Action-Decision Region Proposal Network for Visual Tracking with Reinforcement Learning

Fuyang Zhang

I. Tracking in a proposal region based on action decisions

A. Overview

The existing tracking algorithms are all tracking at a fixed resolution of the fixed lens, in part because the mature datasets are based on static scenes. This type of tracking algorithm can perform good tracking tasks on static cameras such as surveillance cameras in parking lots. But in fact, our real-life target tracking is difficult to complete long-term tracking in a static scene, because the target is likely to run out of our scene after a while. In the actual process, the tracking camera must be rotated and scaled as necessary. So our algorithm considers both the camera attitude adjustment and tracking tasks. In general, we use deep reinforcement learning to train the camera's attitude adjustments and use existing tracking algorithms for tracking.

Our algorithm can be split into two parts, **Region Proposal with Zooming Network**(RPZ_Net) and **Tracker**. RPZ_Net is used to iterate on the original image for cropping and scaling to generate an image that is more suitable for tracking (this series of actions can be seen as a simulation of the camera pose adjustment), and then pass the image to the tracker for tracking. In our experiments, we tried a variety of trackers, e.g. MDNet, KCF, etc. The test results will be explained in Evaluation chapter(III-B).

B. Problem Settings

In the actual process, we do not want to move the camera frequently, since frequent movement will cause unnecessary workload for the camera. So our algorithm encourages as few adjustments as possible. We use RPZ_network to iteratively generate proposal until certain criteria are meet. Since most of the existing datasets, like OTB and VOT, are static scenes, we actually generate a rough proposal on the current image and resize the target frame to a fixed resolution(150x150) to simulate the zoom of the camera. A general description of the entire network is shown in Fig.1.

Basically, We formulize our problem as a Markov Decision Process(MDP), which is defined by a 4-tuple $(\mathcal{S}, \mathcal{A}, f, r)$, where state s $\in \mathcal{S}$, action a $\in \mathcal{A}$, state transition $s' = f(s, a)$, and the reward function $r(s, a)$.

Formally, a video is represented by $V = \{I_1, I_2, \cdots, I_N\}$, where I_t represents the tth frame and N denotes the number of the video. The states and actions are defined as $s_{t,l}$ and $a_{t,l}$ respectively, where $t = 1 \cdots N$, $l = 1 \cdots L_t$, and L_t denotes the number of iterations in frame t. The output image of RPZ_Net is represented by I'_t and scale it to the fixed size 150×15 before pass I'_l to the **Tracker**.

In the beginning, The RPZ_Net is initialized by the ground truth in the first frame with the location $l_1 = (x, y, w, h)$. For each frame, we first crop current image I_t with the last tracking result $l'_{t-1} = (x_{t-1}, y_{t-1}, w_{t-1}, h_{t-1})$, use it as input for the network and then extract output of fc5 layers as feature f_t. Then we concatenate f_t with **action history** vector and feed it into fc6 layer to predict next action. Action a is defined in a discrete series of motions $\{left, right, up, down, zoom\,in, zoom\,out, stop\}$ and **action history** vector contains the last ten actions we take, which is signed to zero vector at first.

Each item in fc6 layer represents the probability of selection for each action, i.e. $p(a|s_{t,l})$, $\sum p(a|s_{t,l}) = 1$, where state $s_{t,l} = \{l_{t,l}, h_{t,l}\}$ contains the location $l_{t,l}$ and **action history** $h_{t,l}$ at the kth iteration of the tth frame.

Unless the action we take is $\{stop\}$ or the number of iterations reaches to the maximum number of iterations, we continuous iterative update tracking state $s_{t,l+1} = f(s_{t,l}, a)$. Meanwhile, we add a into $h_{t,l}$, if the size of $h_{t,l}$ is larger than 10, we abandon old action using $first\,in\,first\,out$ strategy.

Next, we formulate a transition from $s_{t,l}$ to $s_{t,l+1}$. The transition is based on the action a.

- For $\{left, right, up, down\}$
  Those actions are linear movement of the bounding box. Since the objects are various, the scale of movement cannot be consistent, especially small objects can easily over-shift through bounding box. So we set the moving distance according to the size of the target. Due to the similarity of those four actions, we take $left$ action as an example. The amount of movement is defined as

$$(\triangle x, \triangle y) = \alpha(w_t, h_t) \tag{2}$$

  where α is 0.1 in our experiment. $l_{l+1} = (x_t - \triangle x, y_t, w_t, h_t)$.
- For $\{zoom\,in, zoom\,out\}$
  Zooming refers to the change in the resolution of the image, which simulates the change in the stretch of the lens during tracking. However, since we fixed the final output size of the image, the zooming action will degenerate to the larger and smaller on the width and height of the bounding box. Therefore, the movement is defined as $l_{l+1} = (x_t, y_t, w_t \pm \triangle x, h_t \pm \triangle y)$, where $(\triangle x, \triangle y)$ is defined as Eq. (2).
- For $\{stop\}$
  We stop the iteration and resize the image cropped by l_l to 150×150. This action indicates that the visual bounding box is adjusted to the optimal state.

