# Motion Vector Extrapolation (MOVEX) for Video Object Detection

Julian True *jtrue@ryerson.ca*

Dr. Naimul Khan *n77khan@ryerson.ca*

## Contributions

1. We replace expensive optical flow computation with pre-computed motion vectors from the H.264 encoder.
2. We propose a novel parallelism based approach to propagating pre-existing sparse keyframe detections.

Through these contributions, we achieve up to 25x latency reduction with minimal accuracy degradation.

## Background

Prior to our work, a common way to lower latency while maintaining accuracy was through the use sparse keyframe propagation, a diagram of which, is given in Figure 2a.

The drawbacks to this approach are:

- every *k* frames, an expensive object detection inference must be computed,
- frames are propagated using expensive optical flow computations,
- latency performance sensitive to object detection latency.

## MOVEX

We propagate detections through perturbing each detection in a set of prior detections with an aggregate of all motion vectors contained in a given detection, shown in Figure 1.
The motion vectors in our work are used directly from the H.264 encoded video, and since these are computed at encode time, come at zero computational cost.

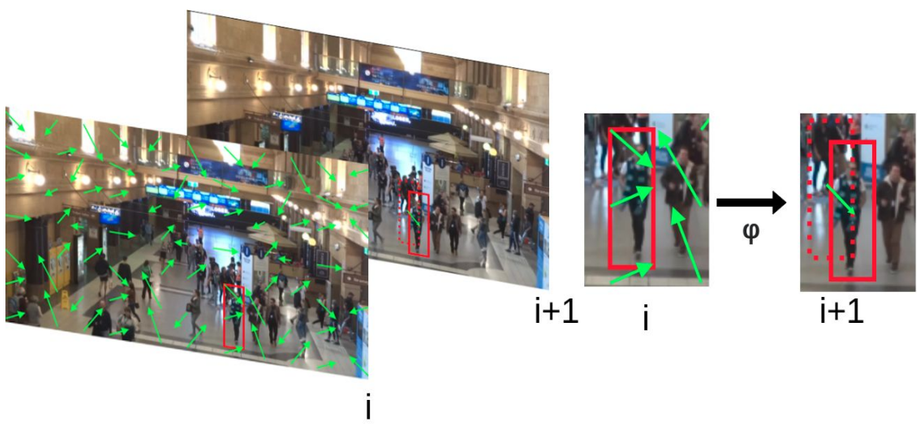


Figure 1: Sample frame propagation from frame i to frame i+1.

## MOVEX (cont..)

This prior set of detections comes from a second worker process shown in Figure 2b, whose sole job is to compute subsequent prior detection sets.
Since the propagation of detections is orders of magnitude faster than computing prior detection sets, worker 1 shown in Figure 2b works several frame ahead of worker2. In the highlighted prior update step, all previous motion vectors stored in the buffer are “replayed” on the detections from frame 2 in order to arrive at the current frame, frame 7.

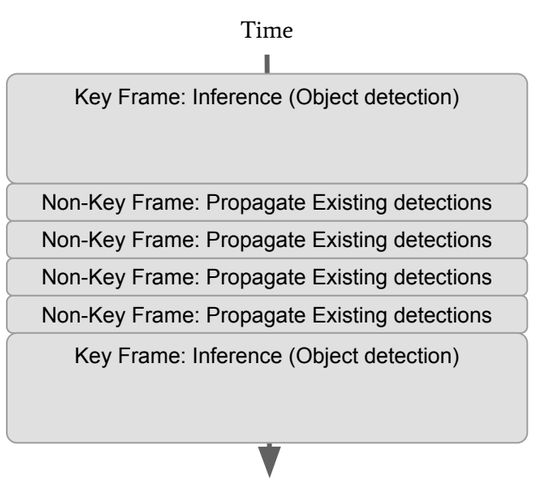


Figure 2a: Single-worker sparse detection propagation.

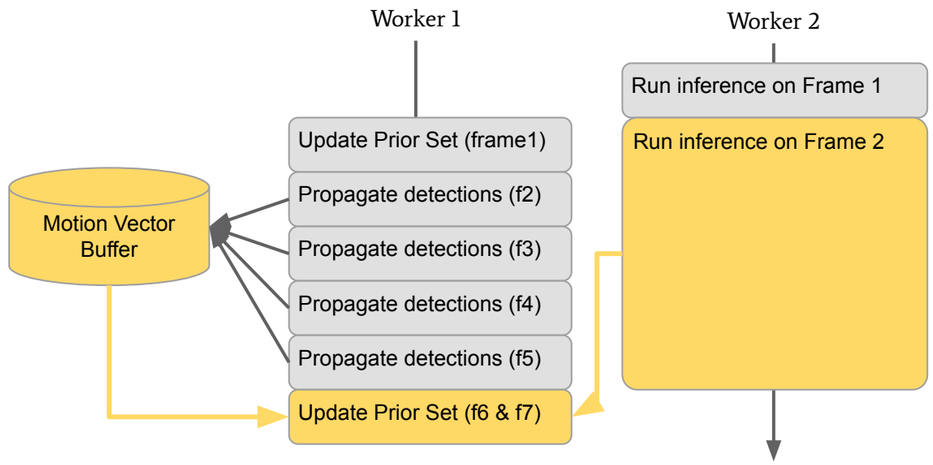


Figure 2b: Multi-worker sparse detection propagation.

| Method | Avg Latency (*ms*) ↓ | AP ↑ |
|---|---|---|
| FRCNN [3] | 131.52 | 63.0 |
| FRCNN [3] w/ MOVEX + FlowNet2 | 263.25 | 62.4 |
| FRCNN [3] w/ MOVEX + H.264 MVs | 12.79 | 62.3 |
| YOLOv4 [1] (960x960) | 79.46 | 40.2 |
| YOLOv4 [1] (960x960) w/ MOVEX + H.264 MVs | 11.06 | 39.8 |
| YOLOv4 [1] (416x416) | 26.34 | 26.1 |
| YOLOv4 [1] (416x416) on CPU | 190.41 | 26.1 |
| YOLOv4 [1] (416x416) on CPU w/ MOVEX + H.264 MVs | 7.53 | 25.2 |

Table 1: Results of evaluating on MOT20 dataset [3].

## Results

- 10x faster than the original Faster RCNN model (drop in AP of 0.6)
- High res YOLOv4 model with MOVEX outperforms low res YOLOv4 in both latency and accuracy
- Latency reduction of 25x in CPU application (drop in AP of 0.9)

Check it out on Github

https://github.com/juliantrue/movex