东南大学自动化学院

机器视觉第一次课程设计报告

——目标跟踪

姓　　名： 董胜豪　　**学　　号：** 08020328

目录

1 设计需求 2

2 算法设计 2

2.1 流程分析 2

2.2 滑动搜索模板匹配 EKF 3

2.3 MOSSE 5

2.4 DaSiamPRN 7

2.5 模型评估（GOT-10k） 9

3 结果分析 11

4 总结 13

1 设计需求

自行采集或网络下载不同场景的多个视频文件，场景数量不少于三种，每个场景视频时长不少于十秒。选择不少于三种视觉目标跟踪算法，编写程序，进行目标跟踪比测实验，评估算法性能，包括跟踪精度与计算负荷。

2 算法设计

2.1 流程分析

视觉目标跟踪 (visual object tracking) 是计算机视觉 (computer vision) 领域的一个重要研究问题。通常来说，视觉目标跟踪是在一个视频的后续帧中找到在当前帧中定义的感兴趣物体 (object of interest) 的过程，主要应用于一些需要目标空间位置以及外观（形状、颜色等）特性的视觉应用中。本文旨在对跟踪做一个尽量全面、细致和具有时效性的综述，研究跟踪的定义、应用、架构、算法以及评估等方面的内容。

目标跟踪算法按照不同的分类方法可以分为很多类别，如按提取的特征分类、特征提取方法分类、最优候选框匹配模型分类等。研究者大多是按照最优候选框匹配模型分类，这里也按此给出目标跟踪的算法分类。按此方法，可以分为生成式算法和判别式算法。前者较老、后者较优。

关于生成式方法，其核心思想即衡量前一帧的预测目标与当前帧候选框的相似度，然后选择最为相似的候选框作为当前帧的跟踪结果（即预测目标在当前帧的位置）。生成式方法被进一步分成下述三类：

1）空间距离：使用空间距离衡量相似度，将跟踪问题转化为空间距离最小化问题。经典算法包括 IVT 和 ASLA，利用像素灰度值的欧氏距离来选择最相似的候选框作为当前帧的预测目标。

2）概率分布距离：使用概率分布距离衡量相似度，将跟踪问题转化为概率分布距离最小化问题。经典算法包括 CBP 和 FRAG，通过比较颜色直方图分布的巴氏距离来选择最相似的候选框。

3）综合方法：代表算法为 MeanShift 和 CamShift，模糊了相似度匹配的距离衡量。MeanShift 利用颜色直方图分布进行聚类，得到预测目标的中心位置，并结合候选框信息确定当前帧的预测目标。CamShift 通过引入图像矩计算相似度匹配，获得目标尺度和旋转信息，进一步提高算法性能。

如前所述，判别式方法侧重于将目标视作前景，然后将其从其它被视作背景的内容中分离出来。从某种程度上来说，判别式方法应用了分类算法的思想，将跟踪问题转换成二分类问题。众所周知，基于经典机器学习（即不包含深度学习的机器学习）和深度学习的算法对于分类问题有着非常出色的表现，因此，这些算法的思想被引入跟踪问题的解决方案是非常自然的事情。此外，判别式方法的本质仍然是解决匹配问题，而一种解决匹配问题非常有效的方法就是相关 (correlation)，即用一个模板与输入进行相关操作，通过得到的响应（输出）来判断该输入与模板的相似程度，即相关性。因此，基于相关操作的算法也同样被引入跟踪

问题的解决方案。判别式方法被进一步分成下述三类：

1）经典机器学习方法 (machine learning)

应用机器学习算法的思想将目标作为前景从背景中提取出来的方法。利用此方法的经典算法包括 STRUCK (STRUCtured output tracking with Kernels)和 Tracking-Learning-Detection (TLD)。STRUCK 和 TLD 算法分别采用经典机器学习算法中的支持向量机 (support vector machine)和集成学习 (ensemble learning) 进行分类，并采取了一系列优化方法来提高算法的性能。

