# 知识碎屑

## 深度学习常见问题

### 对于神经网络权重和数据的矩阵理解

假设输入维度为 `m` 隐藏神经元数为 `d` 输出维度为 `n` 批量为 `batch_size`

那么输入X为 `(batch_size, m)` 每一列代表不同的特征

$W_1$ 为 `(m, d)` 每一行代表X的同一个特征传入不同的隐藏神经元 每一列代表不同隐藏神经元

$b_1$ 为 `(, d)` 实际上可以理解为行向量 计算时触发广播机制

$W_2$ 为 `(d, n)` 每一列代表不同的输出

### 模型选择调参问题

一个很不错的比喻 训练集就是平时训练 验证集(取50%训练集)是模拟考 测试集为高考

同时，训练集用于调参 `(W,b)` ，验证集可以根据数据调整模型的 `超参数(lr，网络层数等等)`

数据集不够 可用 `K-Fold交叉验证` 这样找到最优的 `超参数组合` （通过比较平均分） 然后应用到原始训练集训练得到最终模型

### 过拟合欠拟合

`模型容量` 可以理解为拟合函数的能力，个人认为可以近似成 `函数的泰勒逼近程度` 可以用参数个数&选择范围来估计

数据简单，模型容量大(复杂) 容易过拟合；

数据复杂，模型容量小 容易欠拟合

**考虑过拟合问题 可以采用权重衰退 引入L2正则化限制w 也叫岭回归**

$$\min \quad l(w,b)+\frac{\lambda}{2}\parallel w \parallel^2$$ $\lambda$ 作为 `超参数` 表示模型复杂度

$$w_{t+1}=(1-\eta\lambda)w_t-\eta\frac{\partial l(w_t,b_t)}{\partial w_t}$$ 通常 $\eta\lambda<1$

目的是最小化loss的同时让w尽可能小

L1正则化叫lasso回归 用于特征选择

Dropout 引入噪声 神经网络中相当于部分神经坏死 和正则化一样只是在训练时使用 丢弃概率p作为 超参数

### 梯度爆炸和梯度消失

一般梯度爆炸出现在ReLU 消失出现在sigmoid

$$\prod_{i=t}^{d-1} \frac{\partial \mathbf{h}^{i+1}}{\partial \mathbf{h}^{i}} = \prod_{i=t}^{d-1} \operatorname{diag}(\sigma'(\mathbf{W}^{i}\mathbf{h}^{i-1}))(\mathbf{W}^{i})^{T}$$

梯度消失导致顶部训练的好 但底部无法训练 无法让网络训练更深

一般来说 乘法变加法能很好解决这个问题 如LSTM和ResNet 或者让每层的正向反向均值方差一致

权重初始化时 对于小网络一般服从 $W \sim N(0, 0.01)$

## pytorch一些细节处理

`nn.Flatten()` 函数保留第一维度 将其他维度摊平 `（256，1， 28， 28）-> (256, 784)`

dataset格式第一级返回的是 `(img, label)` 例如 `train_dataset[0]` 返回 `img, label` `train_dataset[0][0]` 返回 `img`

`transforms` 库主要用到 `ToTensor()` 将 `(H,W,C)` 转成 `(C,H,W)` ,并将灰度变成0-1阈值形式

> `Resize(size) or Resize(h, w)` 前者将最短边缩放到size，其他按比例缩放 后者强制缩放到 `h,w`
>
> `Normalize(mean, std)` 用给定的均值 `（mean）` 和标准差 `（std）` 对 `Tensor` 进行归一化。需要注意的是mean和std需要与通道数对应，即是个列表
>
> `Flipping & Rotation` 用到的不多 自行查阅

### 常见的激活函数

`ReLU` $max(x, 0)$ `sigmoid` $\frac{1}{1+e^{-x}}$ `tanh` $\frac{1-e^{-2x}}{1+e^{-2x}}$ ReLU容易爆炸 sigmoid容易消失

`nn.ReLU()`、`nn.Sigmoid()`、`nn.Tanh()`

优化后的 `nn.LeakyReLU()`、`nn.PReLU()` 解决梯度为0的问题 `nn.GELU()` 用于 `Transformer`

`常见的损失函数`

`nn.MSELoss()、nn.CrossEntropyLoss()`

### 优化器

在`torch.optim`定义，常见有`SGD(net.parameters(), lr=...)`

`optim.Adam(model.parameters(), lr=..., weight_decay=0.001)` `decay`是权重衰退程度

## 数据预处理

🌐 数据处理代码

一般用pandas库

### 一些pandas操作

#### 索引选择

`reviews['country']`或`reviews.country`访问列 再索引才是行

分iloc和loc两种 一般不用特征筛选时基本上用`iloc位置索引` 如`reviews.iloc[:, 0] reviews.iloc[1:3, [0, 1, 2]]` 注意`前闭后开`

loc位置索引时前闭后闭 如`reviews.loc[0, 'country']`

`set_index('某一列')` 将原有的行数字索引改为数据的某一列标签

条件选择 如`reviews.country == 'Italy'` 会返回布尔值

`reviews.loc[(reviews.country == 'Italy') & (reviews.points >= 90)]`可以筛选出只符合条件的行 或者用`isin`
`reviews.loc[reviews.country.isin(['Italy', 'France'])]` 来筛选出法国或者意大利

