REVIEW OF OBJECT DETECTION INSTANCE SEGMENTATION & TRACKING VIA PARTICLE FILTER

SUBMITTED BY : **SELAHADDİN HONİ**

SCHOOL NUMBER : **040160046**

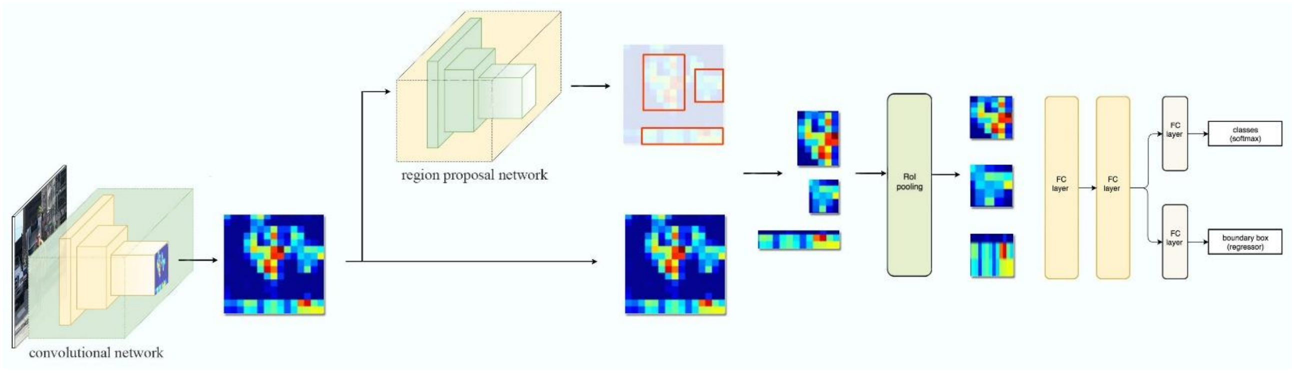
DIGITAL VIDEO PROCESSING

SPRING 2021

REVIEW OF OBJECT DETECTION INSTANCE SEGMENTATION & TRACKING VIA PARTICLE FILTER

Content

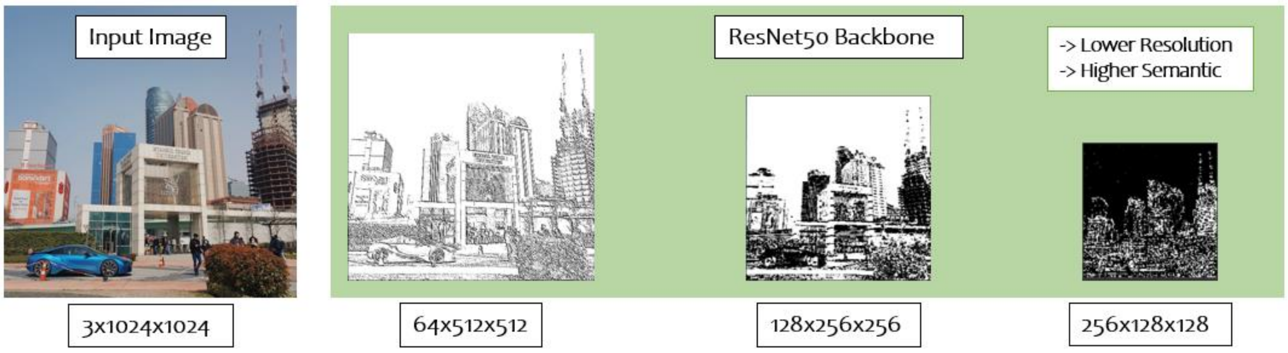
1. Object Detection: Faster R-CNN [1]



1.1. Backbone & Region Proposal Network

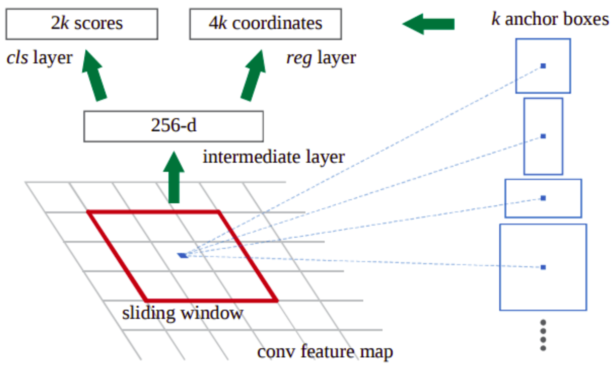
Faster RCNN consists of two stages:

In the first stage, the feature maps which are generated in the backbone get into the **region proposal network (RPN)** as input; then, RPN decides which regions of the input image are more likely to be an object and it returns the location of these regions (RoI) with their **objectness score**.



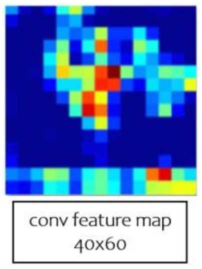
Visualization of the backbone network (ResNet50) feature maps

The second stage, inputs the proposed regions and applies a **crop & resize** process (RoI Pooling) to make the feature maps fixed size to engage them into fully connected (FC) layers. The **final classification** and **bounding box regression** occur after these FC layers.



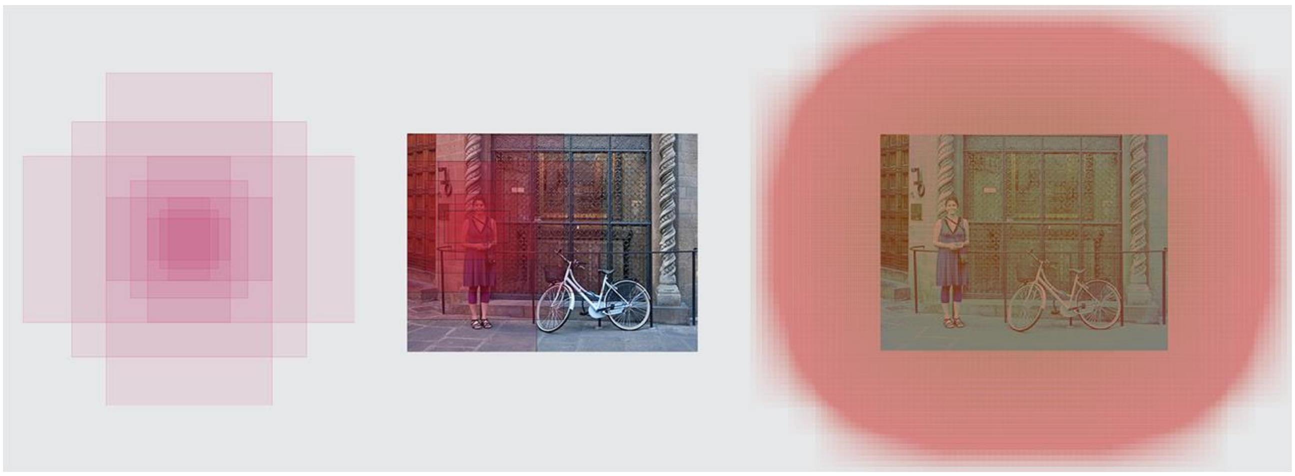
RPN slides a 3x3 window on the convolutional feature map and generates **k anchor boxes** for each point in the map. All boxes have objectness score and four boundary coordinates tuned with ground truth region.

k is selected 9 as a combination of 3 different scales and 3 different aspect ratios.



