Image Classification on the Edge for Fast Multi-Camera Object Tracking

May 13, 2018

1 Abstract

This paper introduces a low-latency method of tracking an object as it moves throughout an area observed by a dense network of video surveillance cameras. This new method utilizes the computing power of edge device to run lightweight image classification closer to the source of the video data. The sensor redundancy in wide camera networks allows us to increase the accuracy of local lightweight image classifiers to provide for a real-time estimate of a target's location in the sensing region. Running image classifiers on the edge eliminates the need to offload all video data to the cloud and would improve the latency issues inherent to offloading solutions.

2 Introduction

In recent years, more and more data is being processed closer to its source. This so-called edge computing produces results more efficiently by utilizing the compute resources within data-collecting devices. The reductions in network communication costs from such a paradigm however are balanced by the need for distributed algorithms capable of running on devices with limited processing abilities[4].

In this paper, we investigate the problem of locating and tracking an object in a space observed by a network of video surveillance cameras. Some cities are monitored by large numbers of video cameras. For example, the city of London alone is estimated to have half a million CCTV cameras in daily use [5]. When terrorist attacks, shootings, abductions, and other crimes happen, police officer should be able to use live video surveillance data for real-time object detection to identify and catch suspects more quickly.

Real-time image classification and object detection are computationally expensive tasks involving multi-layered convolutional neural networks (CNNs). Identifying objects within images is performed in multiple stages. Coarse-grained image classifiers identify video frames containing objects that fall into general classes, i.e. a person, dog, or backpack. Once these are identified, more detailed classifiers are used to refine the query further. Prior methods of approaching object detection and tracking involve offloading video from the surveillance cameras to a central cloud server with greater computing resources [6]. When the amount of video data is large, it may exceed the network's bandwidth capacity and the cloud's compute resources, leading to backlog and increased latency. Other works have leveraged the edge devices' compute resources for image classification. When the backlog in the cloud is too great, instead of offloading a frame, the device will run the expensive image classification locally [6].

In [6], offloading video data is always the preferred action for an edge device; however, in this paper we present a distributed solution for tracking objects as they move throughout an observable sensing region. Instead of offloading to the cloud, video cameras may run the coarser image classifiers locally. Since camera placement is very dense, many cameras view the same regions of the sensing area. We utilize this redundancy and show that by using a Kalman filter to extract useful information from noisy sensor data, we can identify frames that contain a query with great

speed and accuracy.

3 Image Classification

State-of-the-art image classifiers are pre-trained CNNs. Classifiers take images as their input and output a softmax vector corresponding to their confidence that an image contains an object corresponding to each word in its vocabulary. More lightweight image classifiers will have smaller vocabularies. For example, AlexNet can classify the objects in an image into 1000 different classes [3].

Image classifiers pass their output through a softmax layer to give a normalized probability that each of its known classes can be found within the image. AlexNet, a lightweight image classifier that can run on mobile devices with limited computational abilities achieves a top-5 error rate of 37.5%. Moreover, image classifiers are sensitive to the orientation of an object. The same object viewed from different angles by different cameras may not be classified correctly by both cameras' instances of AlexNet.

4 Model and Problem Setup

4.1 Model

We model the target area as an un-directed lattice graph G on $n \times m$ nodes. We let the lattice wrap around on itself to reduce the complexity of handling cases that arise at the edges of the lattice. Furthermore, we define a target as any object on the lattice that matches the same classes in AlexNet as the query would. There can be multiple targets on the grid simultaneously. Each target moves according to a lazy-random walk on G from a random initial position. At any time step the target will stay at its current position with probability p or move to a neighboring node uniformly at random. The probability p can be set to zero to simulate a rapidly-moving object or one for a stationary target.

At every node, we place a camera randomly with probability q. We let l denote the number of cameras. Each camera has a viewing radius of r and can see r nodes in all directions. A sample graph with cameras and a target can be seen in Figure 1. Let C_i be the number of cameras that observe node i. It is clear that

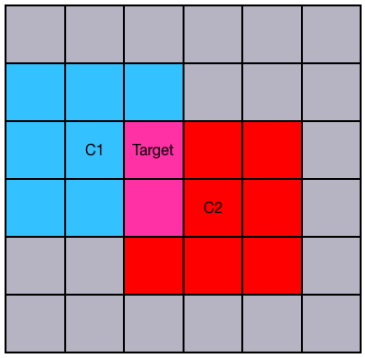

Figure 1: A 6×6 grid with two cameras of radius 1. The target is observed by both cameras at this time step.