`reviews.loc[reviews.price.isnull()]` 筛选出缺失值的列

`reviews.price.isnull().sum()` 有多少行缺失值 isnull返回布尔值列表

#### 数值映射

`describe函数` 一般对数值进行统计 `reviews.taster_name.value_counts()`唯一值列表和出现频率

`map()`函数 一般与lambda函数连用`reviews.points.map(lambda p: p - review_points_mean)`

pandas内置了map 可以直接用 还可以组合 一般用apply

`groupby()` 分组函数 可以多索引分组 类似树状结构
`reviews.groupby('points').price.min()`

排序 `sort_values(by = '某一列', ascending = True/False)`

**数据转换**

`dtype` 获取类型 `astype('数据类型')` 转换

**数据缺失**

遇到字符串如 `“invalid”` 时，一般用 `replace("被替换", “替换词”)`

`reviews[pd.isnull(reviews.country)]`

`数值缺失用reviews['price'].fillna(...)` 注意这是副本 要 `reviews['price'] = ...` 再赋值

**重命名**

`reviews.rename(columns={'points': 'score'})` 列名替换

`reviews.rename(index={0: 'firstEntry', 1: 'secondEntry'})` 行索引替换

**常用操作**

`numeric_features = all_features.dtypes[all_features.dtypes != 'object'].index` 可以提取数值的特征 `features` dtypes返回列名和类型

`all_features = pd.get_dummies(all_features, dummy_na=True)` 将非数值型 离散型用独热编码替代，如 `“MSZoning”` 包含值 `“RL”` 和 `“Rm”` 。 我们将创建两个新的指示器特征 `“MSZoning_RL”` 和 `“MSZoning_RM”` ，其值为0或1

.columns返回所有列名 .index返回行标签

## 强化学习算法

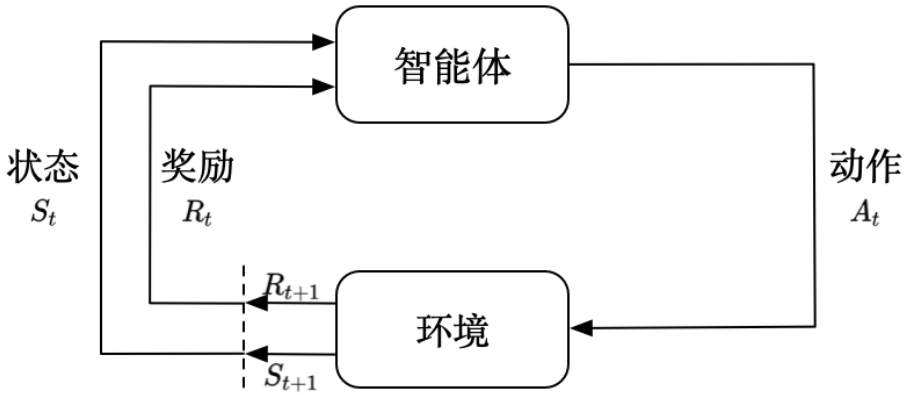
强化学习总体而言 就是智能车与环境的交互，在复杂环境中最大化获得的奖励。

智能体在获得一个 `状态state` 后，然后依据该状态输出 `动作action/决策decision` ，然后在环境中被执行，而后环境依据动作来输出下一个state和带来的 `奖励reward` 以此类推

累计奖励reward的期望叫作 `价值(value)`

面向决策任务的智能体进行交互的环境是一个动态的随机过程，其未来状态的分布由当前状态和智能体决策的动作来共同决定，并且每一轮状态转移都伴随着两方面的随机性：一是智

能体决策的动作的随机性，二是环境基于当前状态和智能体动作来采样下一刻状态的随机性。




## Pytorch

```
from torch.utils.tensorboard import SummaryWriter
from torchvision import transforms
writer = SummaryWriter("logs") # 创建一个logs文件夹，writer写的文件都在该文件夹下
writer.add_image("test",img_array1,1,dataformats="HWC") # 1 表示该图片在第1步
writer.add_image("test",img_array2,2,dataformats="HWC") # 2 表示该图片在第2步
以上是将PIL图像转为array后形式为 HWC 显示
writer.add_image("Temsor_img",tensor_img)
```

