deepsort 算法的原理与代码解析

作者：vcjmhg
原文链接：https://ld246.com/article/1594784676258
来源网站：链滴

概述

前边我们讲了sort算法的原理，并且指出了它的不足--IDsw过大，为了解决该问题，17年时候sort算的团队又提出了DeepSort算法。Deepsort在原来Sort算法的基础上，改进了以下内容：

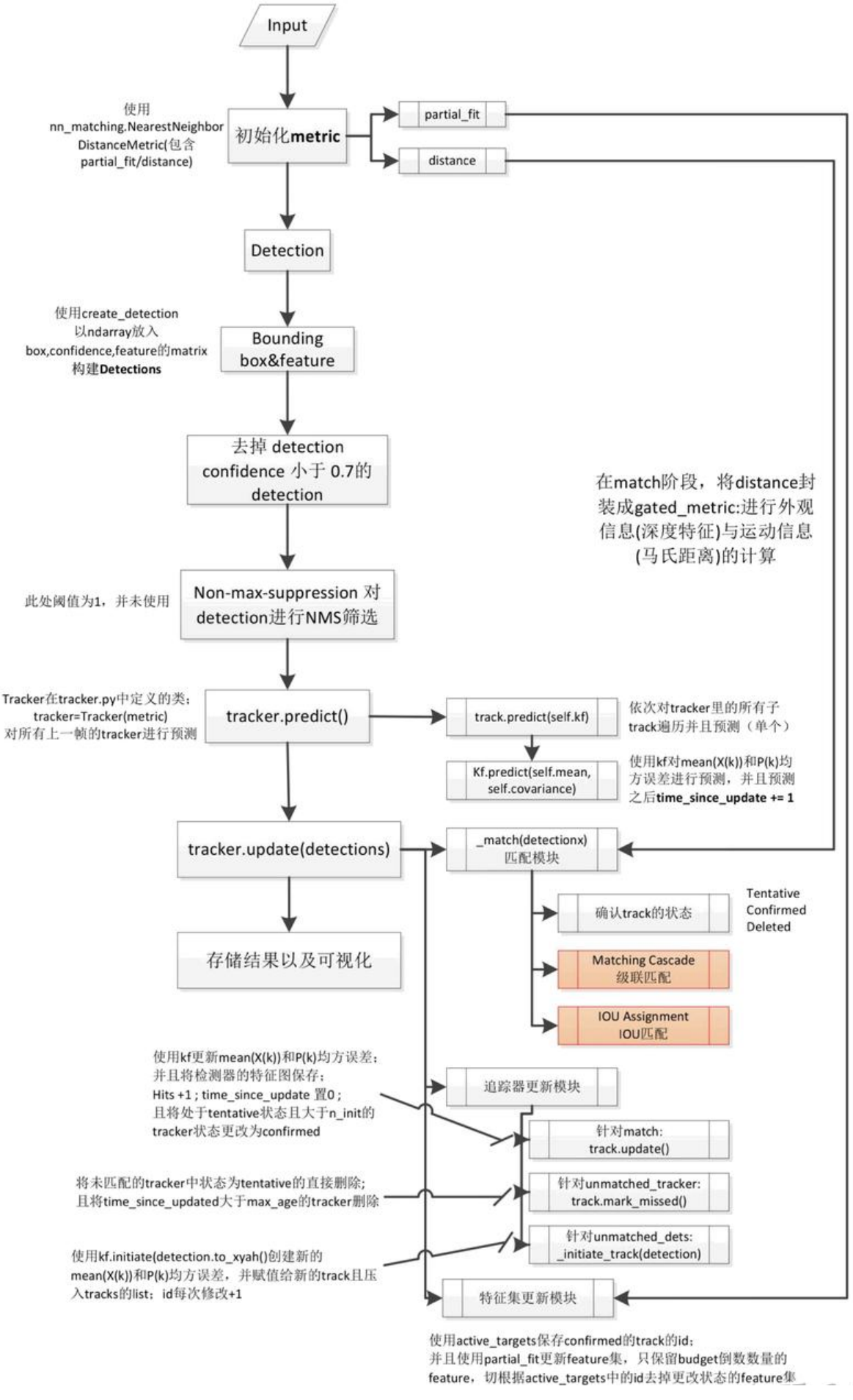
1. **使用级联匹配算法**：针对每一个检测器都会分配一个跟踪器，每个跟踪器会设定一个time_since_update参数。
2. 添加马氏距离与余弦距离：实际上是针对运动信息与外观信息的计算。
3. 添加深度学习特征：这一部分也就是ReID的模块，也是deepsort的亮点之一。

代码流程

由于deepsort的流程和算法原理几乎和sort一样，只是说增加了上边三个特色，因此我们直接从代码始讲起：

整体流程图

算法的整体流程图如下所示：



原文链接：deepsort 算法的原理与代码解析

源码流程

```
if __name__ == "__main__":
    args = parse_args()
    run(
        args.sequence_dir, args.detection_file, args.output_file,
        args.min_confidence, args.nms_max_overlap, args.min_detection_height,
        args.max_cosine_distance, args.nn_budget, args.display)
```

首先我们从主函数部分开始说起，主函数部分整体逻辑是比较简单的，首先是将命令行参数进行解析解析的内容包括，MOTChanlleng序列文件所在路径、需要检测文件所在的目录等一系列参数。解析后传递给run方法，开始运行。

```
# 收集流信息，包括图片名称，检测结果以及置信度等
seq_info = gather_sequence_info(sequence_dir, detection_file)
# metic实例化nn_matching的NearestNeighborDistanceMetric类，输入的初始距离度量函数是cosine,此时可以传入euclidean
metric = nn_matching.NearestNeighborDistanceMetric(
    "cosine", max_cosine_distance, nn_budget)
# 将度量方式对象传入到追踪器对象中，创建一个追踪器对象
tracker = Tracker(metric)
results = []
```