In numerically, if the convolutional feature map has a dimension of 40x60 the number of generated anchor boxes will be approximately 20,000. (40x60x9)

However, most of the anchor boxes overflows the image size as seen in the below figure. These are eliminated and the number of remaining anchor boxes are 6000.



An achor box having higher than 0.7 IoU with ground truth labeled as **positive** and lower than 0.3 labeled as **negative samples**. Rest of them are not carried to upper layers.

RPN Loss Functions

$$\mathcal{L} = \mathcal{L}_{\text{cls}} + \mathcal{L}_{\text{box}}$$
$$\mathcal{L}(\{p_i\}, \{t_i\}) = \frac{1}{N_{\text{cls}}} \sum_i \mathcal{L}_{\text{cls}}(p_i, p_i^*) + \frac{\lambda}{N_{\text{box}}} \sum_i p_i^* \cdot L_1^{\text{smooth}}(t_i - t_i^*)$$

In the classification we try to minimize the loss of predicting the objectness of a region by using a **log loss function** over two classes.

$$\mathcal{L}_{\text{cls}}(p_i, p_i^*) = -p_i^* \log p_i - (1 - p_i^*) \log(1 - p_i)$$

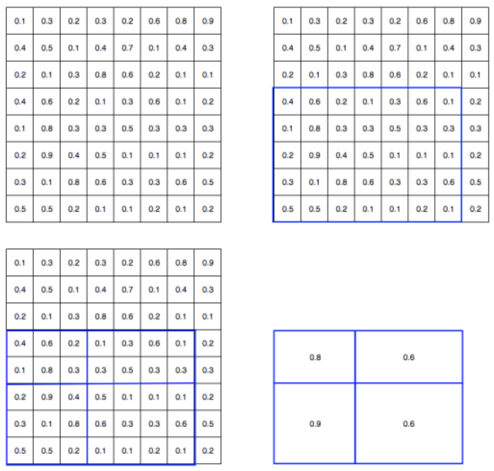
Simply, in the bounding box regression smooth L1 loss is applied to the x, y, width and height predictions and their ground truths.

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

Symbol	Explanation
p_i	Predicted probability of anchor i being an object.
p_i^*	Ground truth label (binary) of whether anchor i is an object.
t_i	Predicted four parameterized coordinates.
t_i^*	Ground truth coordinates.
N_{cls}	Normalization term, set to be mini-batch size (~256) in the paper.
N_{box}	Normalization term, set to the number of anchor locations (~2400) in the paper.
λ	A balancing parameter, set to be ~10 in the paper (so that both $\mathcal{L}_{\text{cls}}$ and $\mathcal{L}_{\text{box}}$ terms are roughly equally weighted).

1.2. RoI Pooling

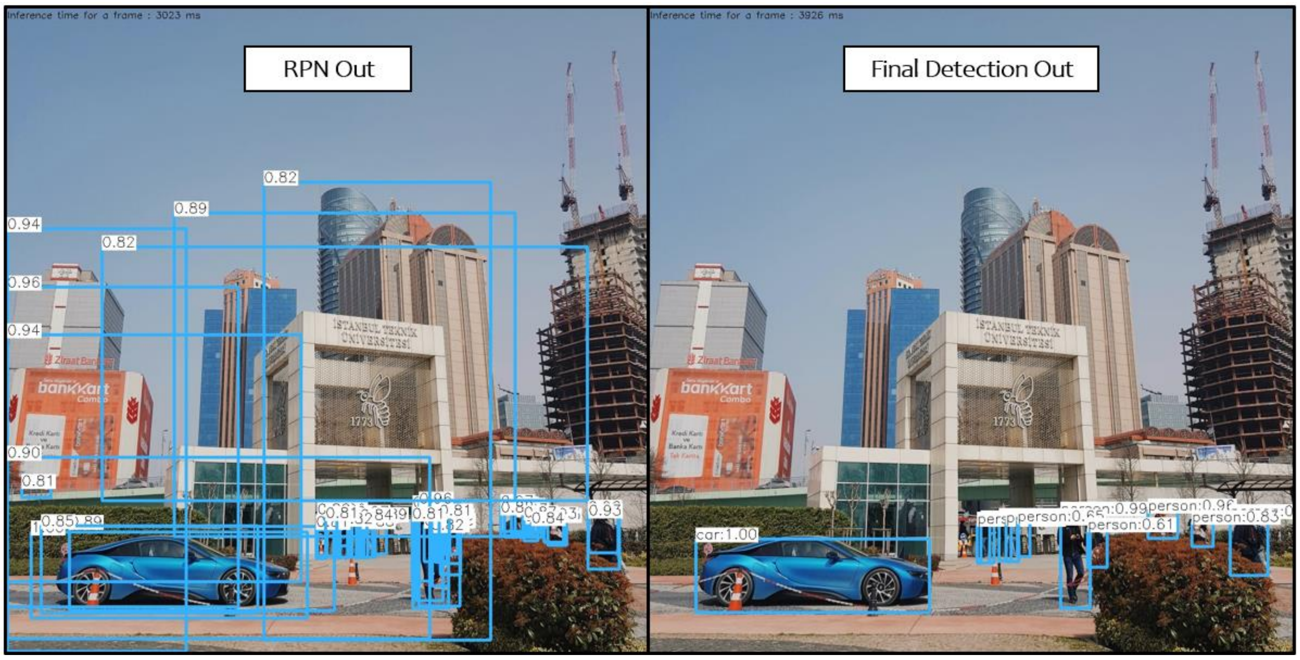
The second stage classifier keeps **fully connected layers** inside to learn actual class of the proposed object regions. Since the input dimension of FC layers cannot be changed; the **extracted RoIs** in previous network must be adjusted to **fixed-size**.



In the above illustration, top left is a feature map and blue box in the top right is a RoI proposed by RPN. We try to reduce the dimension of this RoI to 2x2. Bottom left is an attempt to divide this RoI into 4 equal pieces as much as possible. Then, regular MaxPooling is applied to obtain 2x2 map in the end.

