Anchor-based缺点

- Anchor-Based检测性能**对于anchor的大小，数量，长宽比都非常敏感**，这些固定的anchor极大地**损害了检测器的泛化性**，导致对于不同任务，其anchor都必须重新设置大小和长宽比；
- 为了去匹配真实框，需要生成大量的anchor，但是**大部分的anchor在训练时标记为negative，所以就造成正负样本的不平衡**；
- 在训练中，需要计算**所有anchor与真实框的IOU，这样就会消耗大量内存和时间**；
- **感受野可以看作是天然的Anchor；**

Anchor Free的缺点

- **语义模糊性，即两个物体的中心点落在了同一个网格中**
  - Anchor Free：通常一个位置先安排好了一个正样本后，如果这个位置又来了一个新的正样本，那么前一个就会被覆盖掉，在出了FPN的技术后，这个问题似乎看起来解决了不少，但是它应该落在哪个feature map上又是选择问题：人工设定，自适应；
  - Anchor Based：假设我们设置了3个anchor box先验框，那么让第一个框和一个物体匹配上，那么后面来的物体就和其他两个框去做匹配就行了；
- Anchor Free在优化上，由于没有先验的东西来支持，所以不如Anchor Based的方法**稳定**；

Label Assignment

- RetinaNet根据**Anchor和目标的IoU**来确定正负样本；
- FCOS根据**目标中心区域和目标的尺度**确定正负样本；

Assign算法的原则：

- 中心先验：FCOS / CenterNet
- Loss aware（动态匹配）：FreeAnchor / ATSS
- 不同目标设定不同数量正样本（进一步动态）：PAA / AutoAssign
- 全局信息：IQDet / OTA

数据集 & 框架 & 模块化

数据集发展：

```
PASCAL VOC 2007 -> ILSVRC 2009 -> MS COCO 2014 -> Open Images Detection 2018 ->
Objects365
```

07+12：使用 **VOC2007 的 train+val 和 VOC2012的 train+val** 训练，然后使用 **VOC2007的test**测试；

07++12：使用 **VOC2007 的 train+val+test 和 VOC2012的 train+val**训练，然后使用 **VOC2012的test**测试；

框架：

```
mmdetection  港中文 商汤
detectron2  facebook
PaddleDetection 百度
```

检测网络模块化：

```
dataset: datasources / augs / loader / sampler
trainner: train-loops / lr-scheduler / optimizer / loss / logger   ['skɛdʒʊlər]
evaluator: ap / map / fppi
config: hyper-params
layers: roi pooling / nms
model: backbone / neck / head
```

网络的分类

基于阶段

```
稀疏预测：
 - 多阶段目标检测：Casade RCNN
 - 两阶段目标检测：RCNN / Fast RCNN / Faster RCNN

密集预测：
 - 单阶段目标检测：SSD / YOLO v1~v5 / RetinaNet / EfficientNet / EfficientDet /
CornerNet / FCOS
```

是否使用Anchor

```
Anchor Free:
 - YOLO v1
 - CornerNet
 - FCOS

Anchor based:
  Dimension Clusters:
   - YOLO v2 ~ YOLO v5
   - PP-YOLO
   - EfficientNet / EfficientDet
  Hand pickeed:
   - SSD
   - RPN(Faster RCNN)
```

不同标签方案

```
region proposal-based:
 - RCNN / Fast RCNN / Faster RCNN

基于keypoint-based:
 - YOLO v1
 - CornerNet
 - FCOS

基于author-IoU:
 - SSD
 - RPN(Faster RCNN)
 - YOLO v2 ~ v5
 - PP-YOLO
 - EfficientNet
 - EfficientDet
```

类别不平衡

数据

- OHEM（正负样例不平衡）
- **过采样** / 欠采样
- **增加数据**
- Input Mixup + DRS（基于Re-Sampling的Re-Balancing）+ CAM-BS（基于Class Activation Maps的数据增广，Balance-Sampling） + Fine-Tuning

Loss

- **Focal Loss**（正负样例不平衡alpha，简单困难样例不平衡belta）
- 目前普遍存在一个误解，认为focal loss是解决样本不均衡的杀器，实际上更重要的是分类层bias的初始化(yolox和v5都用了），另外在300epoch的训练轮数下，不均衡问题也基本不是问题了

网络

- **BBN**

传统目标检测

区域选择->特征提取->分类器

- 使用不同尺度的滑动窗口选定图像的某一区域为候选区域；
- 从对应的候选区域提取如Harrs HOG等一类或者多类**特征**；
- 使用 SVM 等分类算法对对应的候选区域进行分类，判断是否属于待检测的目标；

缺点

- 基于滑动窗口的区域选择策略没有针对性，**时间复杂度高，窗口冗余**
- 手工设计的特征对于多样性的变化没有很好的**鲁棒性**

ROI Pooling & Align

```
ROI Pooling
ROI Align
```

两次整数化（量化）过程:

- **region proposal**的xywh通常是小数，但是为了方便操作会把它整数化；
- 将整数化后的边界区域**平均分割成 k x k 个单元**，对每一个单元边界进行整数化；

经过上述两次整数化，此时的候选框已经和最开始回归出来的位置有一定的偏差，这个偏差会影响检测或者分割的准确度； -> **mis-alignment**

ROI Align: **取消量化操作**，使用双线性内插的方法获得坐标为浮点数的像素点上的图像数值,从而**将整个特征聚集过程转化为一个连续的操作**；1.遍历每一个候选区域，保持浮点数边界不做量化；2.将候选区域分割成k x k个单元，每个单元的边界也不做量化；3.在每个单元中计算固定四个坐标位置，用双线性内插的方法计算出这四个位置的值，然后进行最大池化操作；

FPN

- **多尺度特征融合**
- 使用**分治思想解决稠密目标检测**（YOLOF）

- FPN结构**在高层的语义特征进行融合效果并不好**，所以构建FPN**没有必要使用所有的卷积层**;
- 改进：PAN & Bi-FPN

FPN的特征融合为什么是Add

- 对于**两路输入**来说，如果是**通道数相同且后面带卷积**的话，**add等价于concat之后对应通道共享同一个卷积核**;
- FPN里的金字塔，是**希望把分辨率最小但语义最强的特征图增加分辨率**，从性质上是可以用add的；如果用concat，因为分辨率小的特征通道数更多，计算量是一笔不小的开销，所以FPN里特征融合使用相加操作可以理解为是为了**降低计算量**;

FPN为什么能提升小目标的准确率

低层的特征语义信息比较少，但是目标位置准确；高层的特征语义信息比较丰富，但是目标位置比较粗略：原来多数的object detection算法都是只采用顶层特征做预测，**FPN同时利用低层特征高分辨率和高层特征的高语义信息**，通过融合这些不同特征层的特征达到预测的效果，并且**预测是在每个融合后的特征层上单独进行的，**所以可以提升小目标的准确率;

基于FPN的RPN是怎么训练的

在FPN的每个预测层上都接一个RPN子网络，确定RPN子网络的正负anchor box样本，再计算各预测层上RPN的anchor box分类和回归损失，利用BP将梯度回传更新权值;

NMS

```
算法输入：算法对一幅图产生的所有的候选框，每个框有坐标与对应的置信度；

算法输出：输入的一个子集，同样是一组5维数组，表示筛选后的边界框；

算法流程：

1．将所有的框按类别划分，并**剔除背景类**，因为无需NMS；
2．对每个物体类中的边界框(B_BOX)，按照分类**置信度降序**排列；
3．在某一类中，选择置信度最高的边界框B_BOX1，将B_BOX1从输入列表中去除，并加入输出列表；
4．逐个计算B_BOX1与其余B_BOX2的交并比IoU，若IoU(B_BOX1,B_BOX2) > 阈值TH，则在输入去除B_BOX2；
5．重复步骤3~4，直到输入列表为空，完成一个物体类的遍历；
6．重复2~5，直到所有物体类的NMS处理完成；
7．输出列表，算法结束；
```

NMS算法中的最大问题就是它将相邻检测框的分数均强制归零(即将重叠部分大于重叠阈值Nt的检测框移除)：在这种情况下，**如果一个真实物体在重叠区域出现，**则将导致对该物体的检测失败并降低了算法的平均检测率;

经典的NMS算法将IOU大于阈值的窗口的得分全部置为0，可表述如下：

$$s_i = \begin{cases} s_i, & \text{iou}(\mathcal{M}, b_i) < N_t \\ 0, & \text{iou}(\mathcal{M}, b_i) \geq N_t \end{cases}$$

Soft NMS

- soft-NMS在**训练中采用传统的NMS方法，仅在推断代码中实现soft-NMS**
- Soft-NMS可以很方便地引入到object detection算法中，不需要重新训练原有的模型、代码容易实现，不增加计算量

Soft-NMS吸取了NMS的教训，在算法执行过程中不是简单的对IoU大于阈值的检测框删除，而是降低得分。算法流程同NMS相同，但是对原置信度得分使用函数运算，目标是降低置信度得分;

置信度重置函数有两种形式改进，一种是线性加权的：

$$s_i = \begin{cases} s_i, & \text{iou}(\mathcal{M}, b_i) < N_t \\ s_i\left(1 - \text{iou}(\mathcal{M}, b_i)\right), & \text{iou}(\mathcal{M}, b_1) \geq N_t \end{cases}$$

一种是高斯加权形式：

$$s_i = s_i e^{-\frac{\text{iou}\ (\mathcal{M}, b_i)^2}{\sigma}}, \forall b_i \notin \mathcal{D}$$

在迭代终止之后，Soft-NMS依据预先设定的得分阈值来保留幸存的检测框，通常设为0.0001(超参数的不敏感性)

精度提升

这些NMS都基于这样的假设：与当前最高得分框重叠越大，越有可能是冗余框：

- 分类优先：**NMS，Soft-NMS**
- 加权平均：**Weighted NMS** (YOLO v5)
- 自适应阈值**Adaptive NMS**：上述NMS在物体之间有严重遮挡时可能带来不好的结果，我们期望当物体分布稀疏时，NMS大可选用小阈值以剔除更多冗余框；而在物体分布密集时，NMS选用大阈值，以获得更高的召回；
- 中心点距离：**DIoU-NMS**(YOLO v4)

效率提升

- **Fast NMS**
- **Cluster NMS**：cluster是一个框集合，若某一个框A属于这个cluster，则必有cluster中的框与A的IoU≥NMS阈值，且A不会与除这个cluster之外的框有IoU≥NMS阈值，Cluster NMS的迭代次数通常少于Traditional NMS的迭代次数；
- 得分惩罚机制**SPM NMS**
- **Matrix NMS**：SOLOv2：Matrix NMS将mask IoU并行化，得分惩罚机制SPM，PPYOLOv2将Matrix NMS引入

- [x] https://zhuanlan.zhihu.com/p/151914931
- [x] https://zhuanlan.zhihu.com/p/157900024

Concat & Add

- 两者都可以理解为**整合Feature map信息**；
- Add是将对应的特征图相加，再进行下一步卷积操作，**描述图像的维度本身并没有增加，只是每一维下的信息量在增加**，这显然是对最终的**图像分类**是有益的；(相当于加了一个先验：**对应通道的特征图语义类似，从而对应的特征图共享一个卷积核**)
- Concatenate是通道数的合并，**描述图像本身的通道数增加了，而每一特征下的信息是没有增加**，concat每个通道对应着对应的卷积核；
- **对于两路输入来说，如果是通道数相同且后面带卷积的话，add等价于concat之后对应通道共享同一个卷积核**；因此add可以认为是特殊的concat形式，但是**add的计算量要比concat的计算量小得多**；

BackBone -> Neck -> Head

Backbone

Backbone于能为检测提供若干种 感受野大小和中心步长的组合 以满足对 不同类别和尺度的目标检测

```
ResNet / DenseNet / ResNeXt
CSPDarknet53 / CSPResNeXt
EfficientNet / EfficientDet
MobileNet / ShuffleNet / GhostNet
```

Neck

Neck将来自于Backbone上的多个层级的特征图进行融合加工，增强其表达能力，同时输出加工后并具有相同宽度的特征图以供Head使用；

```
SPP
FPN -> FPN结构在高层的语义特征进行融合效果并不好，所以构建FPN没有必要使用所有的卷积层；
BiFPN
PAN
```

Head

参数共享：P3,P4,P5共用一个Head

- 解决了**多Head多尺度样本数量不均衡**；（YOLOF的Uniform Match）
- 为什么可行：**神经网络具有尺度语义不变性**；

- 优点：**减少参数，提升精度**；
- 共享Head**无法使用BN**；

```
RetinaNet - Head    Anchorbased     没有quality分支
FCOS - head   Anchor-free 有quality分支（centerness）
```

漏检 & 小物体

- **只有在precision得到保证条件下讨论recall才有意义**
- **原因：**将图像的分辨率从600×600降低到约30×30：他们在第一层提取的**小目标特征**(一开始就很少)在网络中**间的某个地方"消失"了，从来没有真正到达检测和分类步骤中**；