2）相关滤波方法 (correlation filter)

应用相关操作计算候选框与预测目标匹配度的方法。

3）深度学习方法 (deep learning)

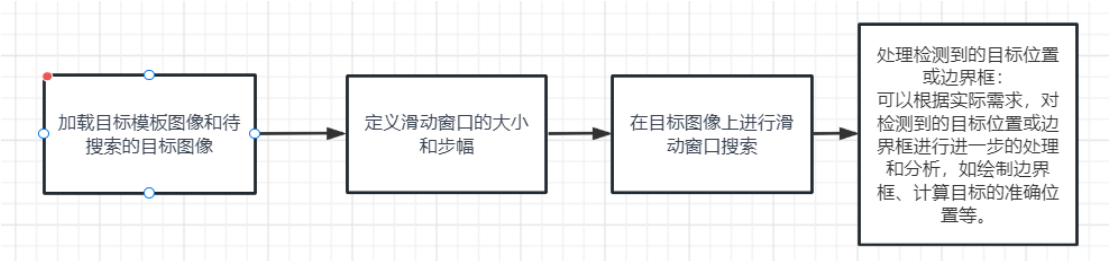
上述提到的应用深度学习算法的思想将目标作为前景从背景中提取出来的方法。

而在本此课程设计过程，采用三种算法实现目标跟踪，分别为基于模板匹配的 EKF 跟踪、MOSSE 目标跟踪以及基于 DaSiamPRN 网络的目标跟踪。

2.2 滑动搜索模板匹配 EKF

滑动模板搜索（Sliding Template Search）是一种基于滑动窗口的目标检测方法，用于在图像中搜索目标的位置。它通常用于在图像中定位目标的多个实例，并返回它们的位置或边界框。

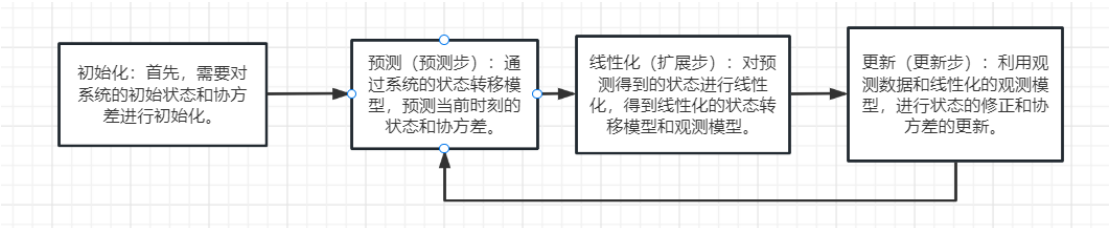
滑动模板搜索的基本思想是通过在图像上以固定尺寸的窗口进行滑动，对每个窗口内的图像区域进行特征提取和分类。滑动窗口通常以固定的步幅在图像上滑动，以覆盖整个图像。算法具体实现如下图所示



而搜索过程中匹配的准则采用相关性匹配，相关性匹配的基本思想是通过计算目标图像与模板图像之间的像素值相关性来确定匹配程度。相关性匹配假设目标和模板之间存在线性相关性，即目标区域与模板之间的像素值变化趋势相似。

而 EKF（Extended Kalman Filter）是一种基于卡尔曼滤波器的扩展形式，用于估计系统状态的一种递归滤波器。它是一种经典的状态估计方法，广泛应用于各种领域，包括机器人导航、目标跟踪、信号处理等。

EKF 适用于非线性系统，其中系统的状态变量和观测变量之间的关系可以通过非线性函数描述。与传统的卡尔曼滤波器相比，EKF 通过在卡尔曼滤波器的预测和更新步骤中使用线性化的状态转移和观测模型来处理非线性性质。具体的算法实现如下图所示：