1.3. Loss Functions

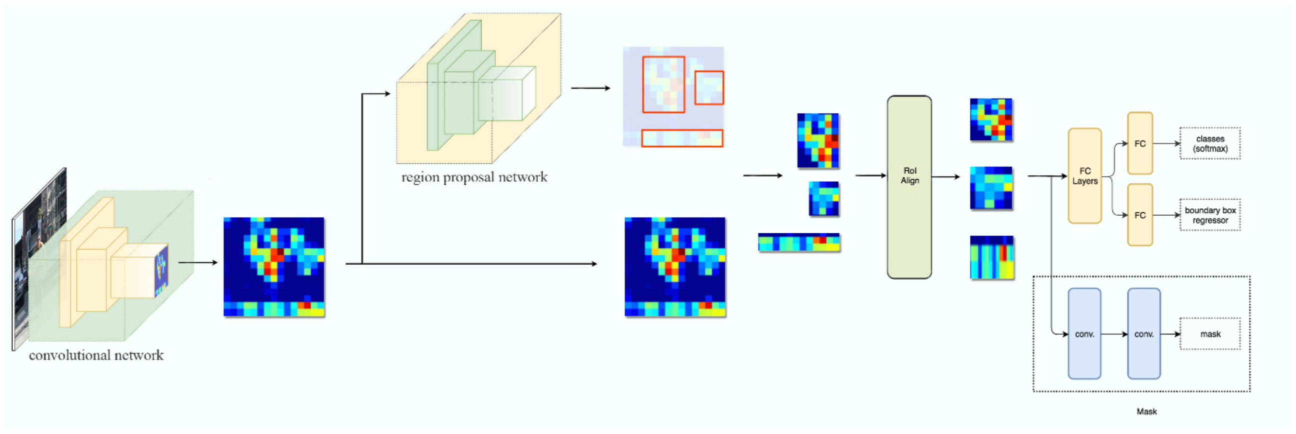
The final decision of Faster R-CNN and also this second stage is very similar to the output of RPN. Bounding box regression **fine-tunes** the **coordinate offsets** of the proposed RoIs again. However, in the classification part, log loss function over multi-classes is used instead of binary-classes like object or background.


RPN Out
Final Detection Out
Inference time for a frame : 3023 ms
Inference time for a frame : 3926 ms
car:1.00

Outputs of Region Proposal Network (left) and Second-Stage Final Detection (right)

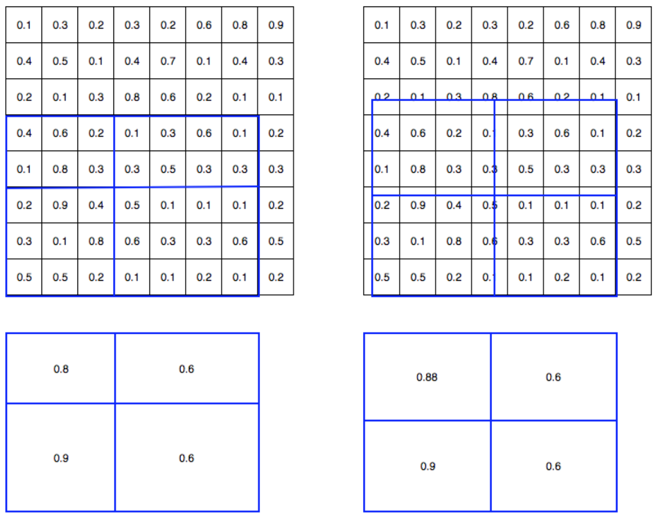
***

2. Instance Segmentation: Mask R-CNN [2]



2.1. RoI Align

First of all, Mask R-CNN is an extended version of Faster R-CNN by adding a **mask branch** to the head of the network. Another difference is changing the RoI pooling with RoI Align to increase the precision of object boundaries.



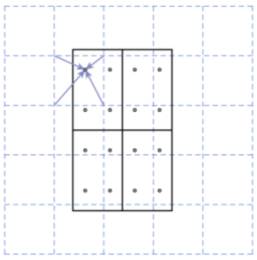
0.1	0.3	0.2	0.3	0.2	0.6	0.8	0.9
0.4	0.5	0.1	0.4	0.7	0.1	0.4	0.3
0.2	0.1	0.3	0.8	0.6	0.2	0.1	0.1
0.4	0.6	0.2	0.1	0.3	0.6	0.1	0.2
0.1	0.8	0.3	0.3	0.5	0.3	0.3	0.3
0.2	0.9	0.4	0.5	0.1	0.1	0.1	0.2
0.3	0.1	0.8	0.6	0.3	0.3	0.6	0.5
0.5	0.5	0.2	0.1	0.1	0.2	0.1	0.2

0.1	0.3	0.2	0.3	0.2	0.6	0.8	0.9
0.4	0.5	0.1	0.4	0.7	0.1	0.4	0.3
0.2	0.1	0.3	0.8	0.6	0.2	0.1	0.1
0.4	0.6	0.2	0.1	0.3	0.6	0.1	0.2
0.1	0.8	0.3	0.3	0.5	0.3	0.3	0.3
0.2	0.9	0.4	0.5	0.1	0.1	0.1	0.2
0.3	0.1	0.8	0.6	0.3	0.3	0.6	0.5
0.5	0.5	0.2	0.1	0.1	0.2	0.1	0.2

0.8	0.6
0.9	0.6

0.88	0.6
0.9	0.6

In short, RoI pooling forces cropping of the feature map by **integer** values; on the other hand, RoI align allows to use of **floating points**.

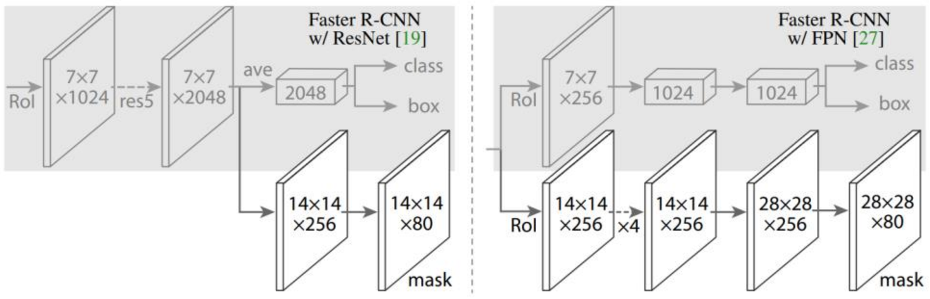


The dashed grid is a feature map and solid lines is a proposed RoI. The dots on the RoI can be represented with the 4 neighboring points of the feature map by **bilinear interpolation**.

This approach resulted in drastically **increase of performance** such that +7.3 point is observed in mean AP seen below.