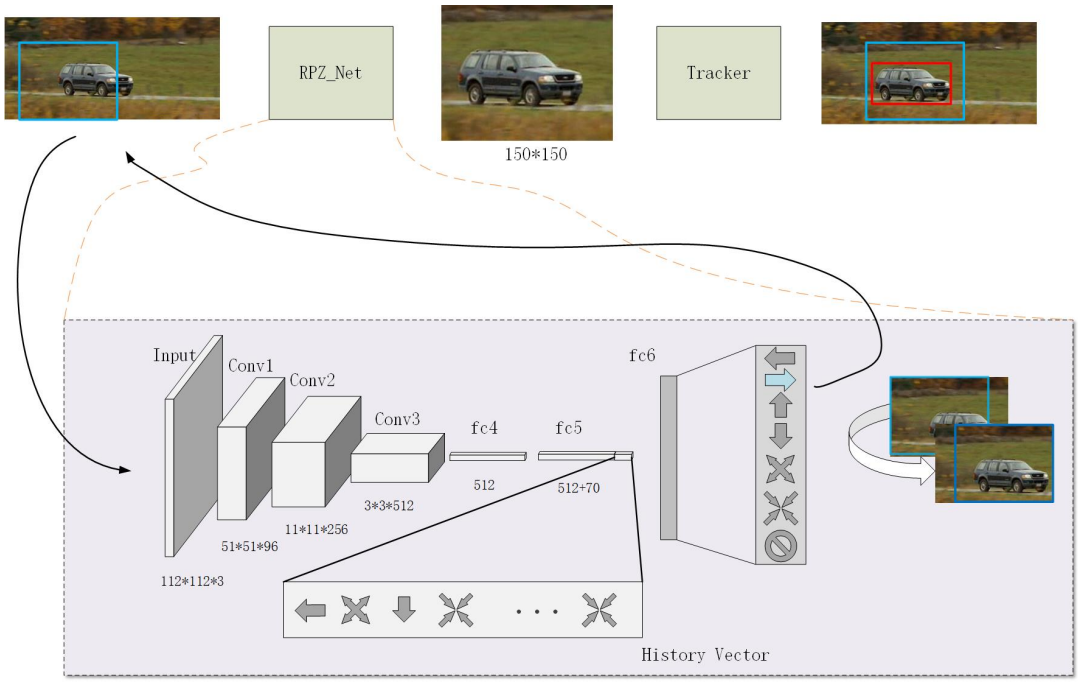

Fig. 1. The architecture of our Region Proposal with Zooming Network and overall tracking process. The RPZ_Network is an algorithmic framework on the tracking camera that provide the best view for the trackers to track the objects. The RPZ_Network initializes the tth frame with the output region at $t-1$th frame, i.e. the blue box at the upper left of the figure. After that, the direction or size of the adjustment is predicted by the 3 convolutional network layers(conv1,conv2,conv3) and 3 fully connected layers(fc4,fc5,fc6. fc5 is concatenated by history vector, which will be elaborated at I-B), and iteratively updates the position of the candidate frame. At the end of the iteration we scaled the candidate area to a fixed 150150 image to simulate the lens stretching operation in the actual process. Finally, the image is passed to the Trackers for tracking.

Generally, Given current frame, RPZ_Net iteratively generate bounding box until the action *stop*, then outputs a bounding box, and transform the bounding box into a fixed size image $I'_t(150\times150)$, which proximately locks the tracking target and zooms into a suitable scale.

C. Network architecture

Our network has three convolutional layers, which are same as the convolutional layers in VGG-M. After the convolutional layers are two fully connected layers fc4,fc5. The input of fc6 layer is fc5 concatenated with action history vector. The output of fc6 is 7 units equaled with number of actions, and is followed by a softmax layer. And the output of the softmax layer is the probability of the actions. The model we use is relatively small and the reasons are summarized below:

- Shallow CNN is more effective for spatial information and we do not need semantic information.
- Instead of tracking the object directly, our network proposals a perfect region for the tracker, so it doesn't need very precise location information.
- A smaller network is more efficient and suitable for online tracking, since less cost of calculation.

II. Training of RPZ_Net

In this section, we first describe the detailed procedure of training RPZ_Net. The whole training is divided into two parts, pretraining and reinforcement training. After describing the offline training, we describe the implementation detail.

A. Pretraining

During this stage, we only pretrain the action selection without considering the problem of reducing the number of actions. The network's parameters denote as W and are initialed by the VGG-M trained on the Imagenet.