我们将二者相结合，使得滑动搜索作为 EKF 中的观测阶段，搜索所得结果输入 EKF 算法中进行状态更新，算法中的状态为当前目标框的中心值。

2.2.1 代码实现

模板匹配的实现采用 opencv 中的 cv2.matchTemplate(target_img, template_img, method),使用该函数可以大大简化孙发的复杂程度，通过改变函数中的 method 参数可以实现不同的方法进行模板匹配，在本此课程设计中将其设置为 **cv2.TM_CCORR** 来实现相关性匹配，具体如下图所示：

```
def find_area(frame0,frame1,i,box):
    x1, y1 = int(box[0]), int(box[1])  # 区域左上角坐标
    x2, y2 = int(x1 + box[3]), int(y1 + box[3])  # 区域右下角坐标
    target_image=frame0
    current_image=frame1
    # 提取目标区域的像素值
    target_region = target_image[y1:y2, x1:x2]
    # 将目标区域转换为灰度图像
    gray_target = cv2.cvtColor(target_region, cv2.COLOR_BGR2GRAY)
    # 将当前图像转换为灰度图像
    gray_current = cv2.cvtColor(current_image, cv2.COLOR_BGR2GRAY)
    # 在当前图像中寻找与目标区域直方图相似的区域
    result = cv2.matchTemplate(gray_current, gray_target, cv2.TM_CCOEFF_NORMED)
    v,i=find_top_values(result,10)
    index=np.array(i)
    points=index.transpose()
    target_point=np.array([[int(box[0]),int(box[1])]])
    target_point=np.int64(target_point)
```

而 EKF 的实现参考上述流程图中的算法流程，具体如下

预测步：

```
def run_EKF_model(state, P, Q, dt):

    state[0] = state[0] + dt * state[2]
    state[1] = state[1] + dt * state[3]
    state[2] = state[2]
    state[3] = state[3]

    J = np.matrix('1.0,0.0,0.0,0.0;\
                   0.0,1.0,0.0,0.0;\
                   0.0,0.0,1.0,0.0;\
                   0.0,0.0,0.0,1.0')
    J[0, 2] = dt
    J[1, 3] = dt

    P = J * P * (J.transpose()) + Q

    return state, P, J
```

扩展步：

```
def run_EKF_measurement(state, measurement, P):

    H = np.matrix('1.0,0.0,0.0,0.0; \
                   0.0,1.0,0.0,0.0')

    R = np.matrix('5.0,0.0;\
                   0.0,5.0')

    z = measurement - H * state
    HPH = H * P * (H.transpose())
    S = HPH + R
    invS = np.linalg.inv(S)
    K = P * (H.transpose()) * np.linalg.inv(S)

    state = state + K * z
    P = P - P * (H.transpose()) * np.linalg.inv(S) * H * P

    if (debug_print == True):...

    return state, P
```

2.3 MOSSE

MOSSE（Minimum Output Sum of Squared Error）是一种用于目标跟踪的相关滤波器设计方法。它是由 David S. Bolme 等人于 2010 年提出的。MOSSE 算法结合了最小二乘支持向量机（LSSVM）和傅里叶变换的概念。

MOSSE 算法的核心思想是首先根据第一帧图像框选的目标构建一个响应，该响应在所绘制的目标框的中心处的响应值最大，向四周缓慢衰减（二维高斯分布）。然后我们希望找到一个滤波器使得图像和这个滤波器进行相关运算之后刚好得到的就是这个响应，那么就可以根据响应值最大处得到目标的位置了。当新的一帧图像进来时，用之前得到的滤波器与新的图像进行相关运算，就可以得到新的目标位置了。

2.3.1 原理分析

相关滤波意思就是现在在第一帧图像中框选了一个目标，然后对这个目标训练一个滤波器（大小相同）使得其输出响应 g（大小相同）在中间值最大。其中输入图像给定，响应图也是可以直接生成的。一般都是用高斯函数，中间值最大，旁边逐渐降低。