进入run函数之后，首先会收集流信息，包括图片名称，检测结果以及置信度等，后续会将这些流信传入到检测框生成函数中，生成检测框列表。然后会初始化metric对象，metric对象简单来说就是度方式，在这个地方我们可以选择两种相似度的度量方式，第一种叫做余弦相似度度量，另一种叫做欧相似度度量。通过metric对象我们来初始化追踪器。

```
# Run tracker.
# 根据display选择不同的可视化对象
if display:
    visualizer = visualization.Visualization(seq_info, update_ms=5)
else:
    visualizer = visualization.NoVisualization(seq_info)
# 调用对应的run方法开始进行追踪。
visualizer.run(frame_callback)
```

接着根据display参数开始生成对应的visuializer，如果选择将检测结果进行可视化展示，那么便会生Visualization对象，我从这个类中可以看到，它主要是调用opencv image viewer来讲追踪的结果进展示。如果display是false则会生成一个NoVisualization对象，它一个虚拟可视化对象，它以给定的序循环遍历所有帧以更新跟踪器，而无需执行任何可视化。两者主要区别其实就是是否调用opencv图片展示出来。其实前边我们所做的一系列工作可以说都是准备的工作，实际上核心部分就是在执行个run方法之后。此处我们可以看到，在run方法中传入了一个frame_callback函数，这个frame_callback函数可以说是整个算法的核心部分，每一帧的图片都会执行该函数。

为什么这样说那?

首先进入run函数之后我们会发现，无论当时选择是可视化操作还是非可视化操作，它的run函数最终要调用frame_callback函数的，比如说Visualization中的run方法，它首先判断帧序号是否大于最大，如果大于最大帧，直接返回。否则回调执行frame_callback函数；同样的如果是NoVisualization象，它的run方法也是类似的，也是调用frame_callback函数，然后帧序号加1。最后上边追踪结果处完之后，然后将追踪的结果保存到对应的目录下边。

```
    detections = create_detections(
        seq_info["detections"], frame_idx, min_detection_height)
    # 筛选检测框
    detections = [d for d in detections if d.confidence >= min_confidence]
```

上边我们说了frame_callback函数实际上是整个函数的核心内容。进入frame_callback函数，我们可看到，它第一步是根据之前的参数，生成检测框列表，然后将置信度小于最小置信度阈值的检测框剔掉。

```
# Run non-maxima suppression.
boxes = np.array([d.tlwh for d in detections])
scores = np.array([d.confidence for d in detections])
# 使用非极大抑制,找出局部范围内得分最大的框
indices = preprocessing.non_max_suppression(
    boxes, nms_max_overlap, scores)
# 更新检测框列表
detections = [detections[i] for i in indices]
```

然后执行非极大值抑制。

什么叫做非极大值抑制那?

简单来说就是就是一个寻找局部最大值的过程，我们以下边检测框为例，我们可以看到在检测人脸的程中可能会产生多个检测框，通过非极大值抑制，可以在局部范围内选择出那个得分最高的检测框，除掉其他得分低的检测框。

```
# Update tracker.
# tracker给出一个预测结果，然后将detection传入，进行卡尔曼滤波操作
tracker.predict()
tracker.update(detections)
```

接着调用predict函数执行预测操作，进入predict函数，我们可以看到，它其实主要就是对轨迹列表所有的轨迹使用卡尔曼滤波算法进行状态的预测。接着调用update函数执行更新操作。在update函中，主要如下几件事：

1. 根据之前预测的结果，进行匹配操作。在该开始匹配的时候都处于不确定态，然后若干次匹配之后如果匹配成功的次数大于n_init的话，轨迹便会从初始态转换成确定态。如果一直没有匹配到检测框会直接进入删除态。由于追踪的目标体可能会消失，因此就算进入到了确定态，如果在后续的匹配中次没有匹配到，大于max_age的时候轨迹便会从确定态转换成删除态。一旦轨迹进入到删除态，则证这个轨迹失效，后续便会被删除。

2. 针对不同状态进行不同的操作

a) 对未匹配的tracker,调用mark_missed标记，后续会删除对应的轨迹

b) 针对未匹配的detection（检测框）。没有匹配到目标，因此检测框失配，进行初始化操作

c) 更新轨迹列表，得到最新的tracks

d) 更新处于确定态的trac_id

e) 最后对特征集更新。

执行到此处当前帧的处理就完成了，直接帧序号加1，继续进行下一帧的操作。

结论

最后我们将代码整体原理整理成树图如下：

了便于大家阅读源码，我将源码的具体逻辑也整理成了一个树图：

deep_sort_app

1. parse_args:对命令行传入的参数进行解析，传递参数执行run方法

2.执行gather_sequence_info函数

3. 初始化metric

通过度量方式初始化追踪器

4.根据display生成不同的visualization

dispalay为true：生成Visuallization对象

如果display为false：生成NoVisualization对象

主要区别就是是否调用opencv将追踪的图片展示出来

5. visualizer.run方式开始执行追踪流程

fame_callback

1.生成检测框列表

1.根据序列化信息、帧序号、最小检测框高度这些参数生成当前帧的序列化列表

2. 将那些置信度小于最小置信度阈值的检测框剔除

2.执行非极大值抑制

什么是非极大值抑制

4.执行追踪器进行预测操作（predicte函数）

3. 更新检测框列表

遍历每一个track的每一个操作分别执行其predict

Track:predict

KalmanFilter：predict

5.执行更新操作（update函数）

1. 根据之前预测的结果，进行匹配操作

三种状态的转换关系

原文链接：deepsort 算法的原理与代码解析