## RNN

在每个时间步 $t$，RNN 执行两个核心计算：

1. **更新隐藏状态** $h_t$：结合当前输入 $x_t$ 和上一个时间步的隐藏状态 $h_{t-1}$。

$$h_t = \tanh(W_{xh}x_t + W_{hh}h_{t-1} + b_h)$$

2. **计算当前输出** $y_t$：基于更新后的隐藏状态 $h_t$。

$$y_t = W_{hy}h_t + b_y$$

自己实现 `rnn_by_hand.py` 有很多细节 常见输入格式是 `(批次，序列长度， 输入维度)`

注意torch.stack()的运用 比如张量列表可用

nn模块内置的RNN模型，需要注意格式为：`nn.RNN(input_size=input_dim, hidden_size=hidden_dim, batch_first=True)` 同时隐藏层的第一个维度是RNN层数，默认是1

```
H_out = floor( (H_in - Kernel_Size + 2 * Padding) / Stride ) + 1
```

## 图像形态学处理

膨胀和腐蚀 开启和闭合

开运算：腐蚀后膨胀

闭运算：膨胀后腐蚀

(4) 开运算：

- 腐蚀后再进行膨胀的组合运算。这可以去除小的前景类物体或毛刺，而保持较大的前景类物体不受干扰。

(5) 闭运算：

- 膨胀后再进行腐蚀的组合运算。这可以去除前景类物体中的小孔洞。

## 注意力机制和变体

https://zhuanlan.zhihu.com/p/410776234 经典自注意力机制、多头注意力、跨模态注意力等

## 对于BN、LN、Add的理解

BN批次归一化 LN层归一化 一般涉及到CNN、视觉用前者，后者一般用于处理transformer、长序列(nlp or 视觉patch)

常见的比较好用的block有：

**CBR： Conv+BN+Activation**

**BAC:BN -> Activation -> Conv**

现代transformer：

**Pre-LN (Norm -> Add) 先归一化再残差连接 或者反过来 一般前者效果比较好**

## 插值基础

https://www.bilibili.com/video/BV1wh411E7j9?spm_id_from=333.788.videopod.sections&vd_source=fdddfeef02a69f986ee4635e0449d1a5