然后滤波器的值推导过程类似于机器学习的线性回归单应矩阵可以将一个平面上的点映射到另一个平面上，它由一个 3x3 的矩阵表示，其形式如下：

$$min_{H^*} = \sum_{i=1}^{m} | H^*F_i - G_i |^2$$

就是要找一个滤波器 H*使得其上式的结果最小，其实也就是找到一个滤波器，使得其

响应在中间的值最大，这就是利用了相关滤波器的原理。

为了防止目标在在空间中微小变换产生影响，对目标进行一定的旋转变换后生成一个数据集对滤波器进行训练，找到符合上式中的 H 卷积核。

对 H*求导，令其为 0，可得到闭式解：

$$H^* = \frac{\sum_i G_i \odot F_i^*}{\sum_i F_i \odot F_i^*}$$

为了能够利用后续视频帧对 H*进行更新，MOSSE 则给出了下面的方法：

$$H_i^* = \frac{A_i}{B_i}$$

$$A_i = \eta G_i \odot F_i^* + (1-\eta) A_{i-1}$$

$$B_i = \eta F_i \odot F_i^* + (1-\eta) B_{i-1}$$

其中η表示学习率，η越大，则对当前帧保留的信息越多，对历史信息保留的越少，论文中给出的η的最佳值为 0.125。。

2.3.2 代码实现

Opencv 中集成了 MOSSE 目标跟踪器在 OpenCV 中，可以使用 `TrackerMOSSE_create()` 函数创建 MOSSE 目标跟踪器。具体代码如下

初始化：

```
def init(self, image, box):
    self.box = box
    global trackers
    trackers = cv2.legacy.MultiTracker_create()
    # image and init box
    img_pil2cv = np.array(image)
    img_pil2cv = cv2.cvtColor(img_pil2cv, cv2.COLOR_RGB2BGR)
    # HxWxC 读取图像
    self.image = img_pil2cv
    frame=self.image
    trackers.add(cv2.legacy.TrackerMOSSE_create(), frame, self.box)
```

跟踪更新：

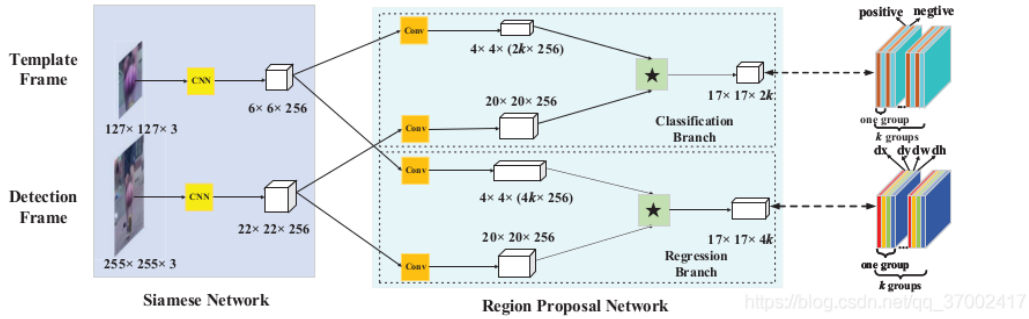
```
def update(self, image):
    # image and init box
    img_pil2cv = np.array(image)
    img_pil2cv = cv2.cvtColor(img_pil2cv, cv2.COLOR_RGB2BGR)
    # HxWxC 读取图像
    self.image = img_pil2cv
    frame = self.image
    global trackers
    #frame = cv2.resize(frame, (600, int(frame.shape[0] * 600 / frame.shape[1])), cv2.INTER_AREA)
    (success, boxes) = trackers.update(frame)
    self.box=boxes[0]
    return self.box
```

