Implement tracking with ECO Algorithm

高禾, 沈詮鈞, 朱泰德
Machine Learning Group 14
Computer Science and Information Engineering, National Central University
Taoyuan, ROC

***Abstract*—**

Problem: Face recognition is usually being process after the event happened nowadays, but for more advance usage face recognition, combining real-time-tracking and dace recognition is a very critical tech-tool.

Solution: By combining face-detection (Viola-Jones), face-recognition (PCA-FLDA), and video-object-tracking in real time (ECO).

Result: It is very hard to determine whether the newly detected face is already in the database or not, but beside that everything seems to work pretty nice.

I. INTRODUCTION

Nowadays, face recognition has become a well-known technology, there are many application of face recognition. For instance, intelligent surveillance, AI, etc. But when it comes to tracking some specific person, we still need a lot of people to keep running face-recognition on different surveillance cameras, on every frames. That is way too not efficient enough to do it in everyday job.

Video-object-tracking is kind of being well-developed, so why not combine this two tech-tools together so that the vot part can cover the part when the target is not looking near the camera? That way, tracking some specific person will be a much easier task. In an other hand, police will track down the bad guys in a much higher efficient way, finding lost man in seconds, etc. So we try hard to develop this tool.

II. BACKGROUND

Some VOT-algorithms are very robust when it comes to non-real-time tracking, but that is not what we are looking for. Some are very fast but not accurate enough, and that's also not what we want for tracking heads. That's why we picked ECO_HC, a handcraft-variables version of ECO.

ECO is an advanced VOT-algorithm base on C-COT, but improved in three different aspects. (i) A factorized convolution operator, in order to reduce the number of parameters in the models. (ii) A compact model of the training sample distribution, which reduces memory and time complexity. (iii) A conservative model update strategy with improved robustness and reduced complexity.

Some of the naïve implementations are to do face-detection, face-recognition and video-object-tracking in each and every frame. But after reading the paper of ECO, we think that the third improvement on C-COT can also do on the VOT system.

III. PROPOSED METHOD

ECO is an improved algorithm base on others VOT-algorithms. Because state-of-the-art has two disadvantages, computational complexity and over-fitting. The problems are caused by three factors, high dimensionality computations make the speed slow, the large size of training set, over model updating causes low frame-rates and the degradation of the robustness. To save some memory, we discard the oldest sample, causing overfitting. ECO algorithm hope to solve these problems to have an efficient and accurate visual tracking.

ECO algorithm is based on Continuous Convolution Operator Tracker (C-COT) in order to optimize the structure. Because the continuous structure can realize convolution on continuous realm, letting us to choose the cell size more flexible in a visual feature. No need to re-sampling. We can predict the object detection scores more easily through continuous function.

First we use an interpolation to transform feature map to the continuous spatial domain, as follows:

```
xt = extract_features(im, sample_pos, sample_scale, features, global_fparams, feature_

% Project sample
xt_proj = project_sample(xt, projection_matrix);

% Do windowing of features
xt_proj = cellfun(@(feat_map, cos_window) bsxfun(@times, feat_map, cos_window), xt_proj

% Compute the fourier series
xtf_proj = cellfun(@cfft2, xt_proj, 'uniformoutput', false);

% Interpolate features to the continuous domain
xtf_proj = interpolate_dft(xtf_proj, interp1_fs, interp2_fs);
```

Then, by convolution T-periodic multi-channel convolution filter to predict the detection score, as follows:

```
% Compute convolution for each feature block in the Fourier domain
% and the sum over all blocks.
scores_fs_feat{k1} = sum(bsxfun(@times, hf_full{k1}, xtf_proj{k1}),
scores_fs_sum = scores_fs_feat{k1};
for k = block_inds
    scores_fs_feat{k} = sum(bsxfun(@times, hf_full{k}, xtf_proj{k}),
    scores_fs_sum(1+pad_sz{k}(1):end-pad_sz{k}(1), 1+pad_sz{k}(2):en
        scores_fs_sum(1+pad_sz{k}(1):end-pad_sz{k}(1), 1+pad_sz{k}(2
        scores_fs_feat{k};
end
```