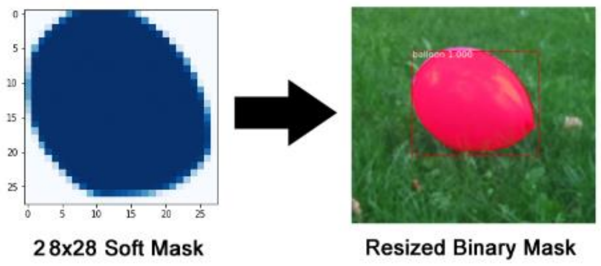
	AP	AP_{50}	AP_{75}	AP^{bb}	AP^{bb}_{50}	AP^{bb}_{75}
RoIPool	23.6	46.5	21.6	28.2	52.7	26.9
RoIAlign	**30.9**	**51.8**	**32.1**	**34.0**	**55.3**	**36.4**
	+7.3	*+ 5.3*	*+10.5*	*+5.8*	*+2.6*	*+9.5*

2.2. Mask Branch



The head architecture of the Faster R-CNN now has a mask branch which generates **Kxmxm** mask output, where K is the number of classes and m is the width and height size of the mask.

m= 28 w/FPN ; K=80 for COCO Dataset

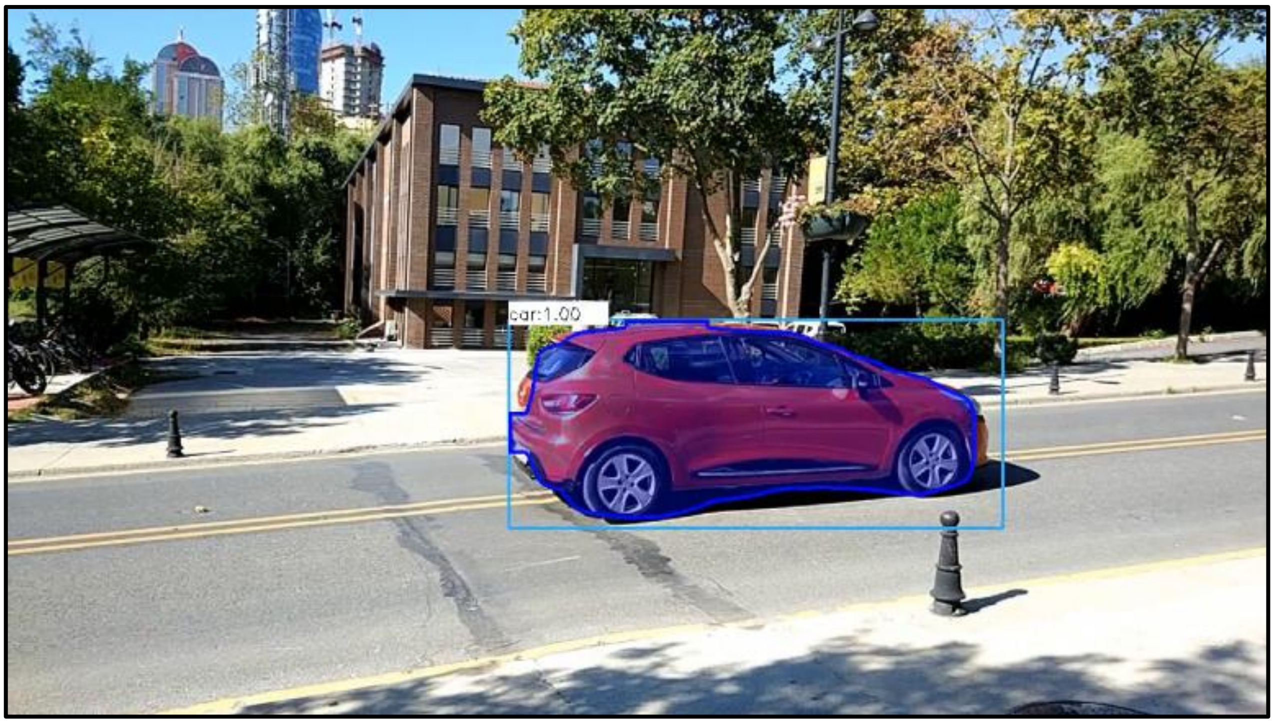


From there, we can say this branch creates mask without knowing the label of the class and proposes masks for each class in the inference time.

$$\mathcal{L} = \mathcal{L}_{\text{cls}} + \mathcal{L}_{\text{box}} + \mathcal{L}_{\text{mask}}$$

The mask branch appends one more term to the loss function. Classification and bounding box regression losses are same with Faster R-CNN. Mask loss is an average **binary cross-entropy loss** that compares the pixels of resized ground truth to reach 0 for background and 1 for object.

$$\mathcal{L}_{\text{mask}} = -\frac{1}{m^2} \sum_{1 \le i,j \le m} \left[y_{ij} \log \hat{y}^k_{ij} + (1 - y_{ij}) \log(1 - \hat{y}^k_{ij}) \right]$$

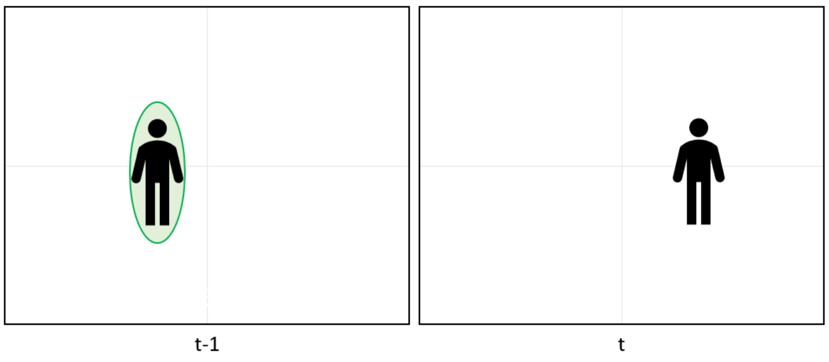


Mask R-CNN output of 'car' sample from ITU MSPR Dataset

***

3. Object Tracking: Color-Based Particle Filter [3]

We are going to discuss the algorithm over a simple scenario.



Suppose we have two sequential frames and tracking is initialized somehow in previous frame, t-1. Our aim is finding the best particle covering the human figure in the current frame.

3.1. State Parameters and Motion Model

First of all, we need to figure out the parameters of the state: **x** and **y** are the center coordinates of the target ellipse; **ẋ** and **ẏ** are the changes of the coordinates over time; **H_x** and **H_y** are the height and width of the ellipse; **ȧ** is the scale change.

$$\boldsymbol{s}_{t-1} = \{x, y, \dot{x}, \dot{y}, H_x, H_y, \dot{a}\}$$

Next state depends on the previous one via multiplication with **state transition matrix**. State transition matrix determines the relation of the state parameters of two sequential states.

$$\boldsymbol{s}_t = A\ \boldsymbol{s}_{t-1} + \boldsymbol{w}_{t-1}$$

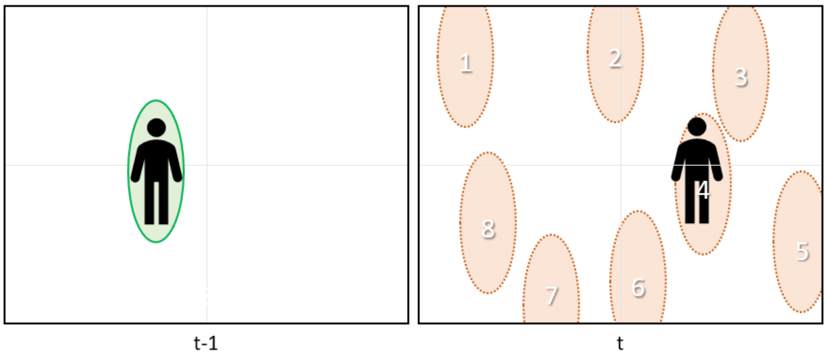
$$\boldsymbol{s}_t = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{t-1} \\ y_{t-1} \\ \dot{x}_{t-1} \\ \dot{y}_{t-1} \\ H_{x_{t-1}} \\ H_{y_{t-1}} \\ \dot{a}_{t-1} \end{bmatrix} + \begin{bmatrix} w_{x_{t-1}} \\ w_{y_{t-1}} \\ w_{\dot{x}_{t-1}} \\ w_{\dot{y}_{t-1}} \\ w_{H_{x_{t-1}}} \\ w_{H_{y_{t-1}}} \\ w_{\dot{a}_{t-1}} \end{bmatrix}$$