First, we need to generate training samples. the steps are:

1) A sample p is obtained by randomly generating bounding box, which must contain the tracking target.
2) We assume that moving the target to the center of the image and scaling the target to $\frac{1}{3}\sim\frac{1}{2}$ of the entire image is optimal. Therefore we can generate actions a with gt and p, e.g. $a=left$ if moving p left can make gt close toward center or $a=zoom\,in$ if enlarge p can make the ratio between $\frac{1}{3}\sim\frac{1}{2}$.

After that, we get a sample (I,l,a). A training batch has a set of randomly selected samples $\{(I_i,l_i,a_i)\}_{i=1}^{N}$.

In the process of pretraining, we do not have action history, so we just concatenate a zero vector to fc5 vector.

The training strategy is to minimize the loss function by stochastic gradient descent.The loss function is:

$$Loss=\frac{1}{N}\sum_{i=1}^{N}L(\hat{a}_i,a_i)$$

where $\hat{a}_i = \arg\max_a p(a|S_i; W)$, L denotes the cross-entropy loss, $S_i = (I_i, l_i, a_i)$.

B. Reinforcement Training

Next, We exploit reinforcement learning to train the RPZ_Net. Firstly, we define the rewards according to the tracking performance.

Since we don't want too many camera adjustments, i.e. reduce the number iterations of the actions $\{left, right, up, down, zoom\,in, zoom\,out\}$ as long as satisfy good tracking performance. Therefore, each iteration, i.e. the actions $\{left, right, up, down, zoom\,in, zoom\,out\}$, is defined as:

$$r_{t,l} = -1 \tag{3}$$

For the action *stop*, the reward is defined by the IoU of the tracker's prediction and the ground truth. We take b_t^* as the bounding box of the prediction.

$$r_{t,L_t} = \begin{cases} 5 & \text{if } IoU(b_t^*, G) > 0.7 \\ -5 & \text{otherwise} \end{cases} \tag{4}$$

Then we define the values of each action.

$$v_{t,l} = \sum_{i=l}^{L_t} \gamma^{i-l} r_{t,i} \tag{5}$$

For each training video sequence, we use policy gradients to learn the weights.

$$W \leftarrow W + \alpha \nabla_W log\pi_W(S_t, a_t) v_t$$

III. Experiments

We implemented our tracker in Matlab with MatConvNet toolbox, and was conducted on a PC with an Intel Core i7 CPU, an 8GB Memory, and a NVIDIA GeForce GTX 1060Ti GPU with 8GB VRAM. In this condition, the proposed tracker runs about 1 fps with MDNet as **tracker** and 20 fps with KCF as **tracker** on OTB-13 Benchmark.

A. Training the Network

Pretraining stage: we randomly generate positive and negative bounding boxes. The positive samples must meet three conditions: 1) bbox>gt. 2) $\frac{bbox \cap gt}{gt} > posThre\_min$ 3) $\frac{bbox \cap gt}{bbox} < posThre\_max$, where $posThre\_min$ and $posThre\_max$ are 0.5 and 0.85 respectively in our experiment. The negative samples are those doesn't satisfy the conditions above.

For the action label of the positive samples, we first judge whether is time to *stop*, the *stop* conditions are mentioned in $Pretraining$ section. Next we try to label with the displacement operation($\{left, right, up, down\}$) that maximum IOU value. At last, if all actions mention above is no need or do not satisfy the conditions, we label with zoom action that zooming toward optimal ratio.

Reinforcement stage: We use MDNet as the **Tracker** to get tracking results as well as calculate training reward. During the training of this stage, we randomly intercept 10 consecutive frames from the video sequence as training samples and the maximum iteration we set is 15 per frame.

For K training sequences, we train the network for 10K iterations with learning rate 0.0001 for conv layers and 0.001 for fc layers, 0.9 for momentum, and 0.0005 for weight decay. We trained our RPZ_Net with VOT datasets.

B. Evaluations

We evaluate our proposed algorithm on OTB 2013. We tried two different type trackers, MDNet and KCF, as our **tracking** part, and campared them with the standard trackers in OTB2013 as well as MDNet, KCF, ADNet and some the state-of-the-art trackers.

IV. Conclusion

In this work, we proposed an approach to generally focus on the tracking object, Region Proposal with Zooming Network(RPZ_Net), which will generate a region that lock the approximate range of the target and scale the region to a better proportion. Given that region, trackers can accomplish better target tracking tasks. That means, the focus strategy with zooming in and out of the image can help tracking algorithms improve the accuracy. What's more, our RPZ_Net can smoothly work on Tracking camera in a real tracking environment too. Since the Reward we define in II-B can suppress the number of adjustments, we believe that it's an effective method for camera adjustment.