数据集：

- **长尾分布**：旷视的**BBN架构**，用于长尾识别的tricks大礼包：**Input Mixup** + DRS（**基于Re-Sampling的Re-Balancing**）+ CAM-BS（基于Class Activation Maps的**数据增广**，Balance-Sampling） **+ Fine-Tuning**；
- 取一些小目标样本**随机粘贴**在背景区域中；
- 对样本少的类别，进行**样本扩增**，考虑新增样本检测背景和样本多样性；
- 更大的**输入分辨率**：使用GAN对小物体进行超分辨；
- **将图像分成小块进行检测**；

网络架构：

- **Anchor的设计**，可以很好地引入样本少的类别的bbox分布先验（k-means来设置anchor），使生成的Pred boxes在这些类别有较高的Recall；
- **FPN / Multi-scale Training & Testing**（图像金字塔-多分辨率图像，特征金字塔-FPN）：**漏检也很大程度上来自于目标尺度的巨大变化**，比如自动驾驶场景中，同是Car类别的bbox样本在远处只占据极小部分pixels，而在近处甚至占据近半幅图像；
- **扩大感受野：**使用不同ratio的dilated conv分别匹配不同的感受野，检测不同尺度目标；[1, 3, 5, 1, 3, 5]
- **利用时间维度**：如果我们有一个来自静止摄像机的视频，我们需要检测它上面的移动物体，比如足球，我们可以利用图像的时间特性。例如，我们可以做背景减法，或者仅仅使用后续帧之间的差异作为一个(或多个)输入通道，网络将**为移动目标创造更"强大"的特性**，而这些特性不会消失在池化和大stride的卷积层中；

损失函数：

- **Focal Loss**
- **OHEM**：由于易漏检样本通常在训练过程中较容易产生较大的loss，那么在每次训练迭代中，将loss值较大的的样本筛选出来加入训练集进入下一次迭代，使得检测模型更多关注易漏检样本；

后续处理：

- **DIoU NMS**用于漏检；（NMS中DIoU替换IoU，直接归一化中心点距离，在IoU中如果物体有重叠，在NMS过程中会删除）
- **上下文很重要**，利用它更好地找到小物体：Finding Tiny Faces - https://arxiv.org/pdf/1612.04402.pdf

---

mAP是否科学

作为一个要落地的模型，mAP只是众多标准的一个，测速也要实际部署了才有比较的意义

> 实际业务通常不会选用mAP来衡量一个detector的性能，一般用FPPI（每张图片错几个），或者相同Recall下比较Precision
>
> 1.应用场景中一般0.5的IoU足够了，并不需要过度严格的指标
> 2.AP会被一些涨recall的方法推上去，比如用soft-nms，focal loss等方法测试或训出来的模型Recall会很高，mAP相应的通常会涨一点，但是都是涨的低Precision的区域，低精度区对应用场景来说没用，一般用的时候都是卡高Precision，涨回来的Recall其实并没有什么用；
> 3.应用的时候会卡单一的阈值，比如0.5，mAP对阈值做了平均，这时候就更不能用了；
> 4.降低IoU阈值：采用0.05的确能够mAP涨点，但是，指标虽然上去了，预测结果却变得有点糟糕了，结果中多了些不是很好的框；

目标检测方向

- 高精度检测
- 小目标检测
- 稠密目标检测
- 长尾分布

Transformer目标检测落地优缺点

工业界已经有不少OCR算法用Transformer做文字公式识别，像图像生成，匹配等比较细分的领域效果也很好，CNN大约只有Transformer 10～25的性能，但是目标检测落地还没有看到有用transformer；

两个网络的capacity和generality不同；

缺点：

- **一堆trick**，不知道是网络创新起作用，还是其他细节起作用：BERT 的激活函数是 GELU 而不是 transformer 原论文中的 ReLU；
- Transformer训练耗时耗数据；

- **Transformer的确是更吃数据**，用于cnn的augmentation不足以train好ViT：有博主做过实验：用mixup=0.8这样强的augmentation在任何cnn上都会掉点，ImageNet上用AdamW去训练ResNet也会产生严重掉点，但是这些trick在Transformer上都能涨点；

优点：

- 相同FLOPs下的参数量更少（3*3卷积替换成MHSA，参数量显然减少了）
- ViT和ResNet50的FLOPs相同，快50% （计算全局注意力时，key set共享，对显存访问友好）
- CNN遮挡30%基本就凉了，Transformer甚至能搞定擦除50%的数据

Transformer为什么更吃数据

- 去掉了先验（二维变1维）
- 感受野计算其实是根据内容动态变换的，所以可表征的空间，比有限权重的cnn多得多（但是最近两年CNN领域有动态网络,如果卷积核根据内容动态生成,动态cnn的capacity和generality不一定差于transformer）

##

YOLO系列是基于深度学习的回归方法；Faster-RCNN是基于深度学习的分类方法；

https://zhuanlan.zhihu.com/p/136382095

YOLO v1

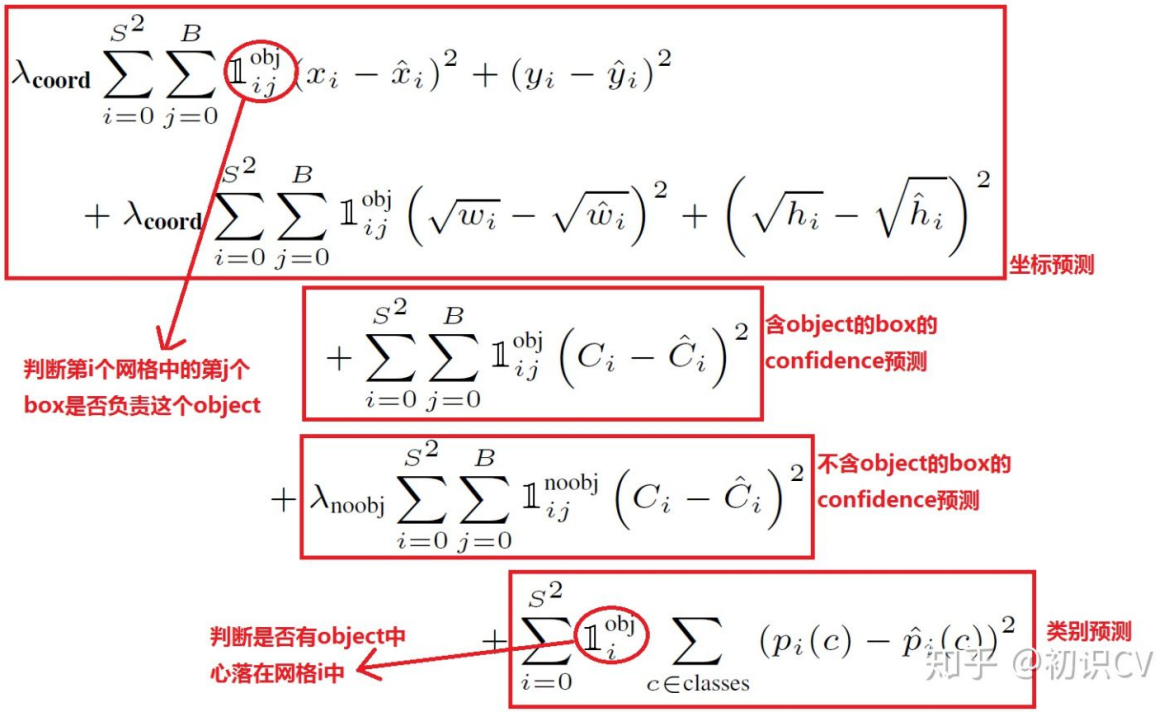
YOLOv1: You Only Look Once: Unified, Real-Time Object Detection.

Pipline:

- 将一幅图像分成SxS个网格，如果某个object的中心落在这个网格中，则这个网格就负责预测这个object;
- 每个网格需要预测B个BBox的位置和置信度，一个BBox对应着四个位置和一个confidence，confidence代表了所预测的box中含有object的置信度[0, 1]和这个box预测的有多准（IoU）：
- 每个bounding box要预测(x, y, w, h)和confidence共5个值，每个网格还要预测一个类别信息，记为C类；则SxS个网格，每个网格要预测B个bounding box还要预测C个categories。输出就是S x S x (5*B+C)的一个tensor。（**注意：class信息是针对每个网格的，confidence信息是针对每个bounding box的**）
- 在test的时候，每个网格预测的class信息和bounding box预测的confidence信息相乘，就得到每个bounding box的class-specific confidence score，这个乘积即encode了预测的box属于某一类的概率，也有该box准确度的信息
- 得到每个box的class-specific confidence score以后，设置阈值，滤掉得分低的boxes，对保留的boxes进行NMS处理，就得到最终的检测结果；

`(x, y)`是边界框的中心坐标，而`w`和`h`是边界框的宽与高，还有一点要注意，中心坐标的预测值`(x, y)`是相对于每个单元格左上角坐标点的偏移值，并且单位是相对于单元格大小的，而边界框的`w`和`h`预测值是相对于整个图片的宽与高的比例；

Yolo算法将目标检测看成回归问题，所以采用的是均方差损失函数，但是对不同的部分采用了不同的权重值，对于`(w, h)`如果不取根号，损失函数往往更倾向于调整尺寸比较大的预测框，取根号是为了尽可能的消除大尺寸框与小尺寸框之间的差异；



优点：

- 快速，pipline简单
- 采用全局特征进行推理，由于利用全局上下文信息，相比于滑动窗口和建议框方法，对背景的判断更准确
- 泛化性强：YOLO对于艺术类作品中的物体检测同样适用

缺点：

- 由于输出层为全连接层，因此在检测时，YOLO训练模型只支持与训练图像相同的输入分辨率
* 物体定位不准确，虽然loss采用长宽平方根进行回归，试图降低大目标对loss的主导地位，但小目标的微小偏差对IOU的影响更严重，导致小目标定位不准
- 召回率低(置信度标签用IoU则学习时候始终很小，无法有效学习，YOLOv3使用1，置信度意味着该预测框是或者不是一个真实物体，是一个二分类，所以标签是1、0更加合理)
- 不管一个单元格预测多少个边界框，其只预测一组类别概率值，一个单元格有多个目标无法检测出来，同时该网络最多预测SxS个物体

- [x] https://zhuanlan.zhihu.com/p/362758477
- [x] https://zhuanlan.zhihu.com/p/32525231
- [x] https://zhuanlan.zhihu.com/p/70387154

YOLO v2 / 9000

YOLOv2 (YOLO9000: Better, Faster, Stronger)

核心点：

- BN / Anchor（K-means） / Darknet-19 / 约束预测边框的位置

$$
\begin{aligned}
b_x &= \sigma\left(t_x\right)+c_x \\
b_y &= \sigma\left(t_y\right)+c_y \\
b_w &= p_w e^{t_w} \\
b_h &= p_h e^{t_h} \\
\Pr(\text{ object }) * \mathrm{IOU}(b, \text{ object }) &= \sigma\left(t_o\right)
\end{aligned}
$$

YOLOv2预测边界框中心点相对于对应cell左上角位置的相对偏移值，为了将边界框中心点约束在当前cell中，使用sigmoid函数处理偏移值，这样预测的偏移值在(0,1)范围内；其中，b_x, b_y, b_w, b_h 是预测边框的中心和宽高， $\Pr(\text{ object }) * IOU(b, \text{ object })$ 是预测边框的置信度，这里对预测参数 t_o 进行sigmoid变换后作为置信度的值， c_x, c_y **是当前网格左上角到图像左上角的距离（cell的左上角坐标），** p_w, p_h 是先验框的宽和高， t_x, t_y, t_w, t_h, t_o 是要学习的参数，分别用于预测边框的中心和宽高，以及置信度；（在Yolov1中，网络直接回归检测框的宽、高，这样效果有限）

- 采用Multi-Scale Training：YOLO v2每10个Batches，网络会随机地选择一个新的图片尺寸，由于使用了下采样参数是32，所以不同的尺寸大小也选择为32的倍数{320，352.....608}，最小320x*320*，*最大608*x608，网络会自动改变尺寸，并继续训练的过程；
- **WordTree** + 提出了一种分类和检测的**联合训练**策略：通过将两个数据集混合训练，**如果遇到来自分类集的图片则只计算分类的Loss，遇到来自检测集的图片则计算完整的Loss**
- v2和v1一样都采用**MSE Loss**
- 对于训练图片中的ground truth，**若其中心点落在某个cell内，那么该cell内的5个先验框所对应的边界框负责预测它**，具体是由那个与ground truth的IOU最大的边界框预测它，而剩余的4个边界框不与该ground truth匹配

- [x] https://zhuanlan.zhihu.com/p/35325884
- [x] https://zhuanlan.zhihu.com/p/362759621

YOLO v3

YOLOv3: An Incremental Improvement

核心点：

- Darknet53
- YOLOV3是将输入图像分成SxS个格子，并有三处进行检测，分别是在52x52, 26x26, 13x13的feature map上，若某个物体Ground truth的中心位置的坐标落入到某个格子，那么这个格子就负责检测中心落在该栅格中的物体，**三次检测：**每次对应的感受野不同，**32倍降采样的感受野最大**