A: state transition matrix
w: stochastic process noise

By using the expression above, we will generate **N = 8** particles over **Gaussian probability density function** as the stochastic process noise. These particles will be called the state hypotheses.

$$state\ hypotheses = \left\{\boldsymbol{s}_t^{(1)}, \boldsymbol{s}_t^{(2)}, \ \ldots\ , \boldsymbol{s}_t^{(7)}, \boldsymbol{s}_t^{(8)}\right\}$$

In the below figure, you will see the **generated state hypotheses** visually. Here, we can say that if the variance of the Gaussian distribution is less; the hypotheses would generated closer to the previous target.



According to the particle filter theory, every hypothesis must have a probability, in other words weight.

$$\pi^{(n)} = p(\mathbf{z}_t | X_t = \mathbf{s}_t^{(n)})$$

If the probabilities of each particle are known, it is easy to reach the next estimation. The weighted sum of these particles is the **expected state** in current time.

$$E[S] = \sum_{n=1}^{N} \pi^{(n)} \mathbf{s}^{(n)}$$

We need to investigate the observation model to calculate the probabilities.

3.2. Observation Model: Color Distribution

In the reference article, it is proposed to use the color distributions of the regions (ellipses) to measure the similarities. The hypotheses similar to the target will have higher probabilities.

There is a suggestion here while calculating the color distribution. Far pixels from the center of the region are more likely to be background. Thus, these pixels should contribute to the color distribution less.

$$k(r) = \begin{cases} 1 - r^2 & : \quad r < 1 \\ 0 & : \quad \text{otherwise} \end{cases}$$

r: distance from the region center

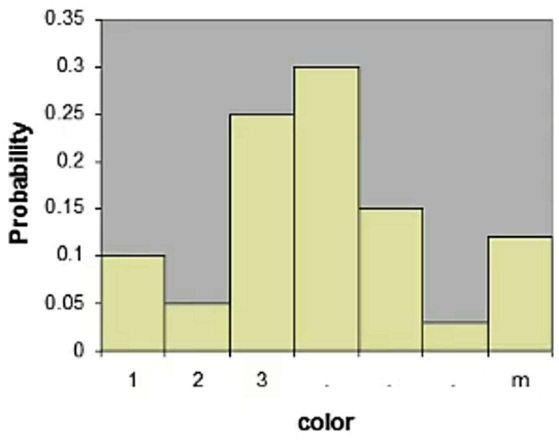
Also, the reason of using ellipses instead of rectangular is the same, too. The pixels near the boundaries of the elliptic regions have more chance to be the feature of the object compared to rectangular regions.

In the summation, all pixels in the region (**i=1 to I**) are evaluated and weighted via **k(r)** function. **y** is the center of each hypothesis region. (RGB color space is used in this histogram calculation and **u** is the bin size.)

$$p_{\mathbf{y}}^{(u)} = f \sum_{i=1}^{I} k\left(\frac{\|\mathbf{y}-\mathbf{x}_i\|}{a}\right) \delta[h(\mathbf{x}_i)-u]$$

$$a = \sqrt{H_x^2 + H_y^2}$$

This color distribution is a template for what is expected before normalization **f** is applied.



After the summation; it is multiplied with **f** normalization factor to make the sum of the probabilities in the color distribution 1.

$$f = \frac{1}{\sum_{i=1}^{I} k\left(\frac{\|\mathbf{y}-\mathbf{x}_i\|}{a}\right)}$$

$$\sum_{u=1}^{m} p_{\mathbf{y}}^{(u)} = 1$$

Now, there are N=8 color distributions of the hypothesis regions, each called **p**. Meanwhile, the color distribution of the target region is also calculated via same method and called **q**.

A popular method to measure the similarities of color distribution is: **Bhattacharyya**. The **similarity coefficient ρ** gets the value 1 for exactly same comparisons; conversely the minimum similarity is resulted in 0.

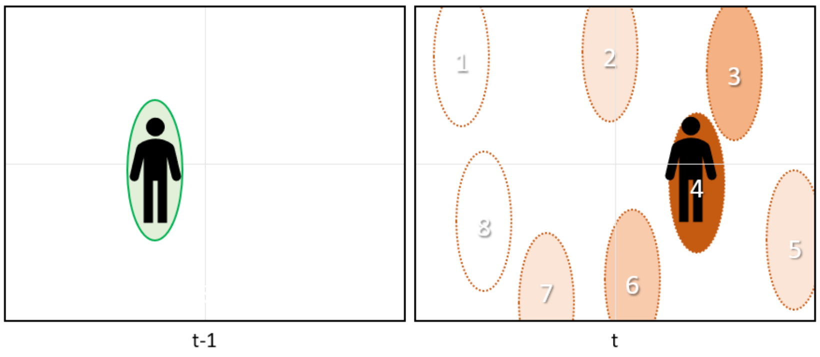
$$\rho[p,q] = \sum_{u=1}^{m} \sqrt{p^{(u)} q^{(u)}}$$

We want to convert this similarity to its opposite, the **distance**.

$$d = \sqrt{1 - \rho[p,q]}$$

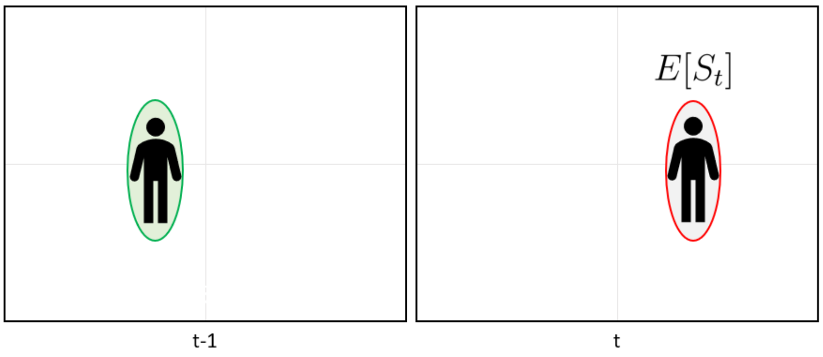
From now on, we have obtained the distance of each hypothesis regions from the target region. Therefore, we can assign the probabilities (weight) of each particle. Again, we will use **Gaussian** distribution.

$$\pi^{(n)} = \frac{1}{\sqrt{2\pi\sigma}} e^{-\frac{d^2}{2\,\sigma^2}} = \frac{1}{\sqrt{2\pi\sigma}} e^{-\frac{(1-\rho[p_{\mathbf{s}^{(n)}},q])}{2\,\sigma^2}}$$