2.4 DaSiamPRN

DaSiamPRN 是一种用于视觉目标跟踪的算法，全称为“Deeper and Spatial-Aware Siamese Network for Real-Time Visual Tracking”。它是 SiamPRN（Siamese Proposal Refinement Network）的改进版本。
DaSiamPRN 算法最初由中国科学技术大学的研究团队提出。具体来说，该算法由 Jiangang Wang、Yanwei Fu、Tianyu Guan、Changxin Gao 和 Xin Yang 等人于 2018 年提出并发表在国际计算机视觉领域的顶级会议——欧洲计算机视觉大会（European Conference on Computer Vision，简称 ECCV）上。自那时以来，DaSiamPRN 已经成为目标跟踪领域的重要算法之一，并得到了广泛的研究和应用。
与 SiamPRN 相比，DaSiamPRN 在特征提取和相似度计算方面进行了改进，使其在目标跟踪的准确性和实时性方面表现更好。这使得它在许多视觉跟踪任务中都能取得很好的效果，并在学术界和工业界得到广泛应用。

2.4.1 SiamPRN

SiamPRN 是一种基于 Siamese 网络的目标跟踪算法，全称为"Siamese Proposal Refinement Network"。Siamese 网络是一种双子网络结构，由两个共享权重的子网络组成。在目标跟踪任务中，SiamPRN 使用 Siamese 网络来估计目标的位置和外观信息。
SiamPRN 算法的核心思想是将目标模板与候选框进行相似度比较，从而确定最可能的目标位置。它通过在目标模板和候选框之间计算相似度分数来衡量它们的相似程度。为了提高准确性，SiamPRN 还引入了一个细化网络（Refinement Network）来对候选框进行进一步的优化和精炼。



网络结构：如上图所示，用于特征提取的孪生子网络，作者仍采用 AlexNet，共包含 5 层卷积。模板和搜索两个分支在 CNN 中共享参数，将$\varphi(z)$，$\varphi(x)$分别表示为孪生子网络的输出特征映射(6×6×256 和 22×22×256) RPN 网络的两个分支分别进行前景背景分类和 proposal 回归，每个分支接受 Siamese 网络的两个输出$\varphi(z)$和$\varphi(x)$通过卷积层(改变通道维度)后的特征图作为输入。

2.4.2 DaSiamPRN

孪生网络的缺点：

- 特征提取网络提取的特征只能区分前景和非语意背景，这里解释下非语义背景，可以理解为不含有用信息的背景，比如一张图像上有二人个(A、B)和一辆汽车 C，跟踪的对象是 A，前景就是 A，B 和 C 就是包含具体语义背景。当图像存在类间干扰 B 时跟踪器很容易发生偏移。
- 训练数据集的物体类别较少，不足以训练具有泛化能力的特征表达，SiamRPN 使用 vid(20 classes)和 ytbb(30 classes)两个训练集，极端情况下，跟踪器漂移到汽车 C 处。
- 大部分跟踪器都是短期跟踪，短期跟踪从另一种角度讲是跟踪过程中，目标总是出现。

为了解决训练数据集类别少的问题，DaSiamRPN 引入了两个新的数据集，coco 和 det，极大地扩展了正样本对的类别，如下图，这些类别在实际应用中，几乎不现实，但使用它们来训练网络可以有效增强特征表达的泛化能力。并且通过对数据集增强技术(平移、旋转、运动模糊等)，可以使用检测数据集静止的图像生成跟踪领域动态的图像。

DaSiamRPN 提出了一种感知干扰物增量学习(Distractor-aware Incremental Learning)，可以有效地将泛化的特征表示转移到某特征视频领域，而不仅仅使用以往 Siam 跟踪器的度量方式。

关键代码：

网络初始化：

```python
# 对网络进行初始化设置，输入参数：图像im，目标位置pos，目标尺寸sz，网络参数net
# 初始化设置中，设定了目标的初始位置
# 返回网络参数，以字典形式返回
def SiamRPN_init(im, target_pos, target_sz, net):
    # 设置一个空的字典
    state = dict()
    # 设置跟踪器的相关参数
    p = TrackerConfig()
    # 将网络的参数加载至跟踪模型中
    p.update(net.cfg)
    # 将图像的尺寸加载至state中
    state['im_h'] = im.shape[0]
    state['im_w'] = im.shape[1]

    if p.adaptive:  # 初始化设置为True
        if ((target_sz[0] * target_sz[1]) / float(state['im_h'] * state['im_w'])) < 0.004:
            p.instance_size = 287  # small object big search region
        else:
            p.instance_size = 271   # 用于设置instance_size，为score_size计算做准备

        # 与TrackerConfig类中计算方式一致
        p.score_size = (p.instance_size - p.exemplar_size) / p.total_stride + 1

    # 生成候选框尺寸锚点
    p.anchor = generate_anchor(p.total_stride, p.scales, p.ratios, int(p.score_size))

    # 计算图像均值
    avg_chans = np.mean(im, axis=(0, 1))
```