（13x13），适合检测大的目标，；**16倍（26x26）适合一般大小的物体**；8**倍的感受野最小（52x52），适合检测小目标**；

- **YOLO v3训练，不再按照ground truth中心点，严格分配指定cell，而是根据预测值寻找IOU最大的预测框作为正例**；
- 置信度标签：Yolov1中使用IoU，Yolov3使用1或0（置信度意味着该预测框是或者不是一个真实物体，是一个二分类）；
- 正例产生置信度loss、检测框loss、类别loss；**忽略样例不产生任何loss**；负例只有置信度产生loss；
- 除预测框仍使用MSE外，其他**置信度、类别使用交叉熵误差（二分类）**；
- 类别置信度使用Sigmoid代替v2中的Softmax，取消了类别之间的互斥（person和man）；

忽略样例：

忽略样例是Yolov3中的点睛之笔：由于Yolov3使用了多尺度特征图，不同尺度的特征图之间会有重合检测部分。比如有一个真实物体，在训练时被分配到的检测框是特征图1的第三个box，IOU达0.98，此时恰好特征图2的第一个box与该ground truth的IOU达0.95，也检测到了该ground truth，如果此时给其置信度强行打0的标签，网络学习效果会不理想，最终loss会变成Loss $_{\text{obj}}$ 与 Loss $_{\text{noob}\ j}$ 的拉扯，不管两个loss数值的权重怎么调整，网络要么预测趋向于大多数预测为负例，要么趋向于大多数预测为正例。而加入了忽略样例之后，网络才可以学习区分正负样例；

优点：

```
- 实时性：YOLOv3通过牺牲检测精度，使用Darknet53而不是Resnet101，从而获取更快的检测速度
- 多尺度：相比于YOLOv1-v2，与RetinaNet采用相同的FPN网络作为增强特征提取网络得到更高的检测精度
- 目标重叠：通过使用logistic + binary cross-entropy loss进行类别预测，将每个候选框进行多标签分类，解决单个检测框可能同时包含多个目标的可能
```

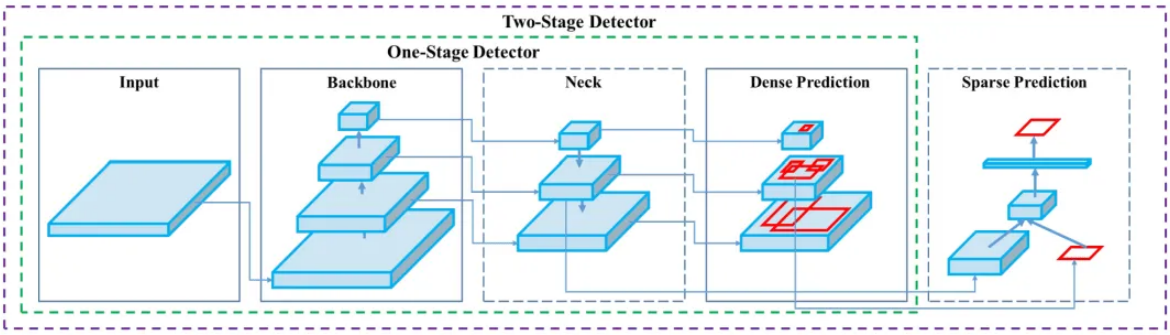
缺点：

```
- 准确率：主要因为Darknet的特征提取不够强，未进行精细化结构模型设计
```

- [x] https://zhuanlan.zhihu.com/p/76802514
- [x] https://zhuanlan.zhihu.com/p/362761373

YOLO v4

YOLOv4: Optimal Speed and Accuracy of Object Detection



two-stage的检测网络，相当于在one-stage的密集检测上增加了一个稀疏的预测器

- **CSPDarkNet53 / SPP / PAN：**
- **不影响推理耗时**：
  - 光度变换：亮度，对比度，色相，饱和度
  - 几何变换：随机缩放，剪裁，翻转，缩放
  - 图像融合：CutMix，Mosaic
  - 正则化方法：DroupBlock（Dropblock则是将Cutout应用到网络中的每一个特征图上），CmBN
  - 处理数据不平衡：Focal Loss， Label Smoothing
  - Bbox Loss：CIoU-Loss
  - Eliminate grid sensitivity
  - multiple anchors for a single ground truth
- **提高了推理耗时**：
  - 增强感受野：SPP
  - 注意力：SAM，
  - 特征融合：PAN，CSP， Multiinput Weighted Residual Connectins(MiWRC)
  - 激活函数：Mish
  - 后续处理： DIoU NMS
- **CSP跨阶段局部网络：**其解决了其他大型卷积网络结构中的重复梯度问题，减少模型参数和FLOPS，即保证了推理速度和准确率，又减小了模型尺寸
- bounding box regression使用CIoU，置信度损失和分类损失仍然使用BCE
- 因为CIOU_Loss中包含影响因子v，涉及groudtruth的信息，而测试推理时，是没有groundtruth的，**YOLO v4在DIOU loss的基础上采用DIOU NMS**，可以用来解决遮挡问题

优点：

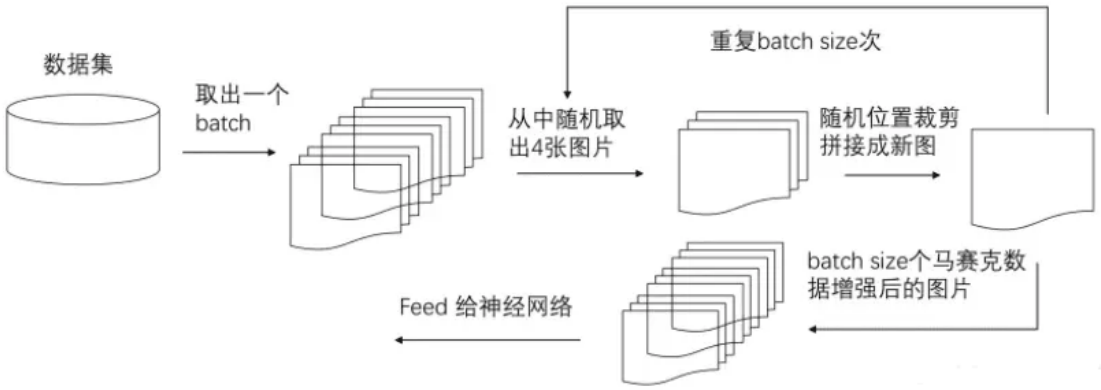
```
- 实时性：借鉴CSPNet网络结构将Darknet53改进为CSPDarknet53使模型参数和计算时间更短
- 多尺度：Neck分别引入PAN和SPP网络结构作为增强特征提取网络，能够有效多尺度特征，相比于引入FPN网络准确度更高
- 数据增强：引入Mosaic数据增强，在使用BN的时候可以有效降低batch_size的影响
- 模型训练，采用CIoU作为目标框的回归，与YOLOv3使用的平方差损失相比具有更高的检测精度
```

- [x] https://zhuanlan.zhihu.com/p/150127712

YOLO v5

核心点：

- Mosaic / 自适应锚框 / 自适应图片缩放 / Focus / PAN / SPP / CSP / CIoU



- CSP： Yolov4中只有主干网络使用了CSP结构，Neck结构中，采用的都是普通的卷积操作；而Yolov5中设计了两种CSP结构，**CSP1_X结构**应用于**Backbone主干网络**，另一种**CSP2_X**结构则应用于**Neck**中；
- **YOLO v4在DIOU loss的基础上采用DIOU NMS**，可以用来解决遮挡问题，而**Yolov5中采用加权NMS的方式**
- **depth_multiple**，**width_multiple**控制网络的宽度和深度：**四种网络结构中每个CSP结构的深度都是不同的**

☑ https://zhuanlan.zhihu.com/p/172121380

YOLOF

YOLOF：You Only Look One-level Feature（CVPR 2021）

核心点：

- FPN的关键在于使用“**分而治之**”思路解决稠密目标检测，而非多尺度特征融合
- **SiMo Encoder**仅仅采用C5特征且不进行特征融合即可取得与MiMo编码器相当的性能
- **SiSo Encoder性能下降**的两个重要原因：1.C5特征图感受野有限 2.positive anchors不平衡 -> 从优化的角度出发，引入了**Dilated Encoder**提升特征感受野，**Uniform Matching**进行不同尺度目标框的匹配，从而可以**仅仅采用一级特征进行检测（C5）**
- **当作者采用SiSo Encoder时，anchor的数量会大量的减少**,若使用`Max-IoU Matching`选择`positive anchor`，`ground truth`尺寸大的目标产生的`positive anchor`要多于`ground truth`尺寸小的目标产生的`positive anchor`，这种现象会导致网络在训练时更关注大尺寸的目标，忽略小尺寸目标；`Uniform Matching`策略：**对于每个目标框采用k近邻锚点作为正锚点，这就确保了所有的目标框能够以相同数量的正锚点进行均匀匹配。**正锚点的平衡确保了所有的目标框都参与了训练且贡献相等；

YOLOX

- **Anchor Free + Multi positives**:
  - 将每个位置的预测从3下降为1并直接预测四个值：即每个location或者grid只预测一个box，两个offset以及高宽（xywh），对于一个给定的gt边界框，首先根据它的size确定所匹配的尺度，然后计算它在这个尺度上的中心点位置，计算中心点偏差 t_x, t_y ，至于宽高 wh ，就直接作为回归目标；
  - **参考FCOS，**我们将每个**目标的中心定位正样本**并**预定义一个尺度范围**以便于对每个目标指派FPN进行检测；
  - 之前只考虑中心点所在的网格，这会改成以中心点所在的网格的3x3邻域，都作为正样本，直观上来看，正样本数量增加至9倍。每个grid都去学到目标中心点的偏移量和宽高，此时，显然这个中心点偏移量不再是01了；
- **Mosaic和Mix-up**，不使用RandomResizedCrop（和Mosaic有点重合了）：**这两个数据增强方式会使得数据集分布有明显的改变，训练最后15Epoch关闭Mosaic和Mixup使得模型收敛于数据集；**
- cls用于分类但不用于划分正负样本，正负样本交给obj branch；
- **预测部分**加入了**IoU-aware**分支，和PP-YOLO是对齐的
  - Yolox采取的是iou aware与cls分支融合的方式(看代码，cls的gt不是1，而是iou)

- **Decoupled Head分类与定位头的解耦**：之前的YOLO模型的classification和regression都是在一起的，分开的话会取得更好的效果，收敛速度也更快（不同level的feature不共享head，和RetinaNet有区别），**适合NMS-Free**

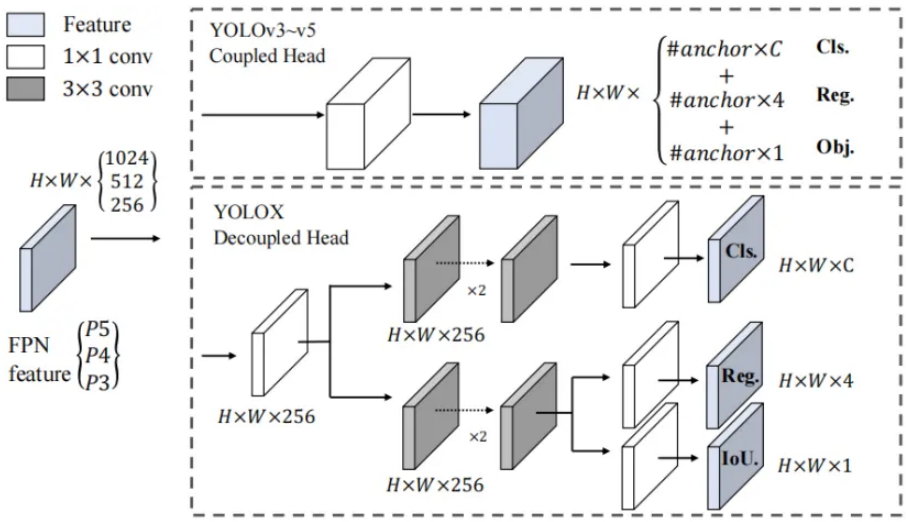
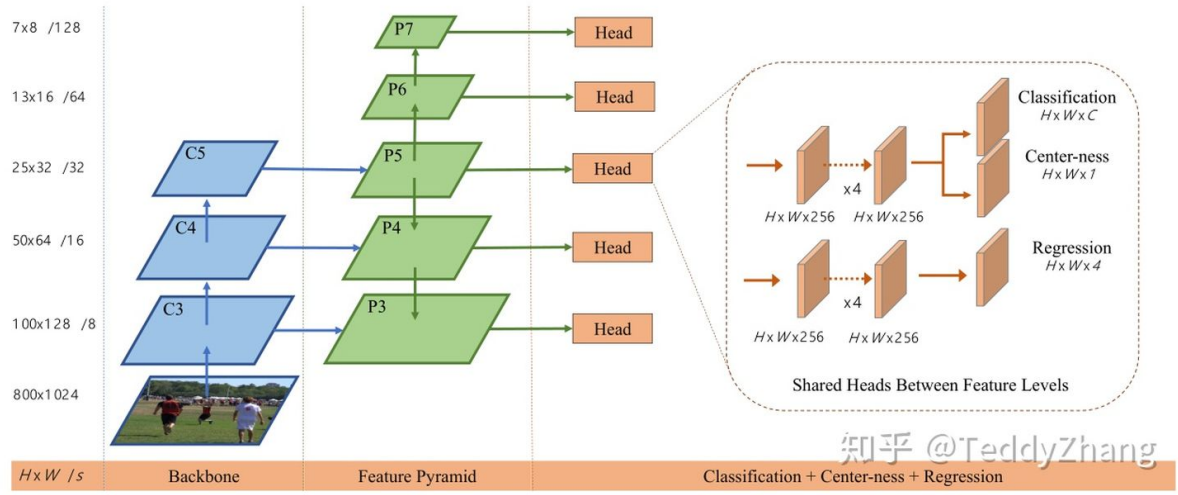