The training of filters is by

$$E(f) = \sum_{j=1}^{M} \alpha_j \left\| S_f\{x_j\} - y_j \right\|_{L^2}^2 + \sum_{d=1}^{D} \left\| w f^d \right\|_{L^2}^2 . \quad (3)$$

and use Parseval's formula to transform to realm:

$$E(f) = \sum_{j=1}^{M} \alpha_j \left\| \widehat{S_f\{x_j\}} - \hat{y}_j \right\|_{\ell^2}^2 + \sum_{d=1}^{D} \left\| \hat{w} * \hat{f}^d \right\|_{\ell^2}^2 . \quad (5)$$

In ECO algorithm we use three techniques to optimize C-COT algorithm, mentioning below:

A. Factorized Convolution Operator

In implementing, we set that filter f^d has contributed little, so we use Factorized Convolution Operator to choose more powerful filter and subtract the remaining calculation, as follows:

$$S_{Pf}\{x\} = Pf * J\{x\} = \sum_{c,d} p_{d,c} f^c * J_d\{x^d\} = f * P^{\mathrm{T}} J\{x\}. \quad (6)$$

P is a DxC matrix, every line represents a dimension feature corresponding filter, use all C filter liner combination coefficient, which is an unknown. We need to learn from the first frame, the later tracking remains the same, therefore, the function is changed to, as follows:

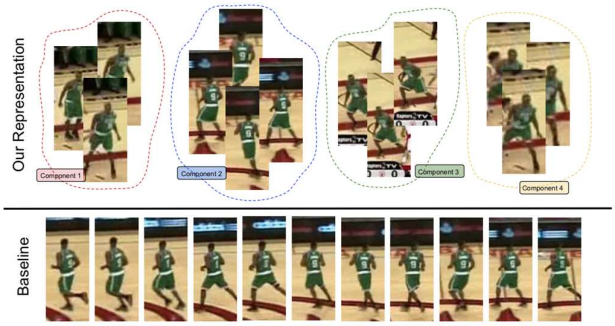
$$E(f,P) = \left\| \hat{z}^{\mathrm{T}} P \hat{f} - \hat{y} \right\|_{\ell^2}^2 + \sum_{c=1}^{C} \left\| \hat{w} * \hat{f}^c \right\|_{\ell^2}^2 + \lambda \|P\|_F^2. \quad (7)$$

Inside, z = J{x}. then add the last $\lambda\|P\|_F^2$, with the Frobenius norm of P to constrain P. The λ is to control weight parameter.

The (7) formula is a minimum square problem. To solve the two-linear problem, the author transformed the problem into another matrix problem, using Gauss-Newton and Conjugate Gradient to get the solution. Finally, get the projected feature maps $P^{\mathrm{T}} J\{x_j\}$, successfully solve the Model Size problem.

B. Generative Sample Space Model

Usually, the tracking most DFS algorithm use, the training data sample is too complex and large, causing trouble with computation and the burden of tracking speed. Therefore, ECO want to simplify the training set.



As the figure above, Below is the traditional training set (C-COT) When a new training component is added, because of the continuous sample has a highly similarity, easily causing the over-fitting problem. Above is how ECO works, using Gaussian Mixture Model (GMM) to generate different training component. Every component correspond to a similar sample, different component has more significant difference. So that the training set is more variety.

Our approach is joint probability distribution p(x, y) of sample feature maps x and corresponding output scores y. we use p(x, y) to minimize the expected error with the following formula

$$E(f) = \mathbb{E}\left\{ \|S_f\{x\} - y\|_{L^2}^2 \right\} + \sum_{d=1}^{D} \left\| w f^d \right\|_{L^2}^2 . \quad (10)$$

The expectation E is evaluated over the joint sample distribution p(x, y) . To estimate p(x), we employ a Gaussian Mixture Model (GMM) such that p(x) = $\sum_{l=1}^{L} \pi_l N(x; \mu_l; I)$. Here, L is the number of Gaussian components N (x; μl ; I), πl is the prior weight of component l, and μl ∈ X is its mean. To update GMM, we use Declercq and Piater, we first initialize a new component m with weight and mean, if the number of components exceeds limit L, We discard a component if its weight πl is below a threshold. Otherwise, we merge the two closest components k and l into a common component n, as follows