跟踪更新：

```python
# 返回跟踪目标的位置target_pos 尺寸target_size 得分score
def SiamRPN_track(state, im):
    # 接收网络参数
    p = state['p']
    net = state['net']
    avg_chans = state['avg_chans']
    window = state['window']
    # 主要接收上一帧目标跟踪的位置以及尺寸
    target_pos = state['target_pos']
    target_sz = state['target_sz']

    # 更新搜索区域，决定本帧的搜索区域（根据上一帧的检测结果来设置本帧搜索区域）
    wc_z = target_sz[1] + p.context_amount * sum(target_sz)
    hc_z = target_sz[0] + p.context_amount * sum(target_sz)
    s_z = np.sqrt(wc_z * hc_z)
    # 获取尺度变化率
    scale_z = p.exemplar_size / s_z
    # 调整搜索区域
    d_search = (p.instance_size - p.exemplar_size) / 2
    pad = d_search / scale_z
    s_x = s_z + 2 * pad

    # extract scaled crops for search region x at previous target position
    x_crop = Variable(get_subwindow_tracking(im, target_pos, p.instance_size, round(s_x), avg_chans).unsqueeze(0))

    # 获得本帧预测结果，target_pos-目标位置 target_sz-目标尺度 score-置信分数
    # Ps: target_sz * scale_z表示本帧的搜索区域大小
    target_pos, target_sz, score = tracker_eval(net, x_crop.cuda(), target_pos, target_sz * scale_z, window, scale_z,
    target_pos[0] = max(0, min(state['im_w'], target_pos[0]))
    target_pos[1] = max(0, min(state['im_h'], target_pos[1]))
```

2.5 模型评估（GOT-10k）

2.5.1 模型评估指标

对于单目标跟踪模型的评估，可以使用以下方法和指标：

准确度评估：

IoU（Intersection over Union）：计算预测边界框和真实边界框之间的重叠程度。常用的 IoU 阈值包括 0.5、0.75 和 0.95。较高的 IoU 值表示更准确的目标定位。

精确度（Precision）：计算模型预测为正的边界框中与真实边界框的重叠比例。

召回率（Recall）：计算模型成功检测到的正边界框与真实正边界框的重叠比例。

实时性评估：

推理时间：衡量模型在给定硬件设备上处理每个帧的时间。较快的推理速度对于实时目标跟踪至关重要。

鲁棒性评估：

长期跟踪：评估模型在长时间跟踪任务中的性能。模型应该能够在较长的时间内保持目标的准确跟踪。

目标形变：评估模型对于目标尺度变化、姿态变化等形变的鲁棒性。

光照变化：评估模型在不同光照条件下的跟踪能力。

2.5.2 GOT-10k

GOT-10k 是一个常用的用于目标跟踪的数据集，它是一个基于 YouTube 视频的大规模

目标跟踪数据集。GOT-10k 数据集由超过 1,000 个独立的视频序列组成，涵盖了多个不同的目标类别和场景。并且利用该工具可以实现跟踪器对多种数据集的性能进行评估。
以下是关于 GOT-10k 数据集的一些重要信息：

数据集内容：GOT-10k 数据集包含了 1,000 多个视频序列，总计超过 1,50,000 帧。每个视频序列都提供了目标的边界框注释，用于表示目标在每一帧中的位置。