$$\mathbb{E}(C_i) = (2r+1)^2 q \tag{1}$$

Since the lattice corresponds to a large search space, and the lightweight image classifiers have more general classes, it is likely that there are multiple objects that match the query's classes according to the image classifier. We model this lack of uniqueness by running multiple simultaneous random walks across the lattice. Furthermore, not all of these objects match the target exactly. For example, the classifier's confidence level for a blue car when observing a green car may be close to 1. Thus, each target j has a corresponding weight $w_j \in [0, 1]$. This weight w_j represents the confidence value that an ideal image classifier would return saying that object j matches the true target. The ultimate goal of the system should be to determine the location on the graph of the object with the greatest weight.

We define a vector $\mathbf{z}_t \in \{0, 1\}^l$. If $\mathbf{z}_t(i) = w_j$ then the i^{th} camera observes the j^{th} target with confidence w_j at time t. We model the inaccuracies of the lightweight image classifier by adding some 0-mean Gaussian noise to $\mathbf{z}_t$. To obtain each element of $\hat{\mathbf{z}}_t$ in reality would require each camera to feed forward a video frame through an image classifier.

$$\hat{\mathbf{z}}_t = \mathbf{z}_t + \mathbf{n} \tag{2}$$

This noise affects the confidence level at which the image classifier reports it sees the query. We argue that it sufficiently models cases in which the random orientation of the object at time t can impact the image classifier's accuracy.

The noiseless observable vector $\mathbf{z}$ is related to $\mathbf{x}$ by Equation 3. Define $\mathbf{H} \in \{0,1\}^{l \times nm}$ such that $\mathbf{H}_{ij} = 1$ if camera i observes node j.

$$\mathbf{z} = \mathbf{H}\mathbf{x} \tag{3}$$

Moreover, we let $\mathbf{A}$ denote the transition matrix of the random walk on the lattice and $\mathbf{x}^t$ denote the position of the targets at time t. According to the random walk we have

$$\mathbb{E}(\mathbf{x}_{t+1}) = \mathbf{A}\mathbf{x}_t \tag{4}$$

From Equation 4, we have a linear difference equation describing the dynamics of our system whose state we can observe via our noisy estimate $\hat{\mathbf{z}}$. We can therefore use a Kalman filter to derive an optimal estimator for $\mathbf{x}$ according to the linear system defined in Equation 4. We use the equations for calculating the optimal state and covariance estimators given in [2].

4.2 Algorithm

We initialize the state estimate $\mathbf{x}$ to be uniform for every node in G. Then at every time step, each camera runs its image classifier and reports its confidence value for the query to a central controller. These confidence values are compiled into the noisy vector $\hat{\mathbf{z}}$. This central controller is necessary to compute the state estimate from the Kalman filter equations. Even though this incurs some communication costs, each camera is offloading a single floating point value instead of an entire image. The Kalman filter is an online estimator so the algorithm continues to run forever, with each camera running its cheap image classifier one frame at a time, reporting its value to the central controller, and the controller updating its estimate of the state.

We recognize that every camera does not need to run the image classifier at every time step. From $\mathbf{x}$ we have an estimate for all the possible locations of the targets. Since the system evolves according to a random walk, we know that each target has only five possible moves.

$$\mathbb{E}(\mathbf{z}^{t+1}) = \mathbf{H}\mathbf{A}\mathbf{x}^t \tag{5}$$

From Equation 5, we can estimate what we expect the image classifiers will return before they are run. In simulations, it is shown that the Kalman filter algorithm returns accurate estimations for the state when it uses information from only the cameras with the top-k highest elements in the expected value of $\mathbf{z}^{t+1}$ given by Equation 5.