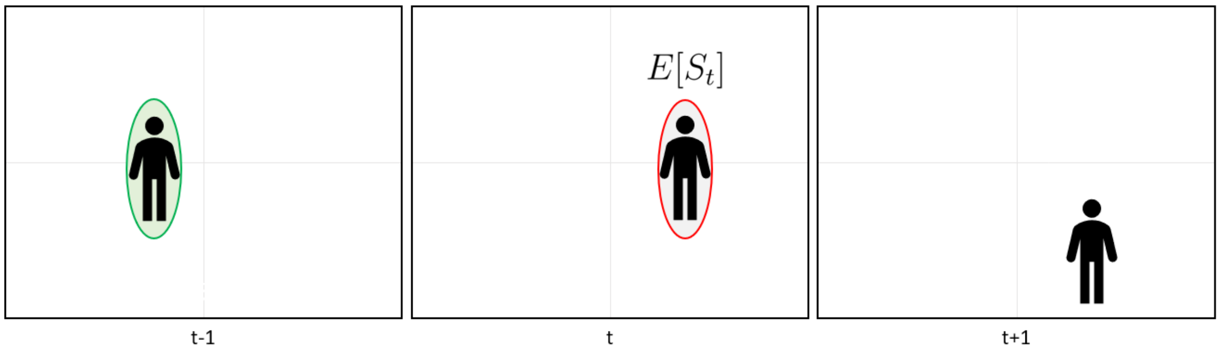
Weighted sum of the particles will give us the new state estimation.

$$E[S] = \sum_{n=1}^{N} \pi^{(n)} \mathbf{s}^{(n)}$$

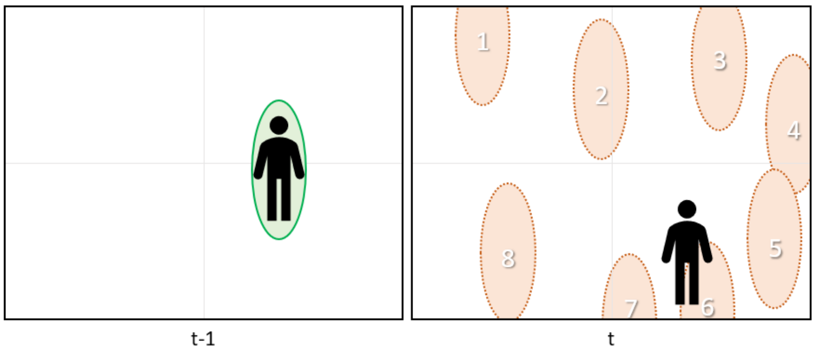


3.3. Resampling

Assume we have successfully estimated the current state; now, we are looking for the **next state**. In the coming frame the human figure will move downside.



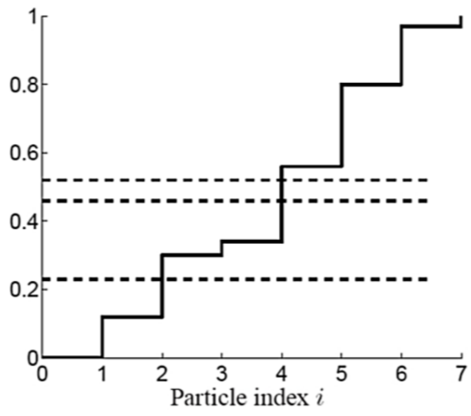
Let's push the time forward (t = t+1) and propose the state hypotheses, again. The proposing in next iterations concept is called **resampling**.



In the resampling, we are sampling based on the previous probabilities. First, we will calculate the **normalized cumulative probabilities**.

$$
\begin{aligned}
c_{t-1}^{(0)} &= 0 \\
c_{t-1}^{(n)} &= c_{t-1}^{(n-1)} + \pi_{t-1}^{(n)} \\
c_{t-1}^{\prime(n)} &= \frac{c_{t-1}^{(n)}}{c_{t-1}^{(N)}}
\end{aligned}
$$

It is illustrated in below figure. [4]



This figure means, if a uniformly distributed random number (**r**) is generated in the range of [0,1] and searched the correspond in particle indices; the particles having larger probabilities also have larger probability to be resampled.

3.4. Target Update

To adapt our particle filter to changes like illumination conditions and the visual angle; there is a forgetting process in the algorithm. For each iteration, the color distribution of the target is updated with below expression.

$$q_t^{(u)} = (1-\alpha)\, q_{t-1}^{(u)} + \alpha\, p_{E[S_t]}^{(u)}$$

Where **a** is the forgetting factor and **p$_{E[S]}$** is the estimated color distribution.

4. Comparative Analyis & Results

I have worked with two videos from two datasets to evaluate performance: 'cat' video from **ITU MSPR Group** and 'graduate' from **Visual Object Tracking 2020** dataset. **Accuracy** of the tracker (average of IoU scores), **IoU score** per frame and **success rate** have been reported in this performance evaluation.

For each video, the bounding boxes have estimated with **3 different approaches**: using only Mask R-CNN, using only CPF tracker and using both together with certain confidence to detector.

In short, using both of them performed the best results of course. Obtained results placed in the next section. Videos are available in the 'videos' folder.

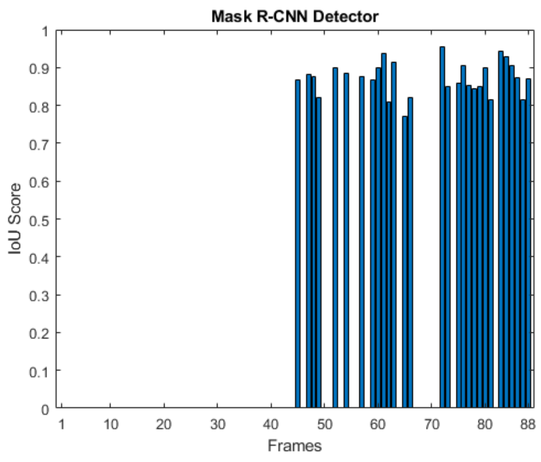
Hyper-parameters of Mask R-CNN:
confidence threshold = 0.3
keep_top_k in RPN = 50
nms threshold in RPN = 0.7
nms threshold in second stage = 0.6

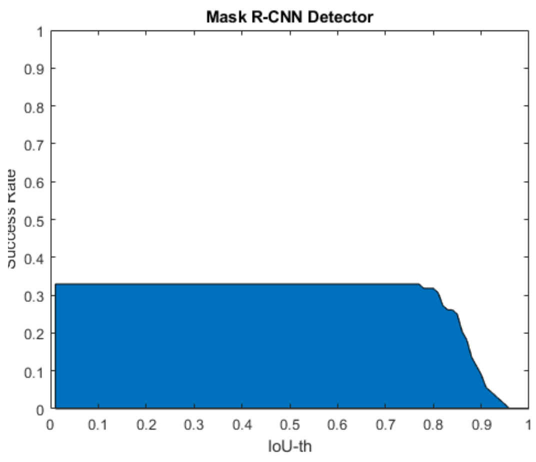
Hyper-parameters of CPF Tracker:
400 particles
standard deviations for x, y, dx, dy, scale = [2, 2, .5, .5, .1]