目标类别和场景：GOT-10k 数据集包含了多个目标类别，包括人、动物、车辆、自行车等。它涵盖了各种不同的场景，如室内、室外、城市、乡村等。

数据集划分：GOT-10k 数据集按照训练集和测试集进行划分。训练集包含 900 个视频序列，而测试集包含 100 个视频序列。训练集和测试集中的视频序列是互斥的，这样可以确保模型在未见过的数据上进行评估。

评估指标：GOT-10k 数据集使用了常见的目标跟踪评估指标，如准确度（Accuracy）、成功率（Success Rate）和归一化预测精确度（Normalized Precision）。这些指标用于衡量目标跟踪算法在数据集上的性能。

挑战性：GOT-10k 数据集对于目标跟踪算法来说是具有挑战性的。它包含了目标形变、尺度变化、遮挡、光照变化等复杂情况，对模型的准确性和鲁棒性提出了较高的要求。

GOT-10k 数据集成为目标跟踪领域的重要基准数据集之一，被广泛用于评估和比较不同的目标跟踪算法的性能。研究人员可以使用该数据集进行模型训练、模型评估和算法改进，以推动目标跟踪技术的发展。

代码实现

```python
from got10k.trackers import Tracker

class IdentityTracker(Tracker):
    def __init__(self):
        super(IdentityTracker, self).__init__(
            name='IdentityTracker',  # tracker name
            is_deterministic=True    # stochastic (False) or deterministic (True)
        )

    def init(self, image, box):
        self.box = box

    def update(self, image):
        return self.box
```

用该工具包定义自己需要测试的跟踪器时，只需要简单的定义 IdentityTracker 类里面的 init 和 update 函数即可。

- init 函数：接收的是测试视频序列当中的初始帧以及初始帧的 bbox。
- update 函数：接收的是后续帧，返回的是更新后的 bbox。

```python
from got10k.experiments import ExperimentGOT10k

# ... tracker definition ...

# instantiate a tracker
tracker = IdentityTracker()

# setup experiment (validation subset)
experiment = ExperimentGOT10k(
    root_dir='data/GOT-10k',    # GOT-10k's root directory
    subset='val',               # 'train' | 'val' | 'test'
    result_dir='results',       # where to store tracking results
    report_dir='reports'        # where to store evaluation reports
)
experiment.run(tracker, visualize=True)

# report tracking performance
experiment.report([tracker.name])
```

3 结果分析

由于视频跟踪结果在此不好展示，具体视频的跟中结果见文件中，在此主要对模型评估的结果进行分析，首先对三个跟踪器在 GOT-10k 数据集上的跟踪结果进行评估，代码如下：

```python
# setup experiment (validation subset)
experiment = ExperimentGOT10k(
    root_dir='test_data',    # GOT-10k's root directory
    subset='val',               # 'train' | 'val' | 'test'
    result_dir='results_GOT10k',       # where to store tracking results
    report_dir='reports_GOT10k'        # where to store evaluation reports
)
#experiment.run(tracker, visualize=True)

#report tracking performance
experiment.report(['MOSSE','EKF_Tracker','DaSiamPRN'])
```

得到的跟踪结果保存在/reports_GOT10k 目录下，过程中生成两个文件，一个是 png 文件，为绘制的 success_plot 图，第二个是详细的性能分析，储存在 performan.josn 文件中。

Performance

得到的性能评估如下

```
"EKF_Tracker": {
    "overall": {
        "ao": 0.13512057497383212,
        "sr": 0.07492545926709628,
        "speed_fps": 33.678961119795424,
```