**坐标映射**，一般按照x，y尺寸得到缩放因子scale_factor,得到映射到原图像的坐标

**最近邻插值** 用round四舍五入取整

**双线性插值** 需要考虑中心对齐方式，分为角对齐(四个角对齐)和像素中心对齐(边对齐) 然后用最近的四个点去拟合映射后的像素点，方法类似于向量分线段比例。

**双三次插值** 同时考虑周边16个点，权重用公式计算出。

## Batch Normalization

批量归一化。对中间网络层的输出，激活函数之前进行。先归一化再缩放平移。

## 扩展卷积操作

**转置卷积upsambling**

https://zh.d2l.ai/chapter_computer-vision/transposed-conv.html

扩展后尺寸

$$o = s(i-1) + k$$

s是步长，i是输入图尺寸，k是kernel尺寸 一般取2，2可以得到2倍上采样的效果

**空洞卷积**

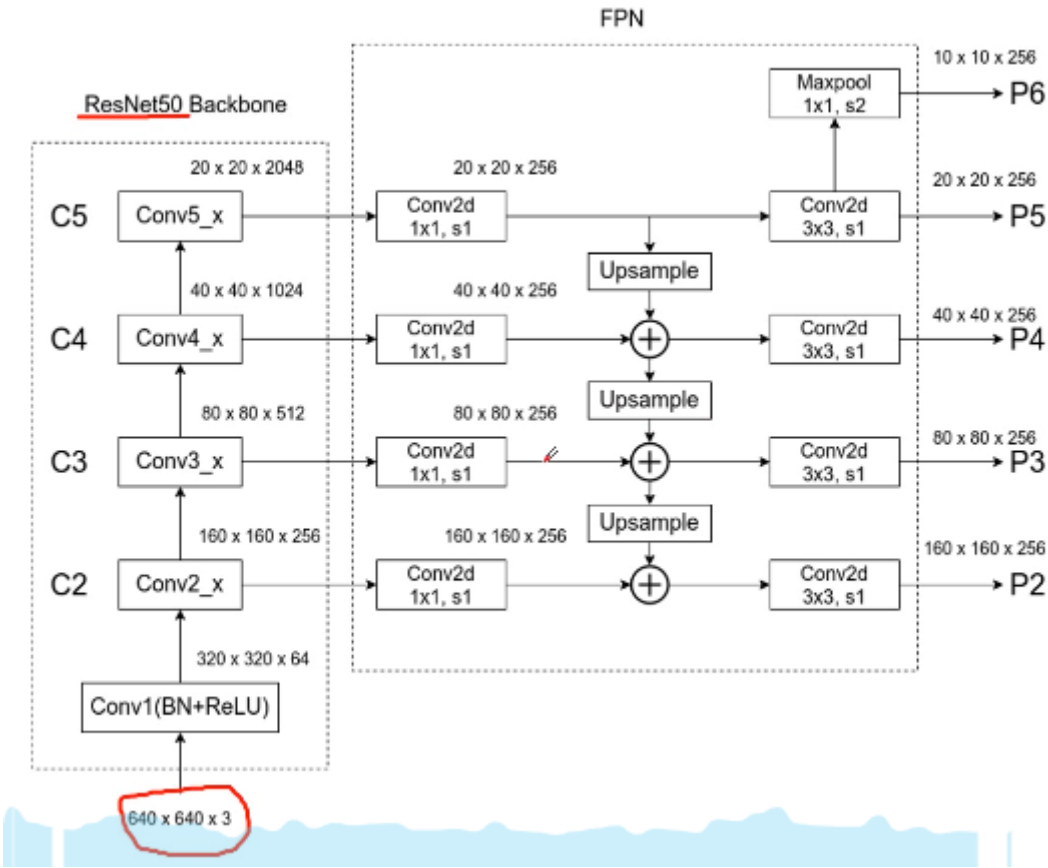
引入膨胀率，增大卷积感受野，避免陷入局部。

**可分离卷积**

空间换时间，使得卷积平方复杂度变为线性复杂度，例如3*卷积核可以拆成*3*1和1*3的向量相乘 另外还有**深度可分离卷积**

inception开端结构 让模型自己去学习用多大的卷积核最好 融合多尺度，Inception 模块在保持特征图空间尺寸（高度和宽度）不变的情况下，通过并行计算和通道拼接，生成了一个包含了多尺度信息的、通道数更丰富的（或按需调整的）新特征图

## FPN特征金字塔网络




# 基于像素的分割 based on pixel and region proposal

这里就只记一下mask-rcnn的流程和细节地方。

maskrcnn多了**rolaligned和新的并行头掩码预测损失**

**faster-rcnn**针对fast的改进：RPN+fastrcnn的backbone网络 前者是区域提案网络 用于在特征图上进行二分类和边界框回归。

**RPN：**

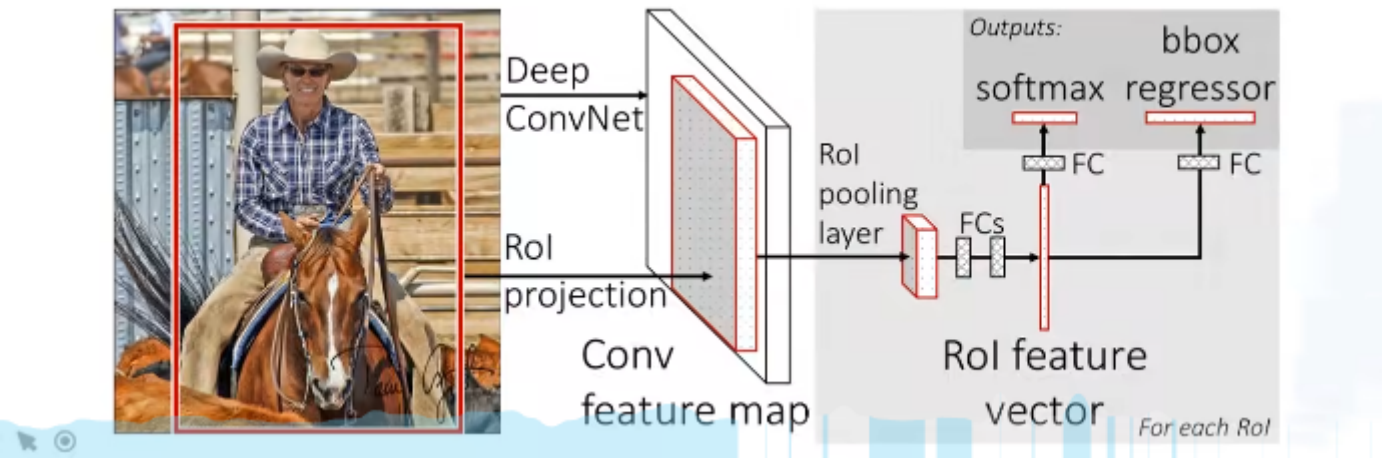
提取特征图的过程可以参考FPN网络，通过对resnet进行上采样得到多尺度特征图，然后对特征图进行3*3卷积对区域进行特征融合后，再对每个像素分别进行1*1卷积(类似全连接)，分别得到每个像素的分类特征向量和回归特征向量，二者的维度分别是锚框数量的倍数2，4(0/1前景背景，四个边界框参数)；

