Computer Vision Homework # 3: KCF Tracker

Jiazheng Li[1]
[1] Beijing Institute of Technology

Abstract

This report addresses the challenge of tracking detection with KCF. In this study, I have employed KCF to address this issue and compare with KCF from Opencv. The code and dataset are available at https://github.com/foreverlasting1202/CV-hw-3.

1. Introduction

Visual tracking has been a fundamental problem in the field of computer vision with recognized research significance and wide application value over the past decades. Challenges in this area include dealing with rapid motion of the target, scale changes, occlusions, and lighting variations, among others. In this context, the introduction of the Kernelized Correlation Filters (KCF) algorithm has provided a new solution for high-speed and accurate object tracking. The KCF algorithm leverages circulant matrices and the discrete Fourier transform to efficiently train classifiers, enabling fast and precise object tracking even on datasets containing thousands of samples. The efficiency of the KCF algorithm is attributed to its simplicity in design and mathematical elegance, significantly reducing computational load while maintaining algorithmic performance [1].

2. Related work

A significant advancement in visual tracking research prior to the KCF algorithm was the widespread adoption of discriminative learning methods. These methods often involve an online learning problem where the goal is to identify the target from an initial image patch using a classifier, which is then used to detect the target in subsequent frames [1]. However, modern trackers walk a tightrope between incorporating as many samples as possible and maintaining low computational demands, often randomly selecting a few samples per frame for updates [1]. Henriques et al. addressed this issue in the KCF algorithm by proposing the use of circulant matrix analytical tools, which simplifies the learning algorithms' complexity while adding samples, by exploiting properties of the Fourier domain [1].

Moreover, the development of trackers also included the adaptation of various machine learning algorithms for online learning, such as Support Vector Machines (SVM), Random Forest classifiers, and boosting methods [1]. For instance, Zhang et al. proposed a Naive Bayes classifier based on compressive sensing techniques, while Hare et al. employed a Structured Output SVM with Gaussian kernels, using a plethora of image features to predict the target's location [1]. These methodologies have laid the theoretical and technical groundwork for the emergence and optimization of the KCF algorithm, which still holds potential for further improvement and exploration [2].

3. Method

KCF, which stands for Kernel Correlation Filter, is a tracking algorithm that primarily employs discriminative tracking. It utilizes the initially provided samples to train a discriminative classifier using techniques such as circulant matrices and kernel methods. Fast Fourier Transform is then used to accelerate this computation process. Subsequently, the classifier is updated as the object moves.

3.1. Ridge Regression

Ridge regression is a regression method used for fitting a linear function $f(\mathbf{z}) = \mathbf{w}^T\mathbf{z}$. Its loss function is defined as

$$\min_{\mathbf{w}} \sum_i (f(\mathbf{x}_i - y_i))^2 + \lambda \|\mathbf{w}\|^2.$$

It's worth noting that this equation has a clear analytical solution, which is

$$\mathbf{w} = (X^H X + \lambda I)^{-1} X^H \mathbf{y}.$$

Here, $\lambda \geq 0$ is a hyperparameter.

3.2. Train

For a vector x, we aim to train a classifier for it. Due to the background influence, we consider using the circulant matrix $X = C(x)$ to train, which can to some extent mitigate the background interference. In this case, considering ridge regression, we can obtain $\mathbf{w} = (X^H X + \lambda I)^{-1} X^H \mathbf{y}$.

Circulant matrices possess excellent Fourier matrix factorization properties, i.e., $X = F\text{diag}(\hat{\mathbf{x}})F^H$. Substituting w into this and taking the FFT simultaneously, we get

$$\hat{\mathbf{w}} = \frac{\hat{\mathbf{x}}^* \odot \hat{\mathbf{y}}}{\hat{\mathbf{x}}^* \odot \hat{\mathbf{x}} + \lambda}.$$

The method for sampling x typically involves applying a Hanning window for weighting, followed by feature description using the HOG histogram. The training label matrix y is usually generated using a two-dimensional Gaussian function.

In real-world problems, the relationship is not typically linear, so kernel methods are used to map the function into a high-dimensional space to achieve linearity. Let $\mathbf{w} = \sum_i \alpha_i \varphi(\mathbf{x}_i)$, and by solving the equation, we can obtain:

$$\hat{\alpha} = \frac{\hat{\mathbf{y}}}{\hat{\mathbf{k}}^{xx} + \lambda}.$$

Here, $\mathbf{k}_i^{xx'} = \varphi^T(\mathbf{x}')\varphi(P^{i-1}\mathbf{x})$, where P is a circulant permutation matrix.

Further mathematical derivations lead to:

$$\mathbf{k}^{xx'} = \exp\left(-\frac{1}{\sigma^2}(\|x\|^2 + \|x'\|^2 - 2\mathcal{F}^{-1}(\sum_c \hat{\mathbf{x}}_c^* \odot \hat{\mathbf{x}'}_c))\right)$$

Here, $\mathcal{F}$ denotes the inverse Fourier transform.

