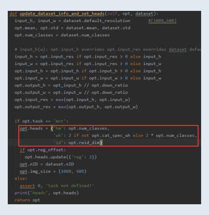
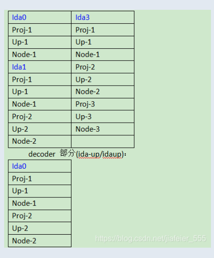
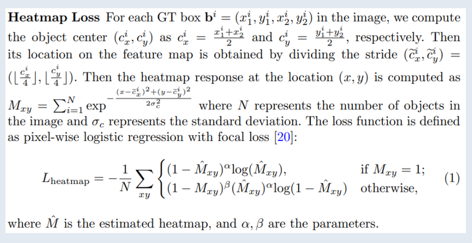
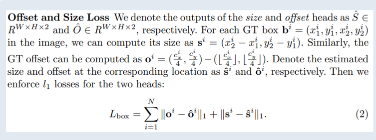
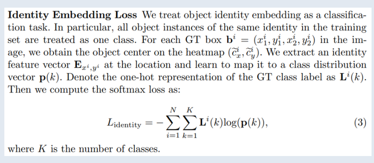
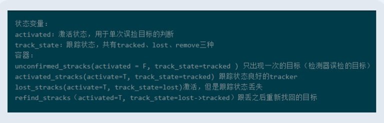
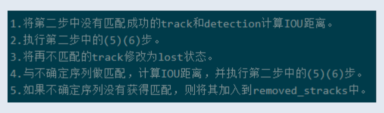
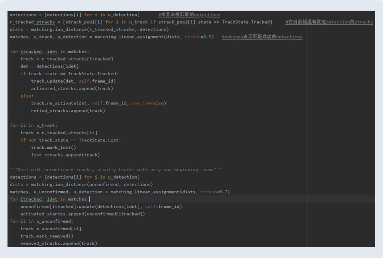
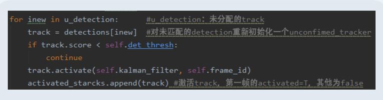
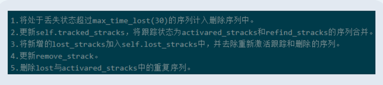
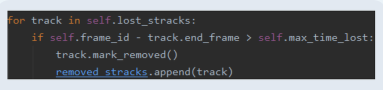
# FairMOT

- **配置文件opt.py**
  - 设置数据集，head、loss和优化及相关路径等参数的设置
  - def update_dataset_info_and_set_heads(self, opt, dataset)
  - 
    opt.heads即设置head，分别是hm、wh和id，对应的维度分别是1、2、512
- **数据读取jde.py**
  - 数据的读取及预处理(绘制hm) — H×W×1
  - def letterbox(img, height=608, width=1088, color=(127.5, 127.5, 127.5)) — img=[608,1088]
  - class JointDataset(LoadImagesAndLabels)
  - gaussian_radius()
- **Network：pose_dla_dcn**
  - encode&decode :提取高分辨率特征图
    - 
      encode：实现不同block、不同深度之间的特征融合
      — 
      decode：反卷积上采样，输出原图1/4图像
  - 网络中添加了三个平行回归head，分别用来预测hm、中心偏移量及wh，具体实现方式为对backbone的输出特征图进行3×3卷积，再进行1×1卷积生成。
    - 
      head_conv=256，fc（对初始化权重及hm初始bias进行设置）
  - Identity Embedding Branch — backbone特征图上应用具有128个核的卷积层提取每个位置的身份嵌入，得到feature map，进一步得到每个（x，y）上的Re-ID特征向量 — 128×W×H
- **Loss：mot.py**
  - 各head结构对应的loss
  - hm loss:MSEloss — 
  - offset and size loss:L1loss — 
  - ID loss:CrossEntropyLoss — 
  - det_loss — 
    line:74
  - tol_loss — 
    line:76
- **Track：multitrack.py**
  - class JDETracker(object):
    def update(self, im_blob, img0)
    - 变量 — 
    - step1 — 
      获得当前帧(第一帧)的检测框和id特征
      - output = self.model(im_blob)[-1]
      - id_feature = id_feature[remain_inds]
    - step2 — 
      首次将当前帧(第二帧)与前一帧进行外观+距离匹配 — 
    - step3 — 
      第二次将当前帧与前一帧进行匹配，通过IOU — 
    - step4 — 
      初始化新序列 — 
    - step5 — 
      更新状态 — 
- **Detection：centernet**
  - decode.py(将hm解码成bbox)
    - _nms()

```
def _nms(heat, kernel=3):
  pad = (kernel - 1) // 2
  hmax = nn.functional.max_pool2d(
      heat, (kernel, kernel), stride=1, padding=pad)
  keep = (hmax == heat).float()
  return heat * keep
寻找8邻近极大值带点，keep为h极大值点的位置，返回heat*keep，筛选出极大值点，为原值，其余为0。
```

    - _topk()

```
def _topk(scores, K=40):            #取最大值的前K个
  batch, cat, height, width = scores.size()
  topk_scores, topk_inds = torch.topk(scores.view(batch, cat, -1), K)
  #topk_scores, topk_inds: 每个batch中前K个最大scores, id   Dim:[batch,cat,K]

  topk_inds = topk_inds % (height * width)    ## xs、ys是inds转化成在heat_map上面的行、列
  topk_ys   = (topk_inds / width).int().float()
  topk_xs   = (topk_inds % width).int().float()

  topk_score, topk_ind = torch.topk(topk_scores.view(batch, -1), K)   #Dim:[batch,K]
  topk_clses = (topk_ind / K).int()
  topk_inds = _gather_feat(
      topk_inds.view(batch, -1, 1), topk_ind).view(batch, K)
  topk_ys = _gather_feat(topk_ys.view(batch, -1, 1), topk_ind).view(batch, K)
  topk_xs = _gather_feat(topk_xs.view(batch, -1, 1), topk_ind).view(batch, K)

  return topk_score, topk_inds, topk_clses, topk_ys, topk_xs      #找所有batch的前K个heatmap值，Dim[batch*K]
  调用utils中的_gather_feat()函数
  输入feat(topk_inds):batch*(cat*K)*1,ind(topk_ind):batch*K,
  对ind扩展为batch*K*1使用gather()将ind对应值取出，返回feat: batch*K*1
```

    - _tranpose_and_gather_feat()

```
def _tranpose_and_gather_feat(feat, ind):      #feat: [batch*C*W*H],ind:[batch *K]
  feat = feat.permute(0, 2, 3, 1).contiguous()    #C放到最后并将内存变为连续，用于view
  feat = feat.view(feat.size(0), -1, feat.size(3))    #feat:[batch*(W*H)*C ]
  feat = _gather_feat(feat, ind)
  return feat    #[batch*K*C]
```

    - mot_decode() — 获得每个筛选出的目标[x,y,w,h],用于跟踪