**RoI pooling操作**：首先将rpn网络筛选出来的区域映射到特征图上，通常是float浮点数，直接**取整**。

例如要获得7*7大小的统一尺寸，相当于把原尺寸取整后/7**再取整**，划分为7*7的小网络，然后对每个小网络进行**最大池化**

**RoIAligned操作**是在rol pooling基础上用了双线性插值，而不是简单的取整，对于每个小网络采样4个点（一般是四个2*2的中心）进行插值计算，再最大池化

然后后面就是fastrcnn的内容了：




先对锚框进行回归剪裁，这样能得到初始的边界框，然后再插值映射到特征图，进行rol pooling就行了，紧接着两个全连接层Fcs(**高层抽象降维**)融合尺度信息，进行回归和分类预测了。

**Smooth l1损失 结合了l1和l2损失**

训练时可采用**联合训练，或者先训练RPN。**

损失函数是以下损失的加权求和：

1. **RPN 分类损失** (前景/背景的交叉熵损失)
2. **RPN 回归损失** (对锚框位置的 Smooth L1 损失)
3. **最终分类损失** (对 RoI 类别的交叉熵损失)
4. **最终回归损失** (对 RoI 位置的 Smooth L1 损失)

$$L(p_i, t_i, u, v) = L_{RPN}(p_i, t_i) + L_{FastRCNN}(p, u, t^u, v)$$

$$L_{RPN}(p_i, t_i) = \frac{1}{N_{cls}} \sum_i L_{cls}(p_i, p_i^*) + \lambda \frac{1}{N_{reg}} \sum_i p_i^* L_{reg}(t_i, t_i^*)$$

- `i`: 表示一个批次中所有锚框 (anchor) 的索引。
- `p_i`: 模型预测的第 `i` 个锚框是前景的概率。
- `p_i^*`: 第 `i` 个锚框的真实标签 (ground-truth label)。如果锚框是前景，则为 1；如果是背景，则为 0。
- `t_i`: 模型预测的与第 `i` 个锚框相关的 4 个边界框回归参数。
- `t_i^*`: 第 `i` 个前景锚框对应的真实回归目标。
- `L_cls`: 分类损失，通常是二分类对数损失 (交叉熵)。

$$L_{cls}(p_i, p_i^*) = -p_i^* \log(p_i) - (1 - p_i^*) \log(1 - p_i)$$

- `L_reg`: 回归损失，采用 Smooth L1 Loss。
- `N_cls` 和 `N_reg`: 归一化项，分别是 mini-batch 的大小和锚框位置的数量。
- `λ`: 平衡参数，用于调节两个损失的权重。

$$L_{FastRCNN}(p, u, t^u, v) = L_{cls}(p, u) + \lambda' \cdot [u \geq 1] \cdot L_{loc}(t^u, v)$$

## Unet及其变体

正常的Unet就是下采样 上采样 跳跃连接； 后来引入transformer，也为后面多模态的扩散Diffusion打个基础。

### Trans-Unet

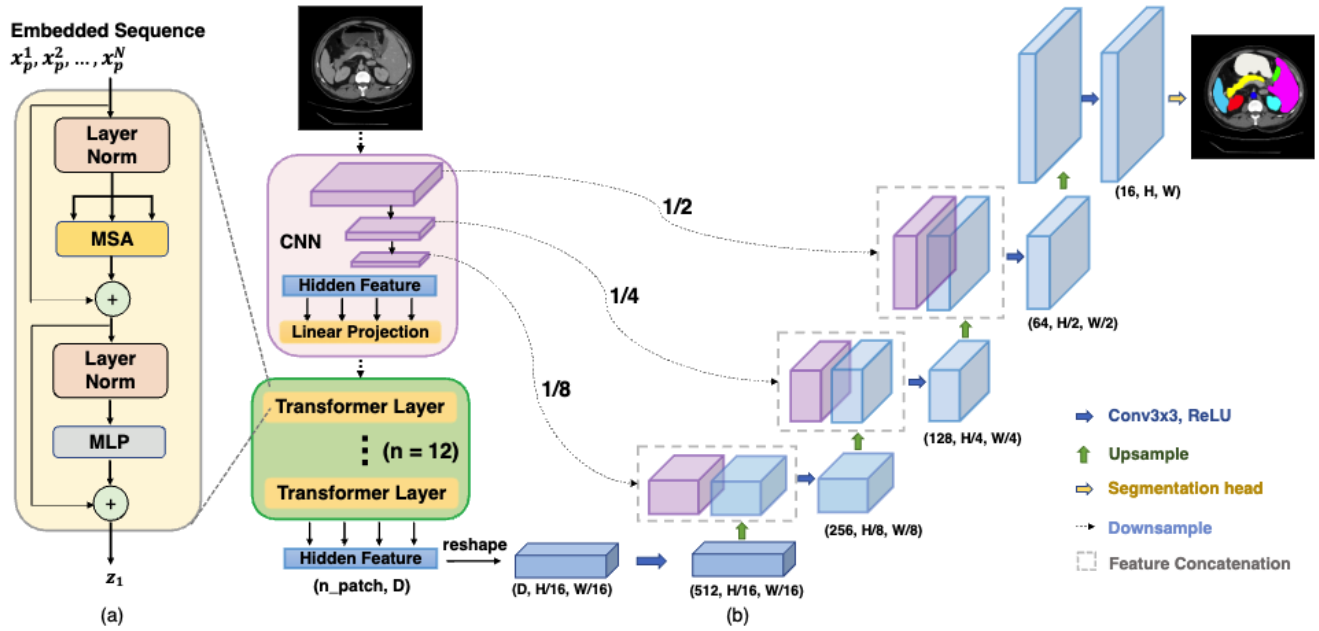



Fig. 1: Overview of the framework. (a) schematic of the Transformer layer; (b) architecture of the proposed TransUNet.