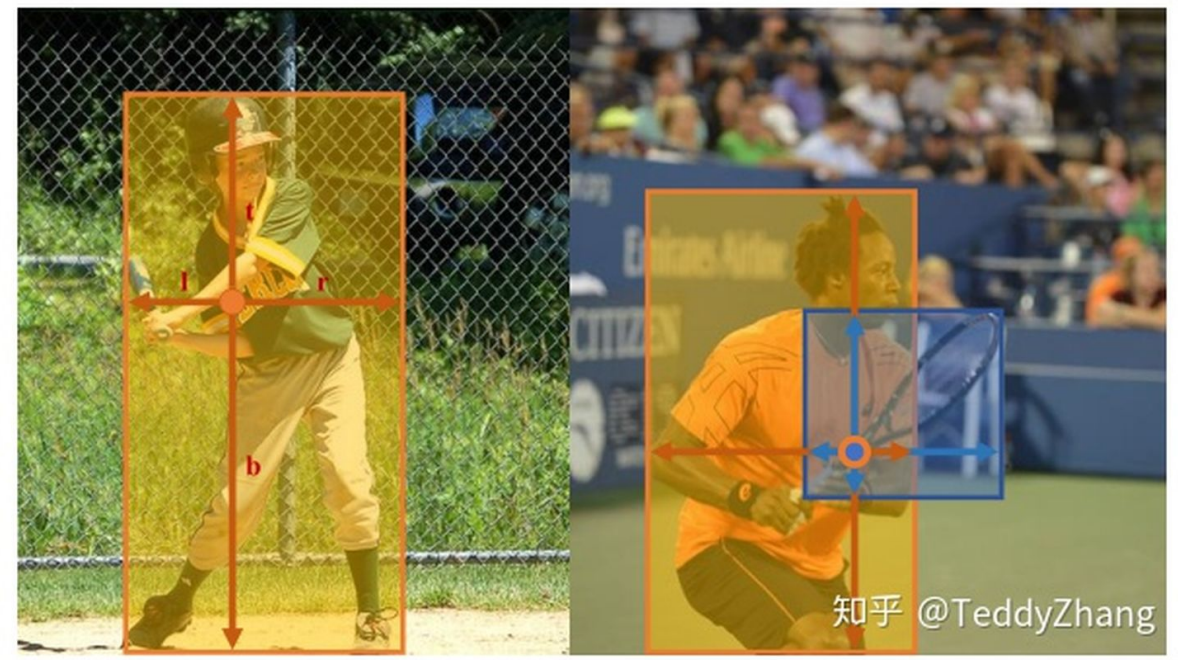
Figure 2: Illustration of the difference between YOLOv3 head and the proposed decoupled head. For each level of FPN feature, we first adopt a 1×1 conv layer to reduce the feature channel to 256 and then add two parallel branches with two 3×3 conv layers each for classification and regression tasks respectively. IoU branch is added on the regression branch.

- **SimOTA：**对任何一个gt，根据最小cost选择限定中心区域的**top k predictions作为正样本** - 使用one-to-many策略；

- 标签分配的四个关键因素：(1) loss/quality aware; (2) center prior; (3) dynamic number of positive anchors; (4) global view。OTA满足上述四条准则，因此我们选择OTA作为候选标签分配策略 -> **OTA从全局角度分析了标签分配并将其转化为最优运输问题**取得了SOTA性能，然而最优运输问题优化会带来25%的额外训练耗时。因此，我们**将其简化为动态top-k策略以得到一个近似解(SimOTA)**,SimOTA不仅可以**降低训练时间**，同时可以**避免额外的超参**问题；
- 使用SimOTA之后，FCOS样本匹配阶段的FPN分层就被取消了，匹配(包括分层)由SimOTA自动完成；

FOCS





语义分割的思想逐像素点的来解决检测

正负样本：

如果一个location(x, y)落到了任何一个GT box中，那么它就为正样本并标签为类别 c 用于分类问题，我们还可以得到一个4D的向量 (l^*, t^*, r^*, b^*), 也就是这个 点到左、上、右、下边的距离

因此根据我们训练目标，最后一层的网络输出为一个80D的向量 p 用于分类层, 4D的向量也就是 $(\mathrm{l}, \mathrm{t}, \mathrm{r}, \mathrm{b})$, 用于回归层，和RetinaNet一样，我们使用4个卷积层的backbone用于分类和回归, 接着由于回归目标 $(\mathrm{l}, \mathrm{t}, \mathrm{r}, \mathrm{b})$ 都是正的, 作者使用exp()将其映射到范围 (0, 正无穷)

损失函数分为三个分支：

- centerness，CE loss
- 对于分类层使用的为focal loss用于平衡样本
- 而回归层使用IOU loss

1 - FCOS通过限定不同特征图的边界框的回归范围来进行分配：作者为了减少尺度差异大的物体重叠，引入参数 m_i 为特征层 i 的最大距离，如果一个 location (x, y) 满足 $\max(l^*, t^*, r^*, b^*) > m_i$ 或者 $\max(l^*, t^*, r^*, b^*) < m_{i-1}$, 那么我们在这个特征层就将其视为负样本，不进行回归，其中 m_i 分别设置为0,64,128,256,512和正无穷，正好可以形成5个区间，在5个层上进行限制尺寸以减少重叠区域；

2 - 如果**一个像素点在同一层落到了多个GT区域，那么我们就取最小的那个当做回归目标**（利用FPN进行多尺度预测会极大地减少这种情况的发生）

FCOS距离目标中心较远的位置产生很多低质量的预测边框：这是由于我们把中心点的区域扩大到整个物体的边框，经过模型优化后可能有很多中心离GT box中心很远的预测框，为了增加更强的约束，作者提出了Center-ness的方法来解决这个问题，Center-ness取值为0,1之间，使用交叉熵损失进行训练

测试时，将预测的中心度与相应的分类分数相乘，计算最终得分

$$\text{centerness}^* = \sqrt{\frac{\min(l^*, r^*)}{\max(l^*, r^*)} \times \frac{\min(t^*, b^*)}{\max(t^* b^*)}}$$

中心度可以降低远离对象中心的边界框的权重：当loss越小时，centerness就越接近1，也就是说回归框的中心越接近真实框。那么为什么要使用这么大的区域呢？作者解释，从anchor-based角度，我们通过两个IOU阈值将标定的anchor分为negative、ignored和positive三种，这样不能充分的利用正样本。而在FCOS中，只要没有location没有落在GT Box区域，那么就是负样本，反之为正样本。增大区域为了获得更多的正样本以更好地用于回归器中。

改进

```
1.centerness分支与cls分支共享前面几个卷积，目标的 centerness 值比较小，最终cls分数很容易被阈值卡掉 -> centerness分支与reg分支共享前面几个卷积 :0.5 mAP
2.GT bbox内的点，分类时均作为正样本 -> 只有GT bbox中心附近的一定范围内的小bbox内的点，分类时才作为正样本
3.bbox loss weight：所有正样本点的 weight 平权 -> 将样本点对应的 centerness 作为权重，离 GT 中心越近，权重越大
4.IoU loss -> GIoU loss -> CIoU loss
5.centerness 分支利用 l，t，r，b 计算 centerness -> 直接用 IoU
```

CornerNet

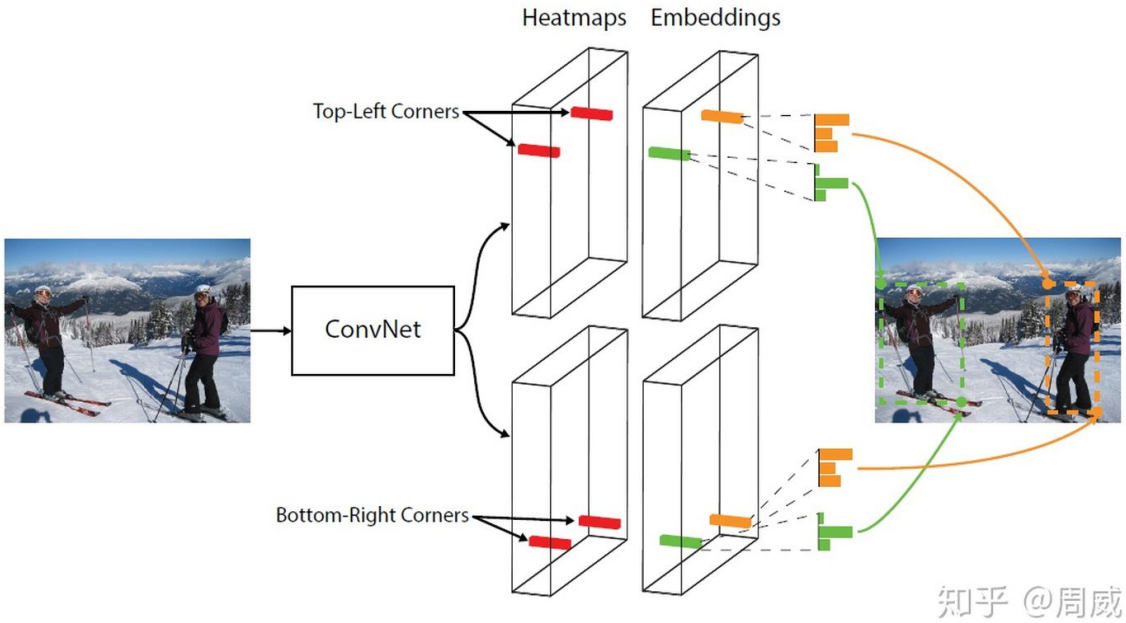
CornerNet 只需要预测物体包围框的左上点坐标（top-left corner）和右下角坐标，那么就可以完成对物体的检测了，那么原来需要设置很多anchor进行 **region proposal** 的方法变成了**一对对关键点**的检测了

主要创新

- Anchor-Free 的目标检测新思路
- Corner Pooling的提出
- CornerNet网络的提出

如何匹配同一物体bounding box的左上角和右下角

> `cornerNet`在进行预测的时候，会为每个点分配一个`embedding vector`，属于同一物体的点的`vector`的距离较小



Corner Pooling

- **目的**是为了建立Corner和目标的位置关系，用于定位顶点，一般而言，知道了bounding box的左上角和bounding box的右下角就可以确定位置所在的**范围和区域**
- 当求解某一个点的 top-left corner pooling时 ，就是以该点为起点，水平向右看遇到的最大值以及竖直向下看最大的值之和
- 实现：方向颠倒后，我们每次都将**沿着该方向**上**遇到**的**最大的值**作为**填充值**即可快速实现 corner pooling（log(1)时间内获得栈最小元素思路一样）
- 论文认为corner pooling之所以有效，是因为（1）目标定位框的中心难以确定，和边界框的4条边相关，但是每个顶点只与边界框的两条边相关，所以corner 更容易提取（2）顶点更有效提供离散的边界空间，实用O(wh)顶点可以表示O(w2h2) anchor boxes

网络输出及作用

- **heatmaps预测定位框的顶点对（左上角和右下角）热点图（batch size,128,128,80） Focal Loss**

top-left corners和bottom-right corners的两类heatmaps，每类heatmaps有c个channals(c是种类数量，没有背景类)，大小为HxW，每个channel是二进制掩膜，表示的是相应类别的顶点位置和置信度信息

对于每个顶点，只有一个ground-truth，其他位置都是负样本。在训练过程，模型减少负样本，在每个ground-truth顶点设定半径r区域内都是正样本，这是因为落在半径r区域内的顶点依然可以生成有效的边界定位框，论文中设置IoU=0.7

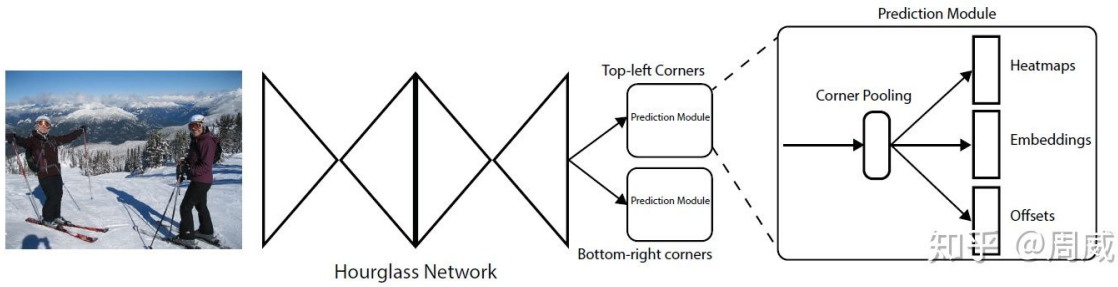
- **embeddings（batch size,128,128,1）pull_loss + push_loss**

