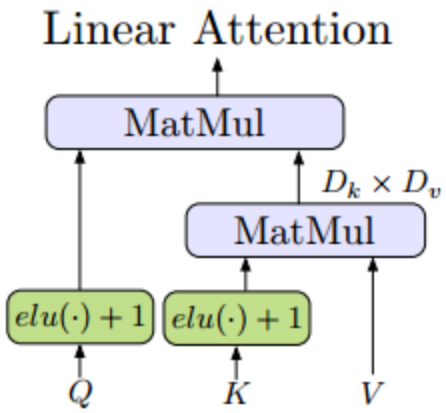
Coarse to fine module的第一步就是把匹配的i和j点分别定位到图A和图B第一层ResNet特征图 $\hat{F}^A$ 和 $\hat{F}^B$ 上(第一层特征图的channel为128, 也就是说每个像素是128-element vector). 由于位置信息的缺失, 这个定位是不准确的, 因此我们就以i和j在 $\hat{F}^A$ 和 $\hat{F}^B$ 上的定位为中心, 分别取w*w大小的窗口. 得到了两个窗口后, 我们把这两个w*w*128的矩阵输入LoFTR module, 进行上面讲解过的self-attention和cross-attention. 在coarse to fine步骤里, LoFTR module要重复Nf次(实际implementation中Nf的取值为1), 最后输出对应于两个窗口的大小还是w*w*128的两个矩阵(我们说过self-attention, cross-attention都不会改变输入的dimension). 然后我们把对应 $\hat{F}^A$ 的w*w*128矩阵取最中间的1*1*128向量. 比如说现在有长宽都是3, channel数为128的矩阵, 我们就取第二行第二列的128维向量. 然后用这个向量和对应 $\hat{F}^B$ 的w*w个128维度向量分别做点积然后全局softmax. 这样就能得到 $\hat{F}^B$ 整个窗口对应 $\hat{F}^A$ 中提取向量的matching probability. 也就是说我们得到了一个w*w*1的矩阵, 矩阵每个元素代表着 $\hat{F}^B$ 窗口的该像素和 $\hat{F}^A$ 提取向量的matching probability, 如图四所示:



图四: coarse to fine module. 先根据i和j在对应特征图定位, 然后取出两个w*w的小窗口. 把两个w*w的窗口输入LoFTR module, 还是得到两个w*w的输出. 第一个输出(对应图A)我们取中间的向量, 分别和第二个输出的每个向量做点积然后softmax, 就能得到一个w*w的matching probability. 最后对这个w*w矩阵找expectation value就能得到j的准确定位

得到w*w的matching probability矩阵后, 通常情况下我们会取argmax找到概率最大的位置作为j的定位. 但是这样就不能达到sub-pixel accuracy. 作者的做法是根据概率找到位置的expectation value. 具体做法我举个例子. 假如现在3个位置(1,0), (1,1), (1,2), 分别对应概率0.3, 0.3, 0.4, 那么expectation value就是0.3*(1,0)+0.3*(1,1)+0.4*(1,2), 最后的结果就是(1, 1.1). 通过这种方法我们可以得到更准确的定位.

在做attention中有一个细节需要讲一下. 作者为了减少计算量, 把原来关于Q, K, V的矩阵乘法改变了一种形式. 通常我们需要做Q*K.T得到score, 然后score*V得到输出. 作者称这个方法为dot product attention. 我们现在分析一下computational cost. Q, K, V的维度都是N*256. 假设我们用最经典的矩阵乘法, 那么运算量就是2*(N^2*256), 原因是Q*K.T和score*V的运算量分别是(N^2*256). 当N很大的时候, 也就是像素点非常多的时候, 运算量就是O(N^2). 作者认为N^2的computational cost是不理想的, 于是他提出了linear attention的概念. 具体实现方法如图五所示:



图五: Linear attention具体运算步骤

根据图五, 我们可以看到作者先对K做elu(K)+1的运算, 然后做K.T*V的运算(这个K是经过elu的K), 我们把这个结果称为D. 最后做Q*D的运算(Q也经过elu). 根据dimensional analysis, 我们可以分析出这个过程的运算量为2*(256^2*N), 此过程中我们忽略掉elu的运算量. 假如N很大的时候, 我们最终的运算量将是O(N). 通过Linear attention, 我们可以把computational cost从O(N^2)变成O(N). 看paper的时候我不由地发出感慨, this is such a brilliant idea!

整个流程讲完了, 最后我们讲一下loss function. 这里作者总共引用了两个loss, 分别监督coarse level matching和fine level matching. 所以最终的loss就是coarse level loss加上fine level loss:

$$\mathcal{L} = \mathcal{L}_c + \mathcal{L}_f \tag{4}$$

Lc代表coarse level loss, Lf代表fine level loss. 我们先讲讲coarse level loss. Coarse level loss在equation 5定义:

$$\mathcal{L}_c = -\frac{1}{|\mathcal{M}_c^{gt}|} \sum_{(\tilde{i},\tilde{j})\in\mathcal{M}_c^{gt}} \log \mathcal{P}_c\left(\tilde{i},\tilde{j}\right) \tag{5}$$

其中$\mathcal{M}_c^{gt}$是ground truth coarse level matches, 也就是说在1/8的特征图上从图片A到图片B的ground truth matches. 其实这个matches非常tricky, 因为我们知道1/8特征图, 每个像素对应原图是一片很大的感受野, 因此感受野中可能会有多个matches. 作者的做法是只考虑感受野的中央像素, 不考虑其他像素. 然后把原图A中央像素, 通过已知的深度图和相机外参warp到图B, 最后把原图

的中央像素的matches对应回1/8特征图. 具体实现可以看代码解读. Equation (5)的 $\mathcal{P}_c$ 就是上面equation (2)中讲过的通过dual softmax得到的probability matrix. 前面的normalization系数是 $\mathcal{M}_c^{gt}$ 的L1 norm, 也就是总共的matches的数量. Coarse level loss的数学含义就是最大化ground truth matches的log probability. 实际的implementation中是用的binary cross entropy.

Fine level loss由equation (6)定义:

$$\mathcal{L}_f = \frac{1}{|\mathcal{M}_f|} \sum_{(\hat{i},\hat{j}')\in\mathcal{M}_f} \frac{1}{\sigma^2(\hat{i})} \left\| \hat{j}' - \hat{j}'_{gt} \right\|_2 \tag{6}$$

其中 $\mathcal{M}_f$ 是coarse to fine module找到的fine matches, 也就是在½特征图找到的fine matches. i和j分别是图A和图B的对应点, j_gt是j的ground truth, 由i点通过深度图以及相机外参warp得到. 这里Fine level loss用的典型的L2 loss, 去强制j向j_gt优化. 这里的 $\sigma^2(\hat{i})$ 是对应i点的w*w matching probability matrix的variance(我们前面说过coarse to fine module会取两个w*w的窗口进行计算). 不过需要注意的是这个variance是没有进行gradient descend的, 作者在实际implementation中把他detach了.

Reference

[1] Vaswani, Ashish; Shazeer, Noam; Parmar, Niki; Uszkoreit, Jakob; Jones, Llion; Gomez, Aidan N.; Kaiser, Lukasz; Polosukhin, Illia (2017-12-05). jiuhui. arXiv:1706.03762