4.1. Dataset: ITU MSPR Group

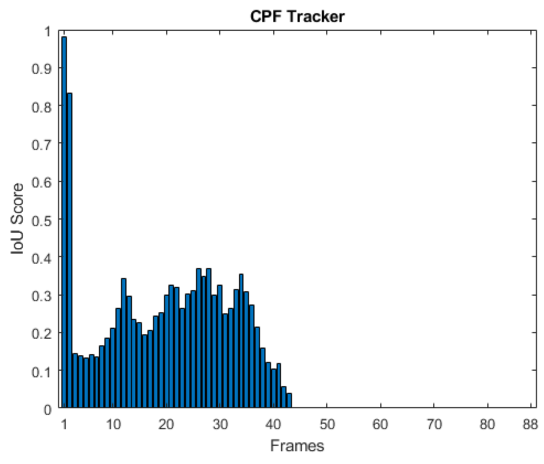
In the cat video from ITU MSPR Group, Mask R-CNN has very poor performance to detect the cat in the initial frames of the video, actually Mask R-CNN thinks black cat is a cow. Thus, **accuracy** of using only Mask R-CNN resulted in **0.28**.

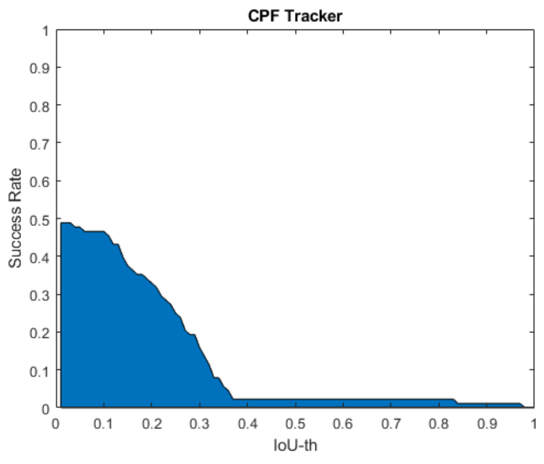
First below figure is the IoU scores for each frame and the right one is the measurement of the success if the IoU threshold is changed from 0 to 1.



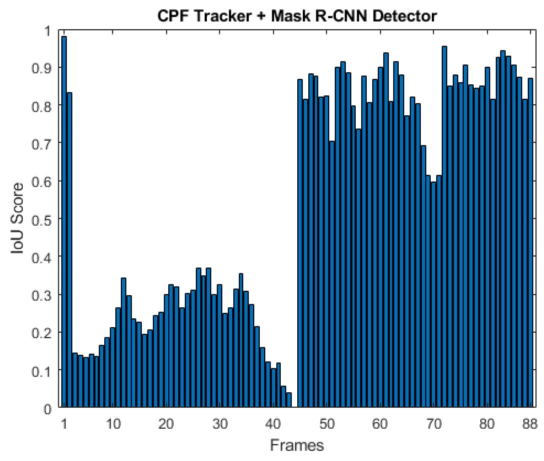


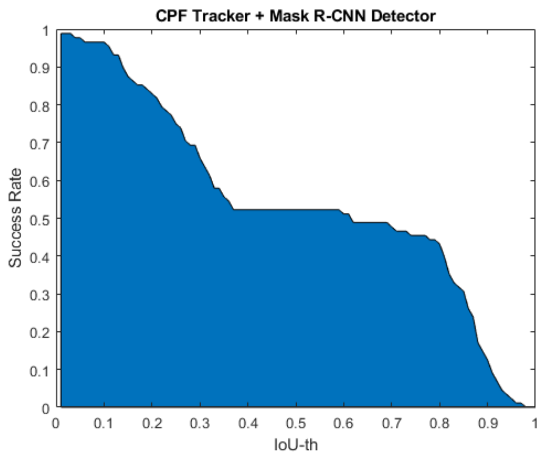
From the right figure, we can say localizations are very successful if it detects. However, there are lots of gaps in the frames which decrease the success rate much.





CPF tracker initialized with ground truth and performed worse with an accuracy of **0.13.** But, instead of Mask R-CNN tracking is realized in the initial frames.





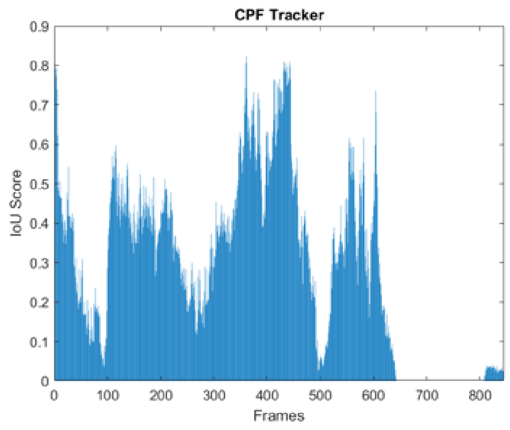
Using both together with a certain confidence to detector gave **0.54** accuracy. Initial frames are estimated from the tracker and when it has lost the target Mask R-CNN reinitialized the tracker. Also, the tracker supported the bounding box estimations in the 65~75 frames where detection gaps occurred in the use of only Mask R-CNN.

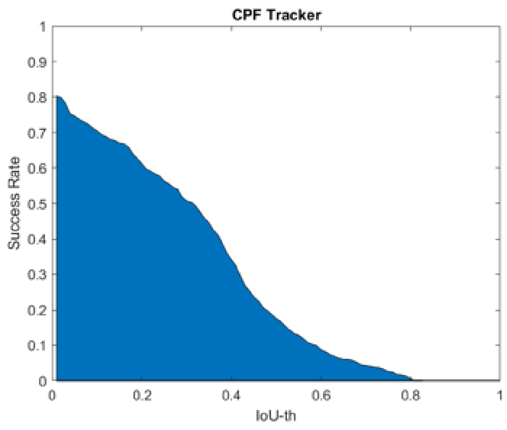
***

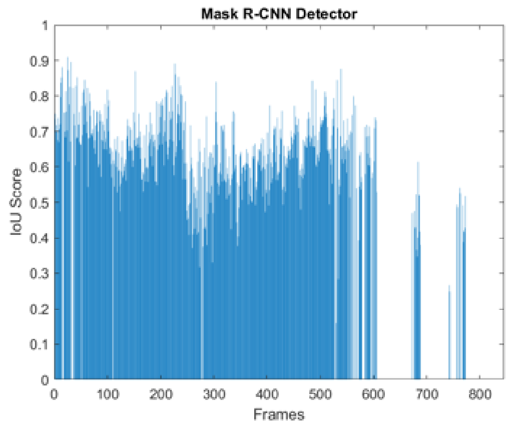
4.2. Dataset: Visual Object Tracking (VOT 2020)

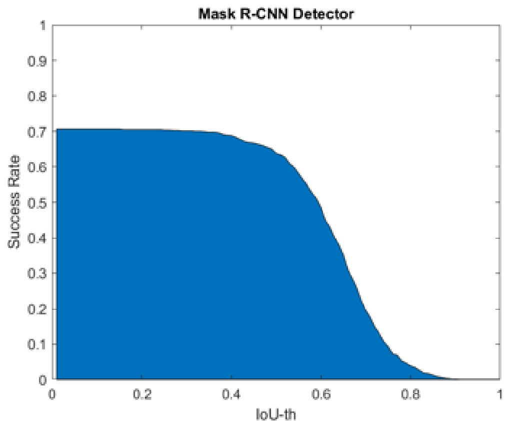
In the graduate video from VOT 2020 Benchmark, similar results have been observed. However, there is no significant detection gaps in this evaluation such that resulted with close accuracy scores between the use of only Mask R-CNN and use of together.