衡量左上corner和右下corner的距离的，从而判断某一对角点是否属于同一个物体的两个角点

- **offsets（batch size,128,128,2） regr_loss（smooth_l1_loss）**

主要解决downsample之后，upsample造成的精度损失，这会影响到小物体位置框的位置精度，所以offsets会缓解这种问题

将上面提到的**精确映射位置（浮点型）**与**真实映射位置（整型）**进行相减的值作为监督信息即可



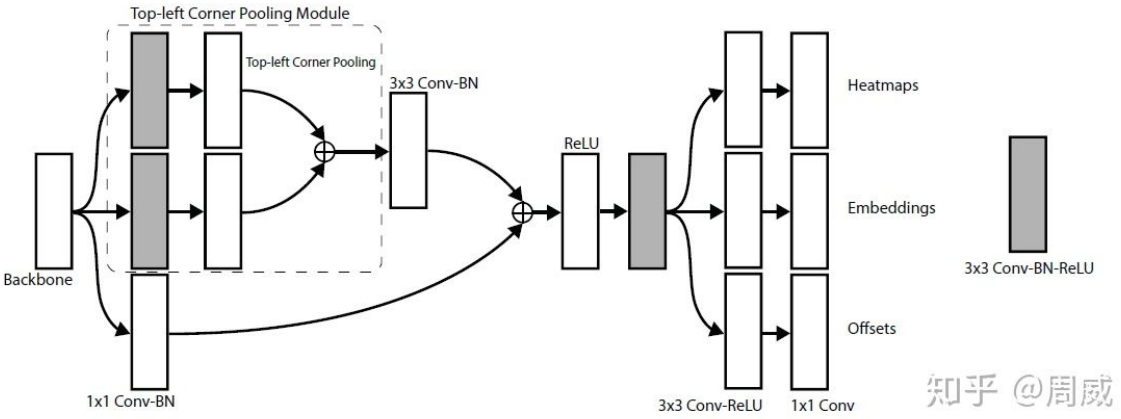
cornerNet的网络结构主要分为以下几个部分：

- backbone: hourglass Network
- head: 二分支输出 Top-left corners 和 Bottom-right corners，每个分支包含了各自的corner pooling以及三分支输出

通过两个hourglass module后的特征图，需要各自再通过一个3x3卷积后才能获得用于预测左上点和右下点的两个分支module（prediction module)后，每个predicition module分别进行如下操作

- corner pooling
- 三分支的输出

prediction module具体实现



RetinaNet

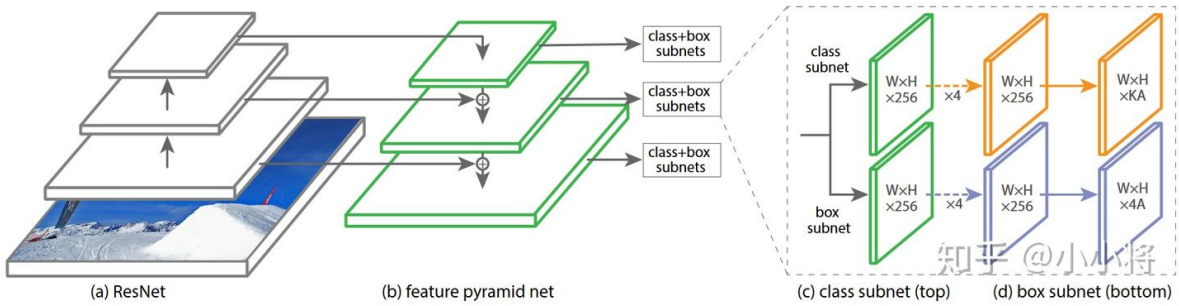
RetinaNet = ResNet + FPN + Two sub-network + Focal Loss

主要优点
1.多尺度：借鉴FPN网络通过自下而上、自上而下的特征提取网络，并通过无代价的横向连接构建增强特征提取网络，利用不同尺度的特征图检测不同大小的目标，利用了底层高分率的特征图有效的提高了模型对小尺度目标的检测精度
2.样本不平衡：引入Focal Loss用于候选框的【类别预测】，克服正负样本不平衡的影响及加大困难样本的权重

主要不足
1.实时性：网络使用ResNet-101作为主干特征提取网络，检测效率略微不足
2.由于采用的K个二分类，某个位置的某个anchor可能最后会输出几个类别不同但是box一样的detections

类别不平衡（class imbalance）是目标检测模型训练的一大难点，其中最严重的是**正负样本不平衡**（1.正负样本固定比例抽样 2.OHEM ）；SSD的策略是采用hard mining，从大量的负样本中选出loss最大的topk的负样本以保证正负样本比例为1:3

RPN本质上也是**one-stage检测模型**，RPN训练时所采取的策略也是抽样，从一张图像中抽取固定数量N（RPN采用的是256）的样本，正负样本分开来随机抽样N/2，**如果正样本不足，那就用负样本填充；**



FPN

RetinaNet特征为P_3, P_4, P_5,去掉了P_2（stride=4，特征图很大，去掉它可以减少计算量），同时新增两个特征P_6, P_7，P_6 在 P_5 上加一个stride=2的3x3 卷积得到, P_7 是在 P_6 后面加ReLU和一个stride=2的3x3卷积得到；这样RetinaNet的 backbone得到特征也是5个level, 分别为 P_3, P_4, P_5, P_6, P_7, 其stride分别为 $8, 16, 32, 64, 128$；

Anchor

- RetinaNet输入特征 P_3, P_4, P_5, P_6, P_7每个Level的anchor：$32^2, 64^2, 128^2, 256^2, 512^2$；
- 每个level的每个位置放置3种scale的anchor$\left\{2^0, 2^{1/3}, 2^{2/3}\right\}$，设置3种长宽比$\{1:1, 1:2, 2:1\}$，这样每个位置共有A=9个anchor；
- scale表示尺度，ratio表示anchor长宽比例：**一般来说anchor面积固定，但是这个面积有不同的长宽比例（ratio）**，RetinaNet对于固定的特征图有三种不同的scale，则**特征图有三种不同的anchor面积，同时有3种不同的ratio，则一共有9种组合**；

训练过程：计算anchor与所有GT的IoU，取IoU最大值，若大于0.5，则认为此anchor为正样本，且负责预测IoU最大的那个GT；若低于0.4，则认为此anchor为负样本；若IoU值在$[0.4, 0.5]$之间，则忽略不参与训练

Detection

检测模块主要包括**分类分支(Focal Loss)和box回归分支(Smooth L1 Loss)**

> 分类分支用来预测每个位置的各个anchor(数量为A) 的类别概率（类别数为K)：分类分支包括4个3x3的卷积(ReLU激活函数，channel是256)， 输出channel为KA， 最后sigmoid激活就可以得到各个anchor预测每个类别的概率，对于 RetinaNet来说，每个位置相当于KA个二分类问题；

> box回归分支用来预测每个位置各个anchor和GT之间的 offset：box回归分支与分类分支类似，只不过最后输出channel是4A，这也表明RetinaNet的box回归是类别无关的；

detection模块在FPN各个 level的特征是参数共享的，这点和RPN类似，但是RetinaNet的detection模块是多分类的，而且无法使用BN；

分类分支最后的卷积层的偏值 b 初始化为：

$$b = -\log\left(\frac{1-\pi}{\pi}\right)$$

这个 π 相当于是为模型训练开始时**每个anchor预测为正例设置一个先验概率值**，论文中采用的是0.01，只用这一条策略，基于ResNet50的RetinaNet在COCO上的AP值就能达到30.2。**这是因为很多anchor是负例，设置先验值可以大大降低负样本在开始训练时的loss, 这样训练更容易, RetinaNet很容易loss出现nan。**

训练与测试

```
分类loss是sum所有的focal loss，然后除以类别为正例的anchors总数

在inference阶段：
 - 对各个level的预测首先取top 1K的detections，然后用0.05的阈值过滤掉负类，此时得到的detections已经大大降低；
 - 再对detections的box进行解码而不是对模型预测所有detections解码可以提升推理速度；
 - 最后把level的detections结果concat在一起，通过IoU=0.5的NMS过滤重叠框就得到最终结果；
```

EfficientDet

SSD

```
1、实时性：相比YOlOv1更快，去除了全连接层
2、标签方案：通过预测 类别置信度 和相对固定尺度集合的 先验框的偏差，能够有效均衡不同尺度对loss的影响程度
3、多尺度：通过使用多个特征图和对应不同尺度的锚框进行多尺度目标预测
4、数据增强：通过随机裁剪的方式进行数据增强提高模型的鲁棒性
4、样本不平衡：通过困难样本挖掘，采用负样本中置信度最高的先验框进行训练，并设置正负样本比例为1：3，使得模型训练收敛更快

主要不足
1、通过人工先验设置的不同尺度的锚框无法适应真实的目标框的尺度分布
2、使用的多个特征图由于高分辨率的特征图不能有效地结合高层特征


改进：
1.FPN
2.更大输入尺寸
3.k-means聚类anchor尺寸
```

SSD的检测：

对于**每个单元的每个先验框，其都输出一套独立的检测值**，对应一个边界框，主要分为两个部分：

- 第一部分是**各个类别的置信度**，值得注意的是**SSD将背景也当做了一个特殊的类别**，如果检测目标共有c个类别，SSD其实需要预测$c+1$个置信度值，其中第一个置信度指的是不含目标或者属于背景的评分；**在预测过程中，置信度最高的那个类别就是边界框所属的类别**，特别地，**当第一个置信度值最高时，表示边界框中并不包含目标**；
- 第二部分就是**边界框的location**，包含4个值(cx, cy, w, h)，分别表示边界框的中心坐标以及宽高。但是真实预测值其实只是边界框相对于先验框的转换值.先验框位置用$d=\left(d^{cx}, d^{cy}, d^{w}, d^{h}\right)$表示，其对应的边界框用$b=\left(b^{cx}, b^{cy}, b^{w}, b^{h}\right)$，那么边界框的预测值$l$其实是$b$相对于$d$的转换值：

$$l^{cx}=\left(b^{cx}-d^{cx}\right)/d^{w}, l^{cy}=\left(b^{cy}-d^{cy}\right)/d^{h}$$
$$l^{w}=\log(b^{w}/d^{w}), l^{h}=\log\left(b^{h}/d^{h}\right)$$

- **检测值包含两个部分：类别置信度和边界框位置，各采用一次$3*3$卷积来进行完成；**综上所述，对于一个大小$m*n$的特征图，共有mn个单元，每个单元设置的先验框数目记为k，那么每个单元共需要$(c+4)k$个预测值，所有的单元共需要$(c+4)kmn$个预测值，由于SSD采用卷积做检测，所以就需要$(c+4)k$个卷积核完成这个特征图的检测过程;

提取了6个特征图，其大小分别是 $(38,38),(19,19),(10,10),(5,5),(3,3),(1,1)$，但是**不同特征图设置的先验框数目不同**；

- 对于**先验框的尺度**，随着特征图大小降低，先验框尺度线性增加（**网络越深，感受野越大，能检测越大尺寸目标**），各个特征图的先验框尺度为$30, 60, 111, 162, 213, 264$ （第一层单独设置，后边尺度遵守一个线性递增规则）
- 对于长宽比，一般选取$a_r \in \left\{1, 2, 3, \frac{1}{2}, \frac{1}{3}\right\}$，**每个单元的先验框的中心点分布在各个单元的中心**

训练过程：

- 先验框匹配

在训练过程中，首先要确定训练图片中的ground truth与哪个先验框来进行匹配，与之匹配的先验框所对应的边界框将负责预测它

> 1.对于图片中每个ground truth，找到与其IOU最大的先验框，该先验框与其匹配，这样，可以保证每个ground truth一定与某个先验框匹配，反之，若一个先验框没有与任何ground truth进行匹配，那么该先验框只能与背景匹配；（一个图片中ground truth是非常少的， 而先验框却很多，如果仅按第一个原则匹配，很多先验框会是负样本，正负样本极其不平衡，所以需要第二个原则）
>
> 2.对于剩余的未匹配先验框，若与某个ground truth的IoU大于某个阈值（一般是0.5），那么该先验框也与这个ground truth进行匹配；
>
> 尽管一个ground truth可以与多个先验框匹配，但是ground truth相对先验框还是太少了，所以负样本相对正样本会很多。为了保证正负样本尽量平衡，SSD采用了hard negative mining，就是对负样本进行抽样，抽样时按照置信度误差（预测背景的置信度越小，误差越大）进行降序排列，选取误差的较大的top-k作为训练的负样本，以保证正负样本比例接近1:3