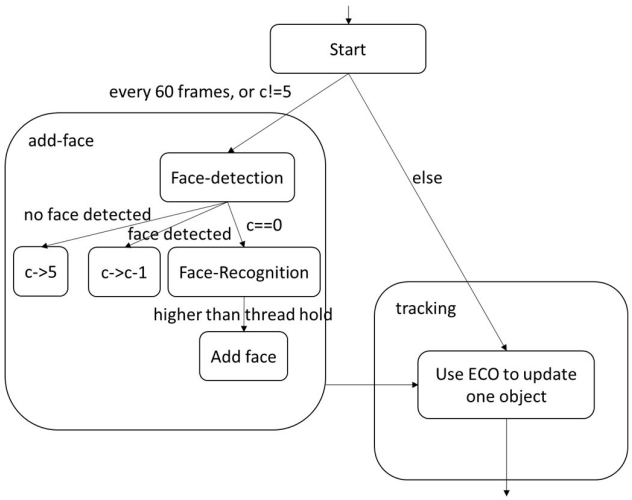
$$\pi_n = \pi_k + \pi_l \quad , \quad \mu_n = \frac{\pi_k \mu_k + \pi_l \mu_l}{\pi_k + \pi_l}. \quad (11)$$

The key difference in complexity compared to (3) is that the number of samples has decreased from M to L, our sample distribution model p(x, y) is combined with the factorized convolution from section 3.1 by replacing the sample x with the projected sample P^{T} J x , which does not affect our formulation since the matrix P is constant after the first frame.

C. Model Update Strategy

In the standard DCF tracking algorithm, the frequency to update model is every frame per time, which is very frequent. So that it has over computational problem. Therefore, ECO algorithm doesn't update the model continuously to reduce the update frequency. Its method is to update per frame, which is setting a variable Ns to make the frame become discontinuous. With Ns to fix the N_{CG} (Conjugate Gradient iterations) to refine the model. In other words, the average number of CG iterations per frame is reduced to N_{CG}/Ns. However, there is one thing to be concerned, Ns only affect to model updating, Ns won't affect the updating of the sample space model. To this end, the reduction of frequency of model update and to avoid the drift of the model has an optimized result. We can't make Ns too large, or the model can't vary with the object.

As for the implementation part, mainly cutting the whole process in 2 parts, add-face mode and tracking part.



In every 60 frames or so the program will run the add-face part, confirm whether there is new face or not. (by checking if the size of the box is normal) If yes, the next frame will still be running the add-face part, as to check if the face is really a face or not. After 5 frames of confirming and cropping the face out, use PCA-FLDA to see if it is recognized or not. If not then ask the user to input the new face's name. After all that, add that face into the tracking list. (variable objh in the code)

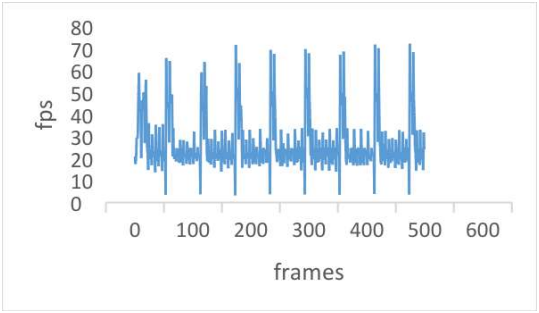
When it comes to tracking part, it will then run every faces in the tracking list, and then update their new positions. Or you can decide to update only one face if you want to keep running the program in a high fps but track a lot of people at the same time.

Now for the core of the tracking part, because eco algorithm is mainly design to track one object, so it is very important to do a fast transfer of the parameters and models between different objects. Keeping every parameters of the previous stage in a global memory space, and only sent the index of the object to the function when needed is a good way to imply. As for the other part, mainly redesign the ECO program from progress 2 into 3-parts: initialize, find best next position, and visualizing.

There are two conditions when we delete the faces from the tracking list. One is when the object is "time out". In normal case, human's face will somehow watch the camera, so if a object that is being tracking is really a face, then longer or shorter will the face detection part scan the face and reset the timer. Another condition is when the object is out of the screen (or very near the edge), as when the object is out of the screen, it is not a valid target to track anymore.