3.3. Update

During the tracking update, we utilize the information obtained from training and examine the maximum value of the response matrix $\hat{f}(\mathbf{z}) = \hat{\mathbf{x}}^{xz} \odot \hat{\alpha}$ in the Fourier domain. We consider this position to be the displacement for the next detection window.

Additionally, based on a certain learning rate (lr), we update parameters like α and the samples x. Depending on the current situation, we also perform some local training again.

4. Implementation details

I have implemented the KCF algorithm using NumPy and conducted experimental comparisons with OpenCV's KCF algorithm and perform codes on macOS Ventura 13.3.1, M1 pro, 16GB with PyCharm.

5. Experiments

5.1. Datasets, metrics

Datasets. We use a video of a fist moving up and down as the dataset.

Metrics. For simplicity, we used visual measurements.

Figure 1. **A frame of the dataset**

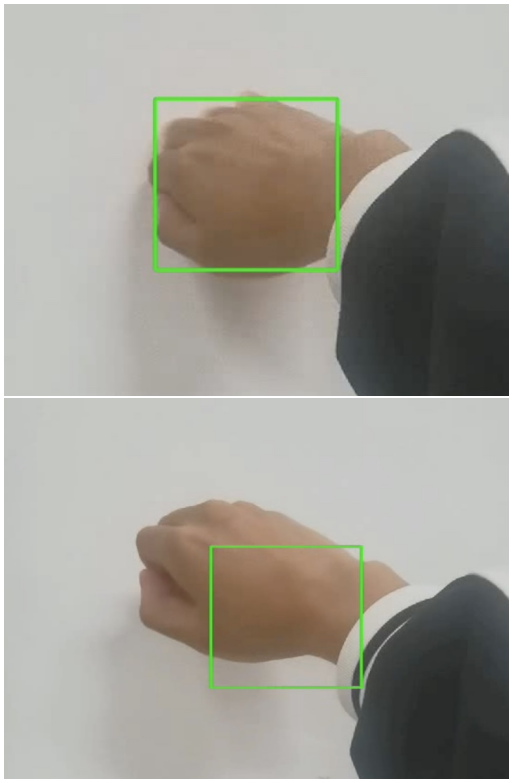

Figure 2. **Two frames of KFC by myself**

5.2. Experiment Result

Comparing Figure 2 and Figure 3, it can be observed that the performance of the KCF tracker I implemented is not very good. In fact, the detection window does not fully follow the movement of the fist. However, the KCF tracker in OpenCV works quite well.

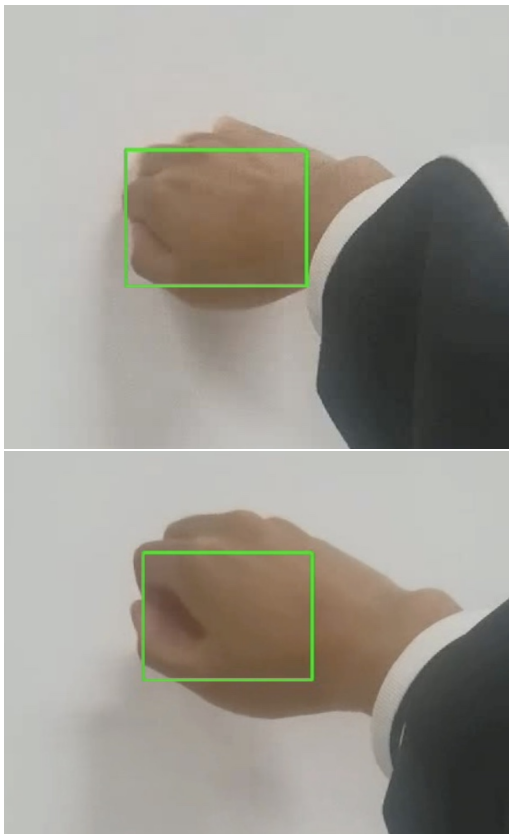

Figure 3. **Two frames of KFC by opencv**

6. Conclusion

In this paper, we have introduced the fundamental principles of the KCF algorithm and provided a manual implementation of the algorithm, comparing it to OpenCV's built-in implementation.

Limitations. However, during the comparison, I found that OpenCV's implementation performs very well. It can still track the object's movement even in scenarios with a cluttered background. In contrast, my handwritten version falls short of achieving the same level of performance. Further modifications to the code are required to eliminate some non-maximum points in the detection process.

References

[1] Joao F. Henriques, Rui Caseiro, Pedro Martins, and Jorge Batista. High-speed tracking with kernelized correlation filters. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37(3):583–596, mar 2015. 1

[2] Srishti Yadav and Shahram Payandeh. Critical overview of visual tracking with kernel correlation filter. *Technologies*, 9(4), 2021. 1