Algorithm 1 State Estimation Algorithm

$t \leftarrow 1$
$\hat{\mathbf{x}}_0 = \pi$
$\hat{P}_0 = I$
$Q = I$
$R = \sigma I$
while 1 **do**
  $\hat{\mathbf{x}}_t^- \leftarrow A\hat{\mathbf{x}}_{t-1}$ {State Prior}
  $\hat{P}_t^- \leftarrow AP_{t-1}A^T + Q$ {Covariance Prior}
  $\hat{\mathbf{z}}_t^- \leftarrow H\hat{\mathbf{x}}_t^-$
  for each camera i **do**
    if $i \in$ top-k $\hat{\mathbf{z}}$ **then**
      $\hat{\mathbf{z}}_t(i) \leftarrow$ AlexNet Confidence Level
    else
      $\hat{\mathbf{z}}_t(i) \leftarrow 0$
    end if
  end for
  $K = \hat{P}_t^- H(R + H\hat{P}_t^- H^T)^{-1}$ {Kalman Gain}
  $\hat{\mathbf{x}}_t \leftarrow \hat{\mathbf{x}}_t^- + K\mathbf{z}_t$ {State Posterior}
  $\hat{P}_t \leftarrow (I - KH)\hat{P}_t^-$ {Covariance Posterior}
end while =0

By only running image classifiers on the top-k highest cameras, we free up many compute resources. This will reduce energy costs. Also, in the case where this computation is being run on offline video data, it allows inactive cameras to start running upcoming frames through their image classifiers. These preemptive runs could reduce latency in future passes of the

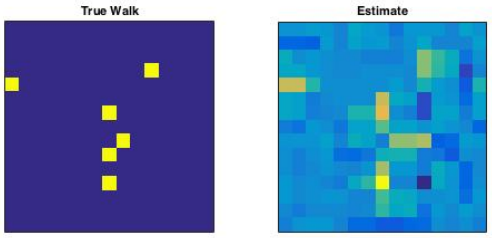

Figure 2: Heat maps of the true value of the state vector $\mathbf{x}$ and the Kalman Filter Estimate

algorithm. The algorithm is outline in Algorithm 7 [2].

5 Simulations

The algorithm above was simulated on a 15×15 grid. The first test demonstrates that the Kalman filter is able to identify 6 stationary targets on the larger grid when the weights are all set to one. For this experiment the standard deviation of the noise was set at .2, the probability of a camera existing at any given node was .3, and the radius of the camera was 1. The heat maps in Figure 2 illustrate this experiment.

The top-10 values of the estimate for the state contain contain the true location of all 6 of the targets, showing that the Kalman filter method is capable of detecting multiple stationary objects in the presence of noise.

Figures 3 and 4 respectively show the effects of the standard deviation of the noise and the camera density versus the effectiveness of the estimator. For these simulations, we have one target performing a random walk on the graph. We measure the success of the estimator by taking the average Euclidean distance on the lattice between the $argmax$ of the estimator and the true location of target. Each value for the success of the estimator is the average of 20 trials.

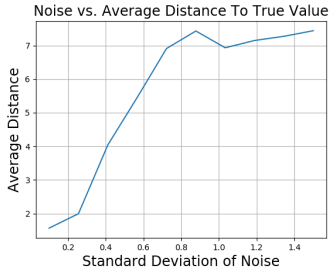

Figure 3: Effect of the amount of noise on the average distance from the estimate to the true value. $q = .3$

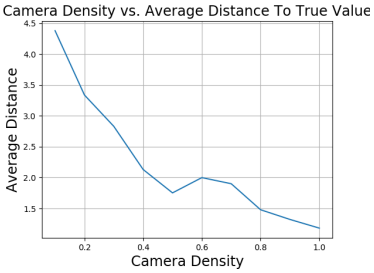

Figure 4: Effect of the density of cameras on the average distance from the estimate to the true value. $\sigma = .3$

In the worst case, when the standard deviation of the noise is too large, the estimate for the target's location will be totally random. Then, since the grid is 15×15, we can expect the average euclidean distance to be around 8. From our plot of the standard deviation vs. distance, the average distance increases linearly with the standard deviation. The average distance stays below half of the worst case distance until the standard deviation of the noise is .5. This corresponds to a 15% error rate in the top-k values of the image classifier.

In Figure 4, the standard deviation of the noise was set at .3. The average distance decreases as the number of cameras increase. The relationship is convex since there must be diminishing returns to adding more cameras.

We also measure the effect of how many image classifiers run at every time step. The algorithm calls for the top-k cameras given by the expected measurement vector at time t to run their image classifiers. A noise standard deviation $\sigma = .3$ and a camera density $q = .3$ were chosen. A graph with this k as the independent variable is shown in Figure 5. This plot reveals a threshold-like behavior. After k gets past a certain value, there is little gain in running image classifiers on more cameras. More work can be done in finding the optimal k at this threshold for any topology.