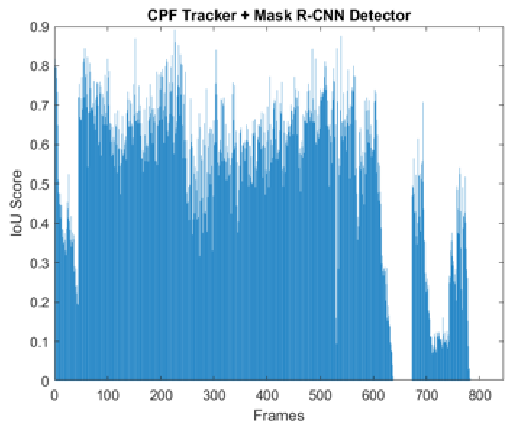
Accuracies for the three approaches are **0.29, 0.45 and 0.49**, respectively.

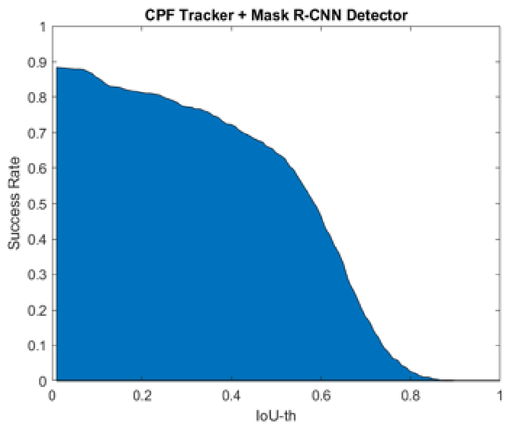












5. Appendix

5.1. An iteration step of the color-based particle filter

```
Given the sample set $S_{t-1}$ and the target model
$q = f \sum_{i=1}^{I} k\left(\frac{\|\mathbf{x}_i\|}{a}\right) \delta[h(\mathbf{x}_i) - u]$,
perform the following steps:
(1) Select $N$ samples from the set $S_{t-1}$ with probability $\pi_{t-1}^{(n)}$:
    (a) calculate the normalized cumulative probabilities $c'_{t-1}$
        $c_{t-1}^{(0)} = 0$
        $c_{t-1}^{(n)} = c_{t-1}^{(n-1)} + \pi_{t-1}^{(n)}$
        $c_{t-1}'^{(n)} = \frac{c_{t-1}^{(n)}}{c_{t-1}^{(N)}}$
    (b) generate a uniformly distributed random number $r \in [0, 1]$
    (c) find, by binary search, the smallest $j$ for which $c_{t-1}'^{(j)} \geq r$
    (d) set $\mathbf{s}_{t-1}'^{(n)} = \mathbf{s}_{t-1}^{(j)}$
(2) Propagate each sample from the set $S'_{t-1}$ by a linear stochastic
    differential equation:
    $\mathbf{s}_t^{(n)} = A\, \mathbf{s}_{t-1}'^{(n)} + \mathbf{w}_{t-1}^{(n)}$
    where $\mathbf{w}_{t-1}^{(n)}$ is a multivariate Gaussian random variable
(3) Observe the color distributions:
    (a) calculate the color distribution
        $p_{\mathbf{s}_t^{(n)}}^{(u)} = f \sum_{i=1}^{I} k\left(\frac{\left\|\mathbf{s}_t^{(n)} - \mathbf{x}_i\right\|}{a}\right) \delta[h(\mathbf{x}_i) - u]$
        for each sample of the set $S_t$
    (b) calculate the Bhattacharyya coefficient for each sample
        of the set $S_t$
        $\rho[p_{\mathbf{s}_t^{(n)}}, q] = \sum_{u=1}^{m} \sqrt{p_{\mathbf{s}_t^{(n)}}^{(u)} q^{(u)}}$
    (c) weight each sample of the set $S_t$
        $\pi_t^{(n)} = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(1-\rho[p_{\mathbf{s}_t^{(n)}}, q])}{2\sigma^2}}$
(4) Estimate the mean state of the set $S_t$
    $E[S_t] = \sum_{n=1}^{N} \pi_t^{(n)} \mathbf{s}_t^{(n)}$
```

5.2. Actual implementation of assigning weights over Bhattacharyya Distance [5]

```cpp
// Calculate the likelyhood for a particular region
float ParticleFilter::calc_likelyhood(Mat& image_roi, Mat& lbp_roi, Mat& target_hist, bool use_lbp )
{
    static const float LAMBDA = 20.f;
    static Mat hist;

    calc_hist(image_roi, lbp_roi, hist, use_lbp);
    normalize(hist, hist);

    float bc = compareHist(target_hist, hist, CV_COMP_BHATTACHARYYA);
    float prob = 0.f;
    if(bc != 1.f) // Clamp total mismatch to 0 likelyhood
        prob = exp(-LAMBDA * (bc * bc) );
    return prob;
}
```

6. References

[1] **Ren, S., He, K., Girschick, R., & Jian, S.**, 2016. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks

[2] **He, K., Gkioxari, G., Dollar, P., & Girschick, R.**, 2018. Mask R-CNN.

[3] **Nummiaro, K., Koller-Meir, E., & Van Gool, L.**, 2002. An adaptive color-based particle filter. Image and vision computing, 21 (1), 99-110.

[4] **Schön, T. B.**, 2010. Solving nonlinear state estimation problems using particle filters – an engineering perspective, Division of Automatic Control Technical Report, LiTH-ISY-R-2953, Linköping University.

[5] **Kevin Schluff.**, 2016. Kschluff (Particle filter), GitHub repository, https://github.com/mcv-m6-video/mcv-m4-2016-Team2

[6] **Image Source.** What do we learn from region based object detectors (Faster R-CNN, R-FCN, FPN)? | by Jonathan Hui | Medium

[7] **Image Source.** Demystifying Object Detection and Instance Segmentation for Data Scientists - MLWhiz

[8] **Image Source.** Object Detection for Dummies Part 3: R-CNN Family (lilianweng.github.io)

[9] **Image Source.** Splash of Color: Instance Segmentation with Mask R-CNN and TensorFlow | by Waleed Abdulla | Matterport Engineering Techblog

[10] **Image Source.** Image segmentation with Mask R-CNN | by Jonathan Hui | Medium