- 损失函数

损失函数定义为位置误差（locatization loss， loc）与置信度误差（confidence loss, conf）的加权和：

$$L(x,c,l,g)=\frac{1}{N}(L_{conf}(x,c)+\alpha L_{loc}(x,l,g))$$

其中 N 是先验框的正样本数量，这里 $x_{ij}^p\in\{1,0\}$ 为一个指示参数, 当 $x_{ij}^p=1$ 时表示第 i 个先验框与第 j 个ground truth匹配, 并且ground truth的类别为 p。 c 为类别置信度预测值， l 为先验框的所对应边界框的位置预测值, 而 g 是ground truth的位置参数。权重系数 α 通过交叉验证设置为1

对于**位置误差,** 其采用**Smooth L1 loss**, 定义如下:

$$L_{loc}(x,l,g)=\sum_{i\in Pos}^{N}\sum_{m\in\{cx,cy,w,h\}}x_{ij}^k\,\mathrm{smooth}_{L1}\left(l_i^m-\hat{g}_j^m\right)$$

$$\hat{g}_j^{cx}=\left(g_j^{cx}-d_i^{cx}\right)/d_i^w\;\hat{g}_j^{cy}=\left(g_j^{cy}-d_i^{cy}\right)/d_i^h$$

$$\hat{g}_j^w=\log\left(\frac{g_j^w}{d_i^w}\right)\quad\hat{g}_j^h=\log\left(\frac{g_j^h}{d_i^h}\right)$$

$$\mathrm{smooth}_{L_1}(x)=\begin{cases}0.5x^2 & \text{if } |x|<1\\ |x|-0.5 & \text{otherwise}\end{cases}$$

由于 x_{ij}^p 的存在, 所以**位置误差仅针对正样本进行计算**，值得注意的是, 要先对ground truth的 g 进行编码得到 $\hat{g}$, 因为预测值 l 也是编码值,

对于**置信度误差**, 其采用softmax loss:

$$L_{\text{conf}}(x,c)=-\sum_{i\in\text{ Pos}}^{N}x_{ij}^p\log\left(\hat{c}_i^p\right)-\sum_{i\in\text{ Neg}}\log\left(\hat{c}_i^0\right)\quad\text{where}\quad\hat{c}_i^p=\frac{\exp\left(c_i^p\right)}{\sum_p\exp\left(c_i^p\right)}$$

预测过程：

- 对于每个预测框，首先根据类别置信度确定其类别与置信度值，并过滤掉属于背景的预测框；
- 根据置信度阈值（如0.5）过滤掉阈值较低的预测框，对于留下的预测框进行解码，根据先验框得到其真实的位置参数（解码后一般还需要做clip，防止预测框位置超出图片）；
- 解码之后，一般需要根据置信度进行降序排列，然后仅保留top-k（如400）个预测框；
- 进行NMS算法，过滤掉那些重叠度较大的预测框，最后剩余的预测框就是检测结果了；

Faster RCNN

使用RPN网络代替Fast RCNN使用的选择性搜索进行候选区域的提取，相当于Faster R-CNN=RPN + Fast RCNN，且RPN和Fast RCNN共享卷积层；

1.多尺度目标：通过RPN网络候选区域，并使用不同大小和长宽比的anchors来解决多尺度问题；
2.通过计算anchors与真实框的交并比IOU，并通过阈值建立正负样本；
3.样本不平衡：每批次随机采样256个anchors进行边框回归训练，并尽可能保证正负样本数相同（训练程序会在合适的anchors中随机选取128个postive anchors+128个negative anchors进行训练），避免负样本过多导致的梯度统治问题；

改进：

```
ReNeXt-SPP
FPN / BiFPN
DIoU NMS
ROI Align
```

Backbone

提取图像的feature maps，用于后续的RPN和classifier

feat_stride=16

RPN

用于生成region proposals：该层通过softmax判断anchors属于positive或者negative，再利用**bounding box regression**修正anchors获得精确的proposals

其实RPN最终就是在原图尺度上，设置了密密麻麻的候选Anchor，然后用cnn去判断哪些Anchor是里面有目标的positive anchor，哪些是没目标的negative anchor。所以，仅仅是个二分类而已！

Loss

$$t_x = (x - x_a)/w_a \quad t_y = (y - y_a)/h_a \tag{9}$$

$$t_w = \log(w/w_a) \quad t_h = \log(h/h_a) \tag{10}$$

```
分类Loss使用CE；
回归Loss使用Smooth L1 loss；
```

ROI Pooling

- 收集输入的feature maps和proposals，综合这些信息后提取proposal feature maps，送入后续全连接层判定目标类别
- RoI Pooling的过程就是将一个个大小不同的box矩形框，都映射成大小固定（w * h）的矩形框

解决传统CNN输入图像固定的问题；-> SPP发展而来

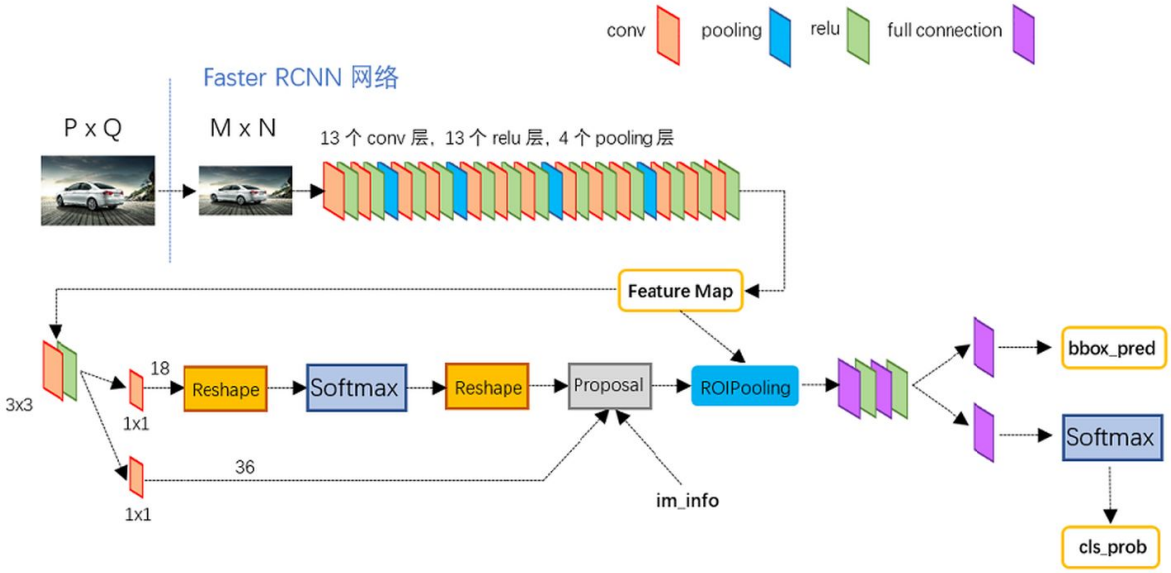
```
1.将每个proposal对应的feature map区域水平分为pooled_w x pooled_h网格；
2.对网格每一份进行max pooling处理；

- > 大小不同的proposal输出结果都是pooled_w x pooled_h固定大小,实现了固定长度输出；
```

缺点： 由于 RoIPooling 采用的是最近邻插值（即INTER_NEAREST），在resize时，对于缩放后坐标不能刚好为整数的情况，采用了粗暴的舍去小数，相当于选取离目标点最近的点，损失一定的空间精度；

映射: 把各个坐标除以“输入图片与feature map的大小的比值”

Classification

利用已经获得的proposal feature maps，通过full connect层与softmax计算每个proposal具体属于那个类别（如人，车，电视等），输出cls_prob概率向量；同时再次利用bounding box regression获得每个proposal的位置偏移量bbox_pred，用于回归更加精确的目标检测框



RPN网络上面一条通过softmax分类anchors获得positive和negative分类，下面一条用于计算对于anchors的bounding box regression偏移量，以获得精确的proposal；

Proposal层则负责综合positive anchors和对应bounding box regression偏移量获取proposals，同时剔除太小和超出边界的proposals，并送入送入后续RoI Pooling Layer（输出的proposals对应MxN输入图像尺度）

Anchor

```
- 长宽比：1：1， 1：2， 2：1；
- 面积尺寸：128 256 512
- 3种尺度anchor size，预设Anchor9个矩形，引入了检测中常用到的多尺度方法
- 全部anchors拿去训练太多了，训练程序会在合适的anchors中随机选取128个postive anchors + 128个negative anchors进行训练
- 目前anchor box尺寸的选择主要有三种方式：人为经验选取、k-means聚类、作为超参数进行学习
- 为什么使用不同尺寸和不同长宽比？ 为了得到更大的IOU
```

正标签：1.具有与GT边框的重叠最高交并比的Anchor 2.具有与GT边框的重叠超过0.7 IoU的Anchor；

负标签：具有与GT边框的重叠低于0.3 IoU的Anchor;

训练步骤：

- **训练RPN**：用ImageNet预训练的模型初始化，用于生成region proposal；
- **训练Fast R-CNN**：利用第一步的RPN生成的region proposals作为输入数据，训练Fast R-CNN一个单独的检测网络，这时候两个网络还**没有共享卷积层**；

- **调优RPN：**用第二步的fast-rcnn初始化RPN再次进行训练，但**固定共享的卷积层，并且只微调RPN独有的层**，现在两个网络共享卷积层了；

- **调优Fast R-CNN**：由第三步的RPN model初始化fast-RCNN网络，输入数据为第三步生成的proposals，**保持共享的卷积层固定，微调Fast R-CNN的fc层**，这样，**两个网络共享相同的卷积层，构成一个统一的网络**

Cascade R-CNN

通过分析Faster RCNN在目标候选区域的位置修正能力， 基于单个检测器的可优化性但优化的程度有限，通过 多次将预测区域作为候选区域进行修正，使得输出的预测区域与真实标签区域的IOU逐级递增；

主要优点：
1、准确性：碾压各种单双阶段目标检测算法，采用RoIAlign取代RoIPooling
2、多尺度：通过FPN网络集成多尺度特征图，利用归一化尺度偏差方法缓解不同尺度对Loss的影响程度
3、实时性：去除了Fater RCNN的全连接层，取而代之采用FCN网络，相比Fater RCNN，具有更少的模型参数和计算时间

Transformer

- CNN是受限的Transformer：CNN只考虑感受野范围之内的信息，Transformer做self-attention时候考虑了整张图像与其相关的pixel信息
- 图像RGB 一个pixel 3通道可以看作一个vector -> query；
- Layer Norm（在Transformer里面比Batch Normalization好，去除LN在YOLov5中效果更好）：
- BERT是双向Transformer的Encoder，GPT是Transformer的Decoder；

目前Transformer应用到图像领域主要有两大挑战：

- **视觉实体变化大**，在不同场景下视觉Transformer性能未必很好
- **图像分辨率高**，像素点多，Transformer基于全局自注意力的计算导致**计算量较大**

DETR

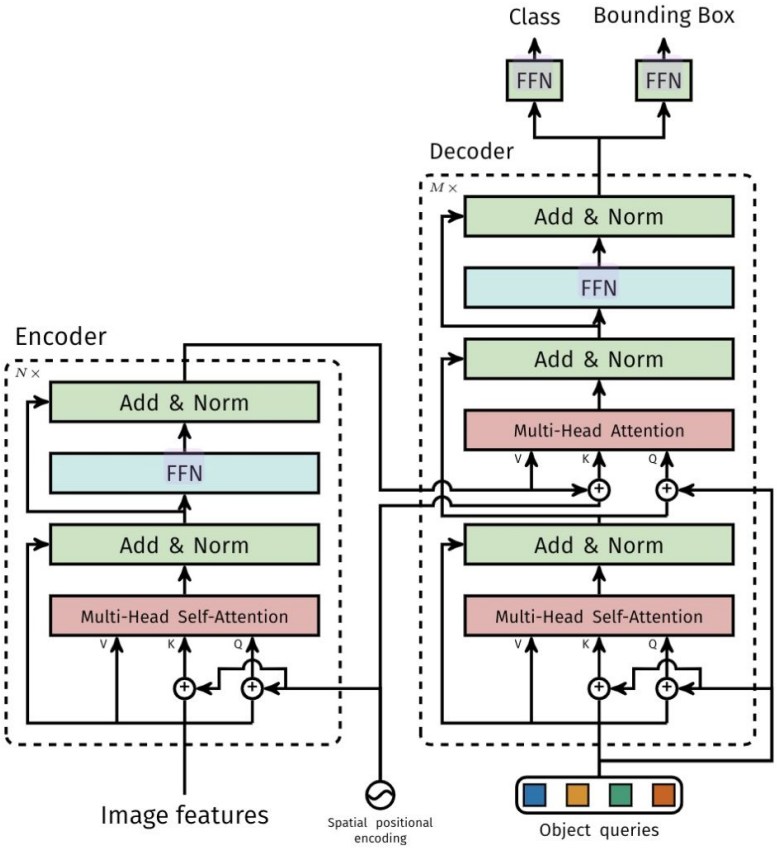


Fig. 10: Architecture of DETR's transformer. Please, see Section ? for details.