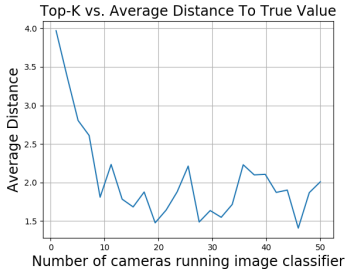

Figure 5: The effect of the number of image classifiers run on the average distance from the estimate to the true location. $\sigma = .3$, $q = .3$

In our final experiment, we simulate six targets moving across the grid. One target is given a weight of 1 and the rest have weights sampled uniformly from 0 to .5. As a metric for success, we find the largest element of the vector $H\hat{\mathbf{x}}$. This is the camera the Kalman filter deems most likely to be observing the target with the largest weight. Over all time steps, the indicator variables for whether this camera is actually detecting the target with the largest weight are summed up and normalized to give an accuracy percentage. In the unfiltered case, the camera with the highest noisy confidence level is selected as the one that observes the correct target.

Figure 6 shows that for small amounts of noise, the filtered and the unfiltered cases perform nearly identically well. But, as the amount of noise increases, the performance gap grows since the unfiltered case is rarely able to properly identify the target. The percent increase over the unfiltered case is shown in Figure 7.

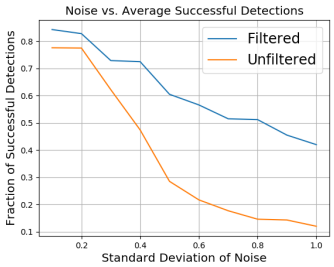

Figure 6: Effect standard deviation on detecting the true target among multiple candidates. $q = .3$

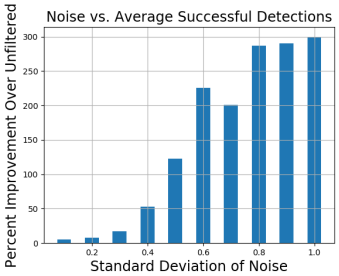

Figure 7: Percent improvement over unfiltered image classifiers. $q = .3$

6 Conclusions

In this paper we present a method of using lightweight image classifiers on the edge instead of offloading video data to more accurate classifiers in the cloud for the purpose of tracking an object as it moves throughout a sensing region. We demonstrate the required conditions for the noise levels in the image classifiers and the camera density for the Kalman filter to return a useful estimate of the object's location. We discuss the potential energy savings provided by a real-time estimate of the location of a query. In the future, it will be necessary to demonstrate the effectiveness of this method on real-world datasets. It is also necessary to compare this method to the existing offloading solutions to see if there are any im-

provements. This paper assumes the image classifiers will automatically run synchronously across all camera nodes. This assumption will have to be relaxed before any real edge solution is actually implemented. Lastly, this paper assumes that motion of the target in the space can be modeled by a Markov random walk. In order to handle cases in which motion may not be linear or Gaussian, we can use a particle filter. Particle filters are sequential Monte Carlo methods that generalize to the Kalman filter used in this paper and apply to non-linear state-spaces [1]. This extension may improve our algorithms performance when applied to real-world datasets.

References

[1] M. S. Arulampalam et al. "A tutorial on particle filters for online nonlinear/non-Gaussian Bayesian tracking". In: *IEEE Transactions on Signal Processing* 50.2 (Feb. 2002), pp. 174–188. ISSN: 1053-587X. DOI: 10.1109/78.978374.

[2] Simon Haykin. "Kalman Filters". In: *Kalman Filtering and Neural Networks*. Wiley-Blackwell, 2002. Chap. 1, pp. 1–21. ISBN: 9780471221548. DOI: 10.1002/0471221546.ch1. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/0471221546.ch1. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/0471221546.ch1.

[3] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems 25*. Ed. by F. Pereira et al. Curran Associates, Inc., 2012, pp. 1097–1105. URL: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

[4] Yuyi Mao et al. "Mobile Edge Computing: Survey and Research Outlook". In: *CoRR* abs/1701.01090 (2017). arXiv: 1701.01090. URL: http://arxiv.org/abs/1701.01090.

[5] Jonathan Ratcliff. *How many CCTV cameras are there in London?* 2018. URL: https://www.cctv.co.uk/how-many-cctv-cameras-are-there-in-london/ (visited on 01/02/2018).

[6] Kevin Chan Zongqing Lu and Thomas La Porta. "A Computing Platform for Video Crowdprocessing Using Deep Learning". In: *Proceedings of IEEE International Conference on Computer Communications* (Nov. 2017).