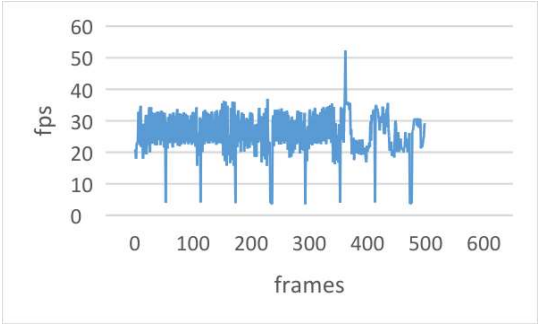
IV. EXPERIMENT

This is the fps of one of the tries we made. You can see that in every 60 frames there will be a low fps, that's because it is doing the face-detection (Viola-Jones) to scan the 1280x720 snap shot. The computer I am using is kind of slow, as it is using i7-4710 but without an additional display card. Using only the power of single core this particular test is running the program to track only one face, and the average fps is 27.013 in 500 frames.

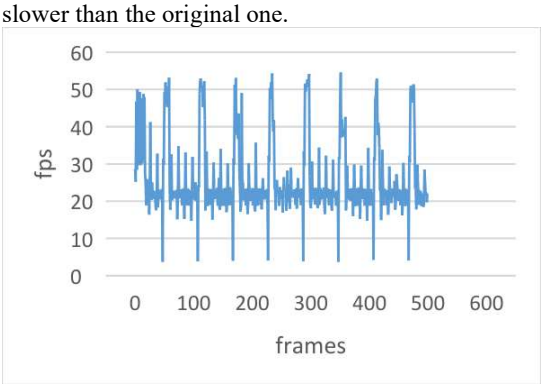


We tried to optimize the program by only updating one object at a time, but it seems that it is not that important when it is tracking two people.

This is the original version with updating every faces in every frames. The average fps is 26.635, not bad at all. It seems that matlab have done some serious optimization to the function "for".

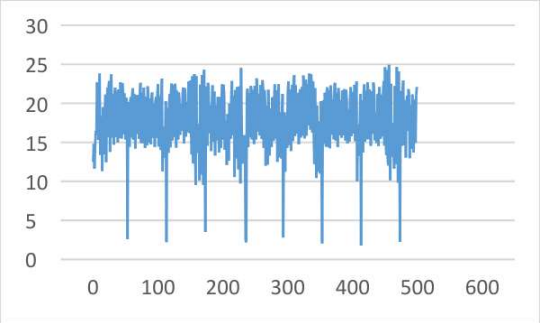


As you can see this is the optimized version which updates only one face at a time. The average fps is 26.2195. This is even slower than the original one.



But when it comes to three or more people, the original version will drop to about fps 18. The update one at a time version still keep above fps 25 as the slow down is due to the drawing part.

This is the original version, the average fps is 18.15. The drop of the fps is kind of obvious, but the optimized version have nearly no differences with the previous graph, only with a average fps of 25.53.



As you can see, the tracking part do work well after the recognition part.

If using only the naïve implemating method, the picture on the right will not run successfully.

Two or more people to track at the same time have no problem at all, as this is one of the exemple with one of my roomate. (He is in group 12, Mr.Hu)

V. RESULTS AND CONCLUSION

With an average fps higher than 20, and in every 60 frames due to the detection and recognition part, fps will drop to 1~2, that's a very reasonable result. Reminding that the tests mention above is running under the environment of single core and without display card. If using the "parfor" in matlab, then maybe the "update one face at a time" strategy won't be needed. (Somehow my computer keep popping error when trying to use the multi processing for loop in matlab) And if using the display card to accelerate the face detection and recognition part, that will let the drop at every 60 frames be a lot more unnoticeable. And maybe using other face detection algorithm other the Viola-Jones will be better, and more accurate. And for the recognition part, maybe using the filters created during the ECO tracking will be as accurate as PCA-FLDA, but with a faster speed. (the filters is created by Conv-1, Conv-5, HOG and CN)

In conclusion, we combined face detection (Viola-Jones), face recognition (PCA-FLDA) and video object tracking (ECOHC), and the result is kind of neat--fast and accurate. The only problem is to understand in what circumstance will a new face be define as unrecognized with the data in the data base. Adjusting thread hold is a way to get the job done, but this process will be needed when using a different camera or when the light or angle is different.