单纯的CNN**缺乏长程依赖关系**，单纯的Transformer架构**无法处理patch内部的关系，二者结合，值得一试**

流程大致上是将CNN提取的深层特征经过线性层转换Transformer输入维度，展平HW，进入trans层，维度不变，摊开HW，瓶颈层进入CBR改变维度，后面进入经典的上采样

## Swin-Unet 纯transformer-U架构

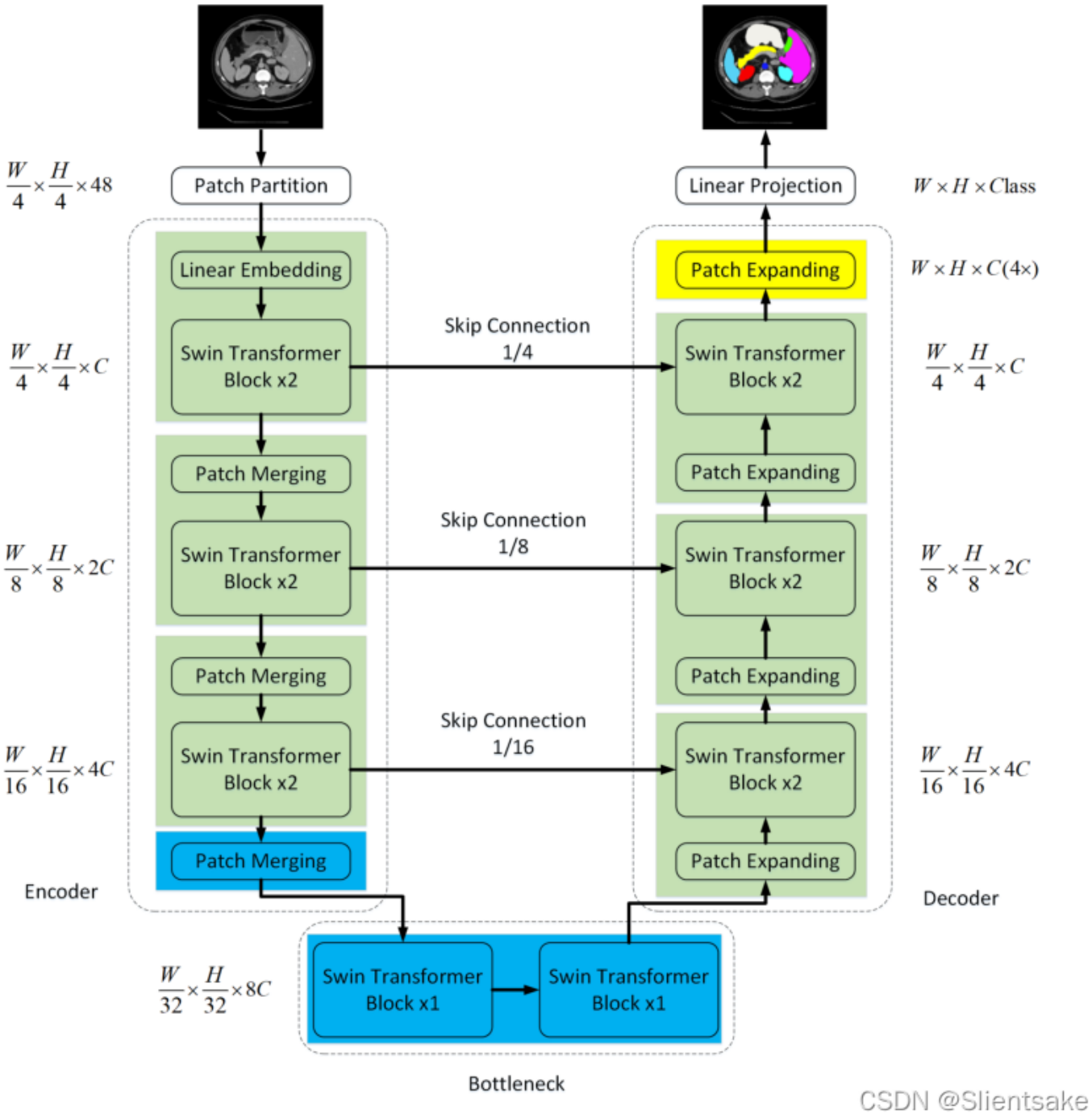
基于Swin Transformer block，构建了一个具有跳跃连接的对称编码器-解码器体系结构。在编码器中实现了从局部到全局的自注意；