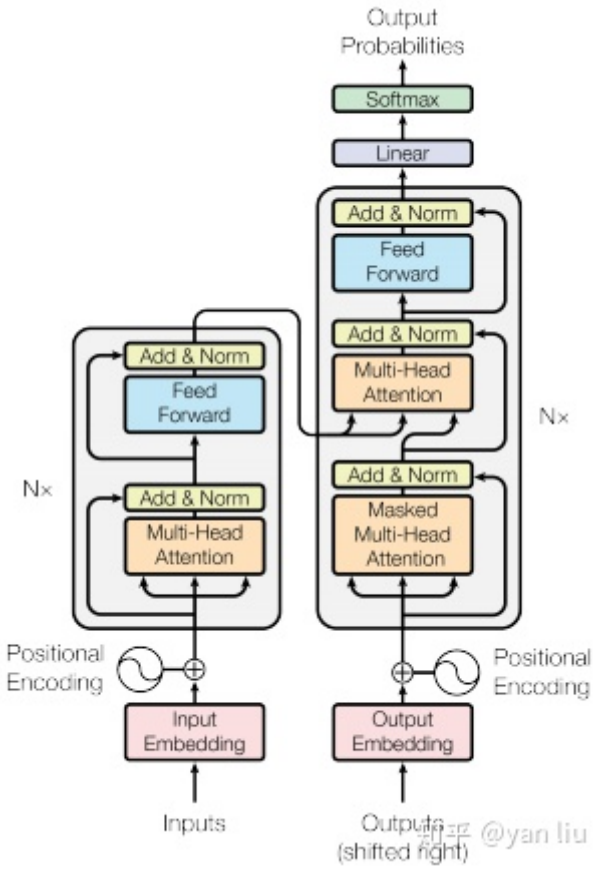
核心点：

- Object Detection上DETR准确率和运行时间上和Faster RCNN相当**（DETR不是一个完全由Transformer处理的架构，还是需要依赖于CNN作为backbone提取特征）**
- Transformer的Encoder和Decoder结构
- Bipartite Matching Loss

Transformer：主要包括三个部分：Encoder, Decoder和FFN

1.Backbone：

在DETR中, 首先用CNN backbone处理 $3 \times H_0 \times W_0$ 维的图像，得到 $C \times H \times W$ 维的feature map (一般 $C = 2048, H = \frac{H_0}{32}, W = \frac{W_0}{32}$)，然后将backbone输出的feature map和position encoding相加, 输入Transformer Encoder中处理, 得到用于输入到Transformer Decoder的image embedding；



原始Transformer

FFN (Feed-forward network)包含两个FC层, 第1个FC层将特征从维度 D 变换成 $4D$, 后一个FC层将特征从维度 $4D$ 恢复成 D, 中间的非线性激活函数采用GeLU, 其实这就是一个MLP

2.Encoder：

Transformer Encoder的结构和Transformer原始论文中的结构相同，将CNN backbone输出的feature map转化为能够被Transformer Encoder处理的序列化数据的过程。主要有以下几个步骤：

1. **维度压缩：**将CNN backbone输出的 $C \times H \times W$ 维的feature map先用 1×1 convolution处理, 将channels数量从 C 压缩到 d, 即得到 $d \times H \times W$ 维的新feature map;
2. **转化为序列化数据：**将空间的维度 (高和宽) 压缩为一个维度, 即把上一步得到的 $d \times H \times W$ 维的feature map通过reshape成 $d \times HW$ 维的feature map;
3. **加上positoin encoding:** 由于transformer模型是顺序无关的, 而 $d \times HW$ 维feature map 中 HW 维度显然与原图的位置有关, 所以需要加上position encoding反映位置信息；

3.Decoder：

Transformer Decoder结构基本上和原始的Transformer相同，可以看到transformer主要有两个输入：

1. **image embedding** (由Transformer Encoder输出) 与 position encoding 之和；
2. **object queries**；

Object queries有 N 个 (其中 N 是一个事先设定的、比远远大于image中object个数的一个 整数) , 输入Transformer Decoder后分别得到 N 个decoder output embedding, 经过FFN处理后就得到了 N 个预测的boxes和这些boxes的类别；
具体实现上, object queries是 N 个learnable embedding, 训练刚开始时可以随机初始化，在训练过程

中，因为需要生成不同的boxes, object queries会被迫使变得不同来反映位置信息, 所 以也可以称为leant positional encoding (注意和encoder中讲的position encoding区分, 不是 一个东西)；

此外，和原始的Transformer不同的是，DETR的Transformer Decoder是一次性处理全部的object queries，即一次性输出全部的predictions；而不像原始的Transformer是auto-regressive的，从左到右一个词一个词地输出

4.Prediction Head - FFN [Feed-forward network]：

有两种FFN：一种预测bounding box的中心位置、高和宽，第二种预测class标签；

5.Loss

上面我们介绍了如何用Transformer生成 N 个prediction boxes (其中 N 是一个事先设定好的 远远大于image objects个数的整数），这里我们将介绍如何得到这 N 个prediction boxes和 image objects的**最优二分图匹配**,，并通过计算配对的loss来评价prediction boxes生成的好坏；

5.1 得到prediction boxes和image objects的最优二分图匹配

假设对于一张图来说, ground truth boxes的个数 (即图中object的个数) 为 m, 由于 N 是一个事先设定好的远远大于image objects个数的整数, 所以 $N >> m$, 即生成的prediction boxes的数量会远远大于ground truth boxes 数量。这样怎么做匹配?
为了解决这个问题, 作者人为构造了一个新的物体类别 ϕ （表示没有物体）并加入image objects中, 上面所说到的多出来的 $N-m$ 个prediction embedding就会和 ϕ 类别配对。这 样就可以将prediction boxes和image objects的配对看作两个等容量的集合的二分图匹配了；

只要定义好每对prediction box和image object匹配的cost, 我们就能用匈牙利算法快速地找到 使总cost最小的二分图匹配方案；
每对prediction box和image object匹配的cost L_{match} 定义如下:

$$-1_{\{c_i \neq \varnothing\}} \hat{p}_{\sigma(i)}\left(c_i\right)+1_{\{c_i \neq \varnothing\}} \mathcal{L}_{\text{box}}\left(b_i, \hat{b}_{\sigma(i)}\right)$$

c_i 为第 i 个image object的class标签, $\sigma(i)$ 为与第 i 个object配对的prediction box的 index；
$1_{\{c_i \neq \phi\}}$ 是一个函数，当 $c_i \neq \phi$ 时为1，否则为0；
$\hat{p}_{\sigma(i)}\left(c_i\right)$ 表示Transformer预测的第 $\sigma(i)$ 个prediction box类别为 c_i 的概率；
$b_i, \hat{b}_{\sigma(i)}$ 分别为第 i 个image object的box和第 $\sigma(i)$ 个prediction box的位置；
$L_{box}\left(b_i, \hat{b}_{\sigma(i)}\right)$ 计算的是ground truth box和prediction box之间的差距；

L_{box} 的计算公式如下:

$$\lambda_{\text{iou}} \mathcal{L}_{\text{iou}}\left(b_i, \hat{b}_{\sigma(i)}\right)+\lambda_{\text{L1}}\left\|b_i-\hat{b}_{\sigma(i)}\right\|_1$$

其中 $\|b_i-\hat{b}_{\sigma(i)}\|$ 为两个box中心坐标的L1距离。如果只用L1距离, 当box大小不同时, 即使 L1距离相同, 差距也会不一样。例如, 两个3x3的box相距3, 和两个30x30的box相距3的差距是 完全不同的。为了解决这个问题, 作者还加入了与box大小无关的 L_{iou}, 通过IOU计算loss；

这样，我们就完全定义好了每对prediction box和 image object 配对时的cost，再利用匈牙利算法即可得到二分图最优匹配；

当配对的image object为 ϕ 时, 我们人为规定配对的cost $L_{match}=0$; 当配对的image object为真实的物体 (即不为 ϕ) 时, 如果预测的prediction box类别和 image object类别相同的概率越大 (越小)，或者两者的box差距越小 (越大)， 配对的cost L_{match} 越小 (越大) ;

5.2 根据最优二分图匹配计算set prediction loss

上面我们得到了prediction boxes和image objects之间的最优匹配。这里我们基于这个最优匹配，来计算set prediction loss，即评价Transformer生成这些prediction boxes的效果好坏;

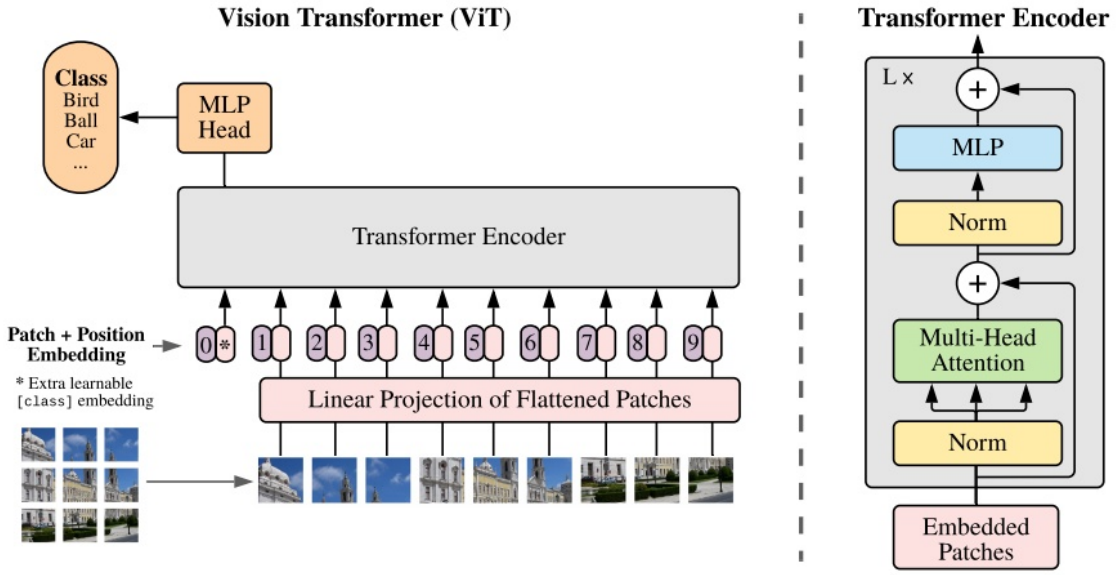
Set prediction loss $L_{\text{Hungarian}}$ 计算公式如下:

$$\mathcal{L}_{\text{Hungarian}}(y,\hat{y})=\sum_{i=1}^{N}\left[-\log\hat{p}_{\hat{\sigma}(i)}(c_i)+1_{\{c_i\neq\varnothing\}}\mathcal{L}_{\text{box}}\left(b_i,\hat{b}_{\hat{\sigma}}(i)\right)\right]$$

其中 $\hat{\sigma}$ 为最佳匹配, 将第 i 个image object 匹配到第 $\hat{\sigma}(i)$ 个prediction box，这个式子各个 部分的含义与 L_{match} 计算公式完全一致, 唯一需要注意的是, 这里用的是 classification probability $\hat{p}_{\hat{\sigma}(i)}(c_i)$ 的对数形式, 而 L_{match} 中直接用的线性形式; 且这里考虑了被匹配到 ϕ object的 $\hat{p}_{\hat{\sigma}(i)}(c_i)$, 而 L_{match} 则直接定义为0

```
import torch
from torch import nn
from torchvision.models import resnet50

class DETR(nn.Module):

    def __init__(self, num_classes, hidden_dim, nheads,
                 num_encoder_layers, num_decoder_layers):
        super().__init__()
        # We take only convolutional layers from ResNet-50 model
        self.backbone = nn.Sequential(*list(resnet50(pretrained=True).children())[:-2])
        self.conv = nn.Conv2d(2048, hidden_dim, 1)
        self.transformer = nn.Transformer(hidden_dim, nheads,
                                          num_encoder_layers, num_decoder_layers)
        self.linear_class = nn.Linear(hidden_dim, num_classes + 1)
        self.linear_bbox = nn.Linear(hidden_dim, 4)
        self.query_pos = nn.Parameter(torch.rand(100, hidden_dim))
        self.row_embed = nn.Parameter(torch.rand(50, hidden_dim // 2))
        self.col_embed = nn.Parameter(torch.rand(50, hidden_dim // 2))

    def forward(self, inputs):
        x = self.backbone(inputs)
        h = self.conv(x)
        H, W = h.shape[-2:]
        pos = torch.cat([
            self.col_embed[:W].unsqueeze(0).repeat(H, 1, 1),
            self.row_embed[:H].unsqueeze(1).repeat(1, W, 1),
        ], dim=-1).flatten(0, 1).unsqueeze(1)
        h = self.transformer(pos + h.flatten(2).permute(2, 0, 1),
                             self.query_pos.unsqueeze(1))
        return self.linear_class(h), self.linear_bbox(h).sigmoid()

detr = DETR(num_classes=91, hidden_dim=256, nheads=8, num_encoder_layers=6, num_decoder_layers=6)
detr.eval()
inputs = torch.randn(1, 3, 800, 1200)
logits, bboxes = detr(inputs)
```