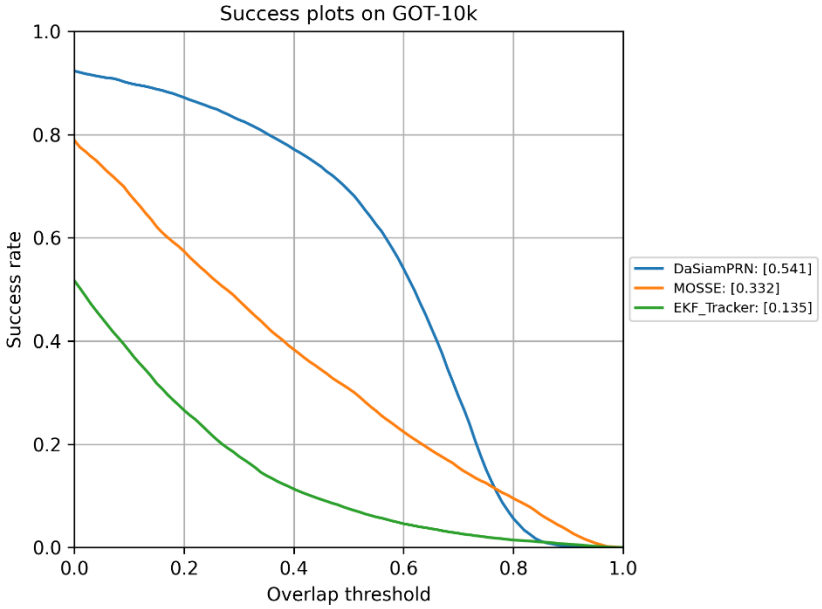
```
"MOSSE": {
    "overall": {
        "ao": 0.3315307726043444,
        "sr": 0.3076368183129749,
        "speed_fps": 147.0172499261589,
```

```
"DaSiamPRN": {
    "overall": {
        "ao": 0.5414423280879417,
        "sr": 0.691593728960277,
        "speed_fps": 12.128582498338837,
```


总结成下表

Name	Average overlap	Success Rate	speed_fps
EKF	0.14	0.07	33.7
MOSSE	0.33	0.31	147
DaSiamPRN	0.54	0.69	12.1

而 Success plots 如下图所示



4 总结

从算法的复杂度来看，模板匹配 EKF 显然是最简单的，当然，其效果也是最差的。就准确度来说，基于孪生网络的 DaSiamPRN 深度学习的算法显然是最高的，但是其 FPS 却很低，而较低的 FPS 代表其处理速度较慢，实时性较差。而 MOSSE 的 FPS 非常高，这也符合我们在上面介绍中的分析，而其准确度稍低，在我看来这样的算法更加可以应用在工程实践之中，尽管深度学习的准确度更高，但是过低的处理速度导致其在实际工程上很难落地实现。

从上面的曲线来看，我们可以很明显的看出 DaSiam>MOSSE>模板匹配 EKF，显然模板匹配的 EKF 算法是最初的目标跟踪算法，而 MOSSE 最初提出在 2010 年 CVPR 会议，而基于深度学习的 DaSiamPRN 算法则在 2018ECCV 提出。随着计算机硬件的发展和深度学习算法的优化，未来可能会有更高效的深度学习目标跟踪算法出现，能够兼顾准确度和处理速度的要求。

综上所述，从算法的复杂度、准确度和处理速度来看，每种算法都有其优势和劣势，适用于不同的应用场景：

- 模板匹配的 EKF 算法：它是最简单的目标跟踪算法，处理速度较快，但准确度相对较低。由于其简单性和较高的处理速度，可以在实时性要求较高的场景中使用，但在复杂背景、目标形变等情况下效果较差。
- MOSSE 算法：它是基于孪生网络的目标跟踪算法，具有较高的处理速度和相对较高的准确度。在处理速度和准确度之间取得了一定的平衡，适用于需要实时性和一定准确度的应用场景。
- DaSiamPRN 算法：它是基于深度学习的目标跟踪算法，具有最高的准确度，但处理速度较慢。由于深度学习算法的复杂性和计算需求较高，导致处理速度较慢，适用于对准确度要求较高，实时性要求相对较低的场景。

根据具体的应用需求和资源限制，选择合适的目标跟踪算法是非常重要的。在实际工程中，通常需要综合考虑准确度、处理速度、实时性等因素，选择最适合特定应用场景的目标跟踪算法。