开发了patch扩展层，**无需卷积或插值操作即可实现上采样和特征维数的增加**。

**其中**，补丁合并层负责下采样和增加维度，Swin Transformer块负责特征表示学习，补丁扩展层被专门设计用于执行上采样

Patch Partition是相当于用一个4*4的卷积核对区域进行合并操作，将4*4的区域合并成一个像素，相应的通道数扩展连接16倍 16*3=48 后面的Patch Merging同样的操作，只不过区域变成2*2，然后还需要通过LN+Fc降维到2C

Patch Expanding操作步骤是先进入Fc+LN进行通道扩展，然后等分成4份，patch的size扩大两倍






**Swin-transformer块**

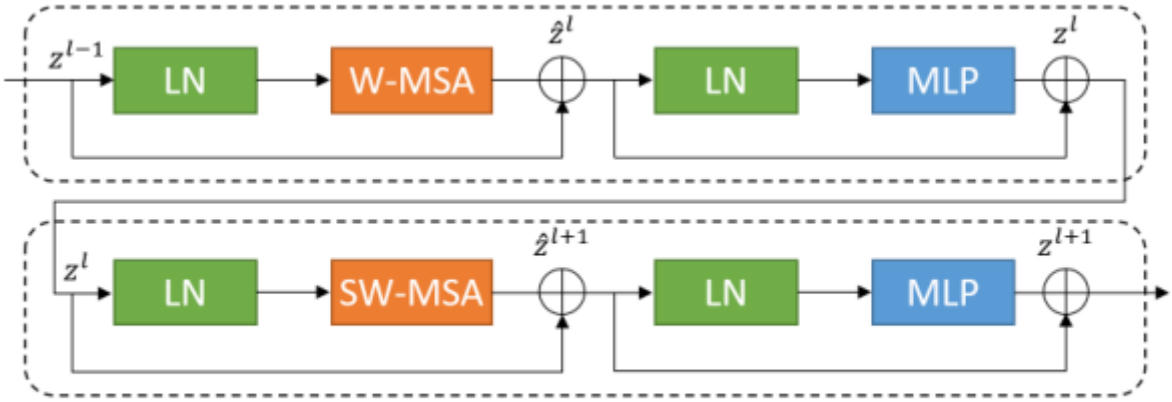



**Fig. 2.** Swin transformer block. 

**这里的注意力模块，个人理解为每个patch可以理解为一个词，比如w*h相当于句子的长度，而通道数C相当于每个词的嵌入向量，这样矩阵size相当于(w*h, C)**

**还需要还需要注意的一是MSA是对各个patch间的全局融合 而MLP也就是FFN是对通道间的特征融合，无独有偶，其他情况也是一样的**

与传统的多头自注意(MSA muti-head self-attention)模块不同，swin transformer块是基于平移窗口构造的。在图2中，给出了两个连续的swin transformer块。每个swin transformer块由LayerNorm (LN)层、多头自注意模块、剩余连接和具有GELU非线性的2层MLP组成。在两个连续的transformer模块中分别采用了基于窗口的多头自注意(W-MSA)模块和位移的基于窗口的多头自注意(SW-MSA)模块。基于这种窗口划分机制，连续swin transformer块可表示为:

$$\hat{z}^l = W\text{-}MSA(LN(z^{l-1})) + z^{l-1}, \tag{1}$$

$$z^l = MLP(LN(\hat{z}^l)) + \hat{z}^l, \tag{2}$$

$$\hat{z}^{l+1} = SW\text{-}MSA(LN(z^l)) + z^l, \tag{3}$$

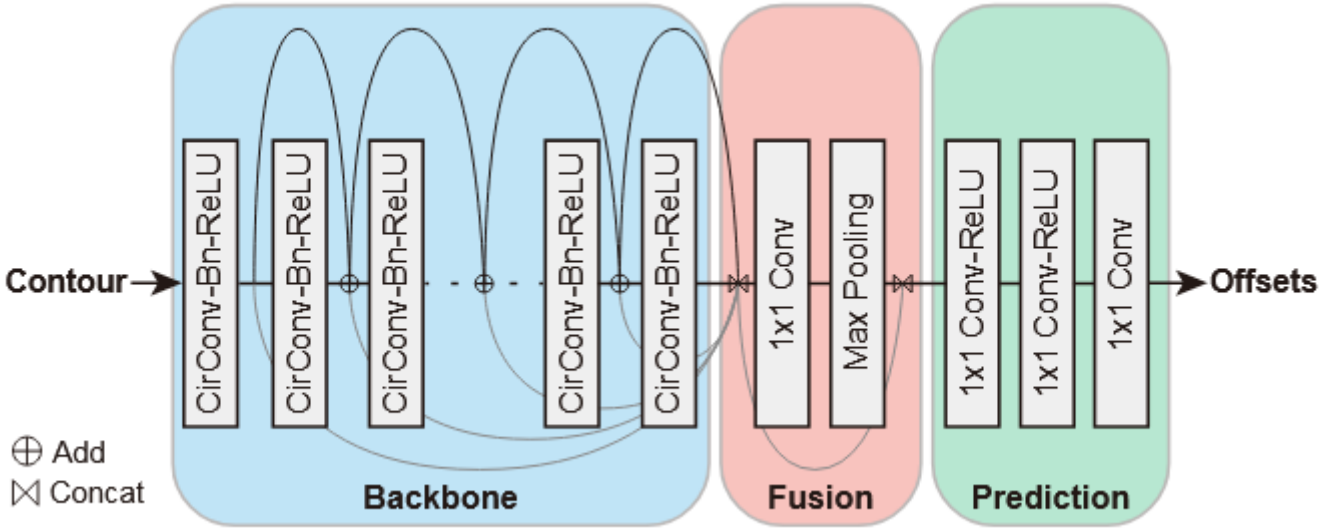
$$z^{l+1} = MLP(LN(\hat{z}^{l+1})) + \hat{z}^{l+1}, \tag{4}$$



## 关于DeepSnake及其改进

*模型总体结构图(按道理应该只是DS的部分)如下：*

*图中上面的线是残差连接，下面是有点与多尺度全局信息融合的意思*




**论文中没有太多讲述CenterNet的内容**，这里着重阐述一下，以及为什么模型是一个端到端的架构

## 模型架构流程

正常输入图像，用BackBone下采样得到特征图，记为F（***同时给检测头和后续的Snake采样点特征***）

两种检测思路：yolo、rpn的**预定义锚框anchor** 以及**anchor-free式**的如CenterNet

检测阶段：有点类似RPN操作，框出检测框，可以选择用yolo等检测算法，这里用*CenterNet*为例，用特征图F接三个head，分别用作**热力图分类**(卷积几层得到K个通道，K是类别数（**每个通道代表一种类别**），每个像素代表一个bbox的中心点置信度)、**尺寸回归**（2个通道，表示中心点对应原图bbox的wh）、**中心偏移offset**（弥补下采样的误差）

**解码bbox**：在热力图上做top-K（提前进行局部NMS），每个得到的位置映射到原图 进行尺寸输出加offset得到bbox（同时还有一个类别score）至此得到检测框。

**菱形插值**：在每个bbox每条边取中点连接得到菱形（top，bottom，left，right），再均匀插值40个点(xi, yi)，然后映射到F特征图上，用双线性插值得到具体的特征向量和坐标拼接(归一化后的坐标，保证平移不变性)。

第一次**Extreme Snake 从菱形逼近物体四个极值点：** 这里的Snake可以参考上面的流程图，8次环形卷积，每两次过渡后感受野增大，中间有残差连接，输出（40，D）得到高浓缩的局部信息。然后**fusion全局融合，**8层的输出拼接，用1*1卷积降维，做max-pooling得到（1，D’），广播后拼接到(40,D+D')。最后**偏移回归** 3层1*1卷积和ReLU，输出（40，2），分别是dx，dy偏移向量，得到8个点的**polygon（在每个极值点所在的那条边上向两侧延伸 1/4 边长）**

**后续的Contour Snake（一般3轮迭代）：** 得到的八边形继续插值**128**个点，与Extreme**同样的操作（论文中没有共享参数）。然后**更新轮廓点坐标，进入下一轮迭代。

## 损失函数设计以及轮廓训练对齐问题

分极值点回归和后续的轮廓回归

**极值点回归**：并不复杂，只需要将原菱形顶点回归4个极值点即可，剩下的36个点只是负责提供感受野信息。

$$L_{\text{extreme}} = \frac{1}{4}\sum_{k=1}^{4}\text{smoothL1}(v_k + \Delta v_k,\ x_k^{ex})$$

金标准的GT如果是binary mask就跑一边findContours，然后取x最小 最大 y最小 最大提取极值点 有一样的就取平均。

**轮廓回归**：**GT 轮廓重采样 + 点对齐**

八边形采样：模型采用从top极值点均匀沿八边形采样128个点

GT采样：均匀采样128个点，往往不包含top极值点，选择离其最近的一个点，然后进行cycle shift，让其变成序列第一个点：

$$q_i = \tilde{q}_{(k^{*}+i-1)\bmod N}$$

loss为：

$$L_{\text{snake}} = \frac{1}{N}\sum_{i=1}^{N}\text{smoothL1}(p_i^T,\ q_i)$$

## 师兄已改进部分和后续一些设想