ViT



- 卷积具有天然的先天优势）：平移等价性（translation equivariance）和局部性（locality）；
- Transformer的核心self-attention的优势不像卷积那样有固定且有限的感受野，self-attention操作可以获得long-range信息（相比之下CNN要通过不断堆积Conv layers来获取更大的感受野）；

ViT的思路：直接把图像分成固定大小的patchs，然后通过线性变换得到patch embedding，这就类比NLP的words和word embedding，由于Transformer的输入就是a sequence of token embeddings，所以将图像的patch embeddings送入transformer后就能够进行特征提取从而分类了；

Patch Embedding

对于ViT来说, 首先要将原始的2-D图像转换成一系列1-D的patch embeddings, 这就好似NLP中 的word embedding。输入的2-D图像记为 $\mathbf{x} \in \mathbb{R}^{H \times W \times C}$ ，其中 H 和 W 分别是图像的高和 宽，而 C 为通道数对于RGB图像就是3。如果要将图像分成大小为 $P \times P$ 的patchs, 可以通过 reshape操作得到a sequence of patchs: $\mathbf{x}_p \in \mathbb{R}^{N \times (P^2 \cdot C)}$, 图像共切分为 $N = HW/P^2$ 个patchs, 这也就是sequence的长度了，注意这里直接将patch拉平为1-D, 其特征大小为 $P^2 \cdot C$ 。然后通过一个简单的线性变换将patchs映射到 D 大小的维度，这就是patch embeddings: $\mathbf{x}_\mathbf{p}' \in \mathbb{R}^{N \times D}$, 在实现上这等同于对 $\mathbf{x}_p$ 进行一个 $P \times P$ 且stride为 P 的卷积 操作 (虽然等同，但是ViT其实是不包含任何卷积操作的)；

Position Embedding

transformer和CNN不同，需要position embedding来编码tokens的位置信息，这主要是因为self-attention是permutation-invariant，即打乱sequence里的tokens的顺序并不会改变结果。如果不给模型提供patch的位置信息，那么模型就需要通过patchs的语义来学习拼图，这就额外增加了学习成本。ViT论文中对比了几种不同的position embedding方案(如下），最后发现如果不提供positional embedding效果会差，但其它各种类型的positional embedding效果都接近，这主要是因为ViT的输入是相对较大的patchs而不是pixels，所以学习位置信息相对容易很多；

ViT中默认采用学习（训练的）的1-D positional embedding，在输入transformer的encoder之前直接将patch embeddings和positional embedding相加

```
# 这里多1是为了后面要说的class token，embed_dim即patch embed_dim
self.pos_embed = nn.Parameter(torch.zeros(1, num_patches + 1, embed_dim))

# patch emded + pos_embed
x = x + self.pos_embed
```

对学习到的positional embedding进行了可视化，发现相近的patchs的positional embedding比较相似，而且同行或同列的positional embedding也相近：

这里额外要注意的一点，**如果改变图像的输入大小，ViT不会改变patchs的大小，那么patchs的数 量 N 会发生变化，那么之前学习的pos_embed就维度对不上了，ViT采用的方案是通过插值来解决这个问题**；

Class Token

除了patch tokens，ViT借鉴BERT还增加了一个特殊的class token：ransformer的encoder输入是a sequence patch embeddings，输出也是同样长度的a sequence patch features，但图像分类最后需要获取image feature，简单的策略是采用pooling，比如求patch features的平均来获取image feature，但是ViT并没有采用类似的pooling策略，而是直接增加一个特殊的class token，其最后输出的特征加一个linear classifier就可以实现对图像的分类（ViT的pre-training时是接一个MLP head），所以输入ViT的sequence长度是`N+1`。**class token对应的embedding在训练时随机初始化，然后通过训练得到;**

Transformer Encoder

和原始Transformer结构相同；

ViT

对于ViT模型来说，就类似CNN那样，**不断堆积transformer encoder blocks，最后提取class token对应的特征用于图像分类，**论文中也给出了模型的公式表达，其中（1）就是提取图像的patch embeddings，然后和class token对应的embedding拼接在一起并加上positional embedding；（2）是MSA，而（3）是MLP，（2）和（3）共同组成了一个transformer encoder block，共有L层；（4）是对class token对应的输出做layer norm，然后就可以用来图像分类；

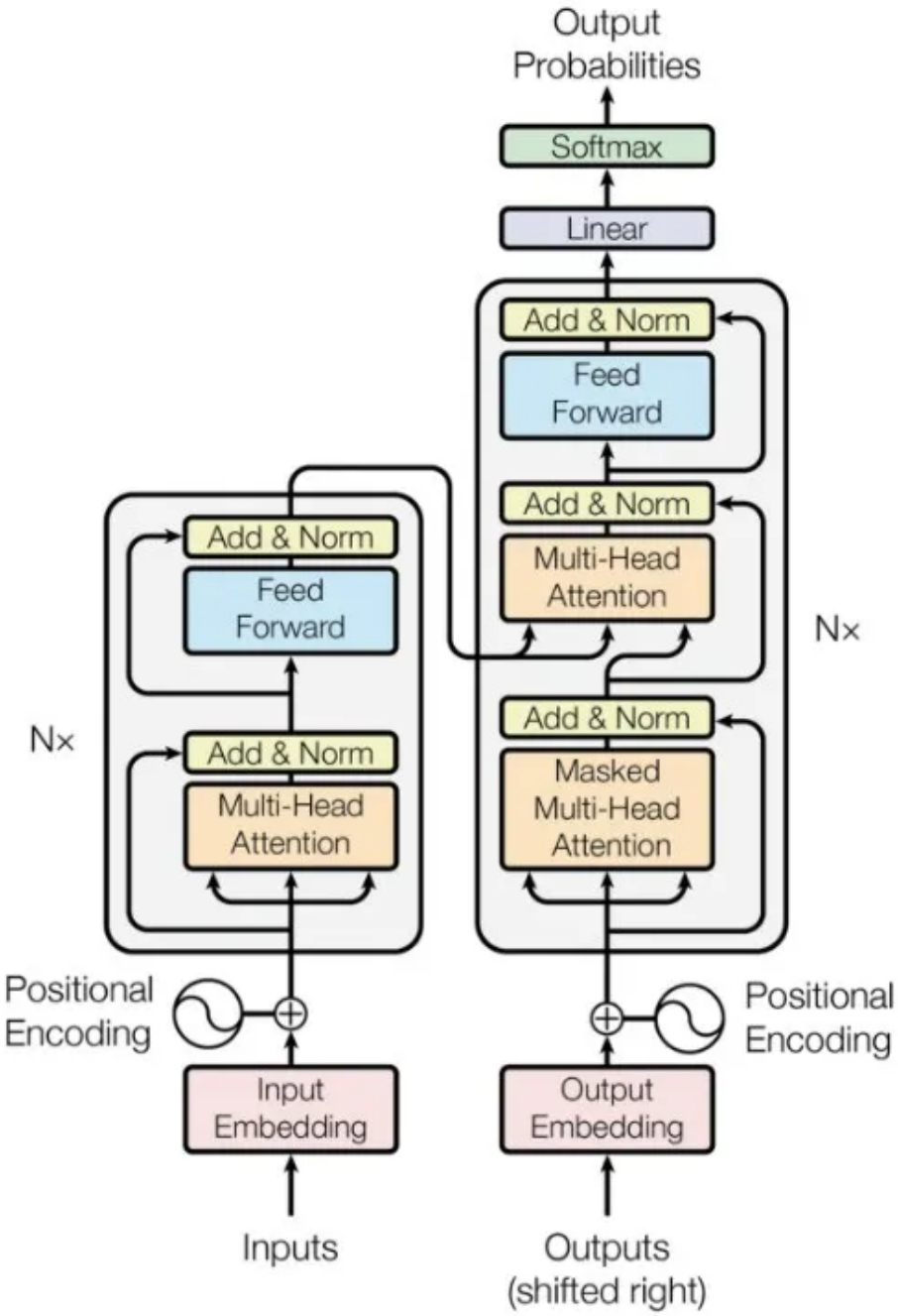
$$
\begin{aligned}
\mathbf{z}_0 &= \left[\mathbf{x}_{\text{class}}\,;\mathbf{x}_p^1\mathbf{E};\mathbf{x}_p^2\mathbf{E};\cdots;\mathbf{x}_p^N\mathbf{E}\right]+\mathbf{E}_{\text{pos}}\,, && \mathbf{E}\in\mathbb{R}^{(P^2\cdot C)\times D},\mathbf{E}_{\text{pos}}\in\mathbb{R}^{(N+1)\times D}\\
\mathbf{z}'_\ell &= \text{MSA}(\text{LN}(\mathbf{z}_{\ell-1}))+\mathbf{z}_{\ell-1}, && \ell=1\ldots L\\
\mathbf{z}_\ell &= \text{MLP}\big(\text{LN}\big(\mathbf{z}'_\ell\big)\big)+\mathbf{z}'_\ell, && =1\ldots L\\
\mathbf{y} &= \text{LN}\big(\mathbf{z}_L^0\big)
\end{aligned}
$$

除了完全无卷积的ViT模型外, 论文中也给出了Hybrid Architecture, 简单来说就是先用CNN对 图像提取特征, 从CNN提取的特征图中提取patch embeddings, CNN已经将图像降采样了, 所 以patch size可以为 1×1 。
ViT模型的超参数主要包括以下，这些超参数直接影响模型参数以及计算量:

1. Layers: block的数量;
2. Hidden size D: 隐含层特征, D在各个block是一直不变的;
3. MLP size：一般设置为4D大小;
4. Heads: MSA中的heads数量;
5. Patch size：模型输入的patch size, ViT中共有两个设置：14x14和16x16, 这个只影响计算 量;

Transformer

Transformer结构中，左边叫做编码端(Encoder)，右边叫做解码端(Decoder)。大家不要小看这两个部分，其中左边的编码端最后演化成了最后鼎鼎大名的Bert，右边的解码端在最近变成了无人不知的GPT模型；



- **Self-attention**，可以让NLP模型做到像CV模型一样，**并行输入**：对于Self-attention结果而言，它可以一次性的将所有的字都当做输入。但是NLP的输入是有特点的，其特点是输入的文本要按照一定的顺序才可以。因为，文本的顺序是带有一部分语义关系的；
- **输入的内容具有一定的位置信息：Position机制**；

Transformer位置编码

- **函数型**：通过输入token位置信息，得到相应的位置编码；✅
  - 相对位置编码的特点，关注一个token与另一个token距离的相对位置(距离差几个token)，使用相对位置的方法，我们可以清晰的知道单词之间的距离远近的关系；-> **Transformer编码是线性关系，$PE_{(}pos)$和 $PE_{(}pos+k)$相乘后的结果为一个余弦的加和，这样的关系可以得到，如果两个token距离越远则乘积的结果越小；**
- **表格型**：建立一个长度为L的词表，按词表的长度来分配位置id；
  - **使用[0,1]范围分配：**
    - 将0-1这个范围的，将第一个token分配0，最后一个token分配去1，其余的token按照文章的长度平均分配；
    - 如果句子长度不同，那么位置编码是不一样，所以无法表示句子之间有什么相似性；
  - **1-n正整数范围分配：**
    - 按照输入的顺序，一次分配给token所在的索引位置；
    - 往往句子越长，后面的值越大，数字越大说明这个位置占的权重也越大，这样的方式无法凸显每个位置的真实的权重；

Transformer的Position

Transformer的位置信息是函数型的。在GPT-3论文中给出的公式如下：

$$\overrightarrow{p_t}^{(i)} = f(t)^{(i)} := \begin{cases} \sin(\omega_k . t), & \text{if } i = 2k \\ \cos(\omega_k . t), & \text{if } i = 2k+1 \end{cases}$$

上个公式给出的每一个Token的位置信息编码不是一个数字，而是一个不同频率w_k分割出来，和文本一样维度的向量。向量如下：

$$\overrightarrow{p_t} = \begin{bmatrix} \sin(\omega_1 \cdot t) \\ \cos(\omega_1 \cdot t) \\ \sin(\omega_2 \cdot t) \\ \cos(\omega_2 \cdot t) \\ \vdots \\ \sin(\omega_{d/2} \cdot t) \\ \cos(\omega_{d/2} \cdot t) \end{bmatrix}$$

关于w_i（频率）：

$$w_i = \frac{1}{10000^{2i/d_{model}}}$$

关于t:：这里的 就是每个token的位置，比如说是位置1，位置2，以及位置n；

得到位置向量P之后，将和模型的embedding向量相加，得到进入Transformer模型的最终表示:

$$\psi'(w_t) = \psi(w_t) + \overrightarrow{p_t}$$

Transformer的Position编码问题：只能的到相对关系，无法得到方向关系：对于两个token谁在谁的前面，或者谁在谁的后面是无法判断的；