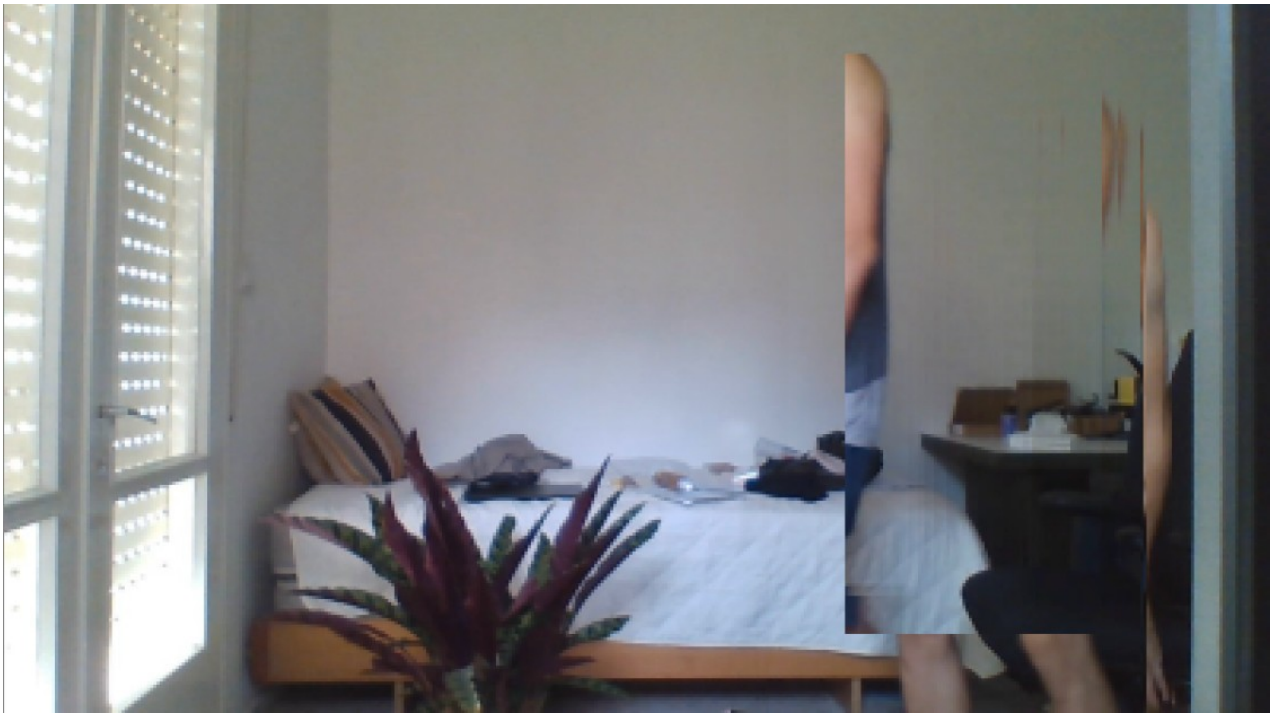
Project Description – Object sensitive video merge

Nadav Cohen, HUJI, Computer Vision Lab Project

Stage 1: Create a simple algorithm by using for each pixel its first color which was not detected as a person in order to get a shot closest to the shot which was taken while the video started running using the entire frames in the video.

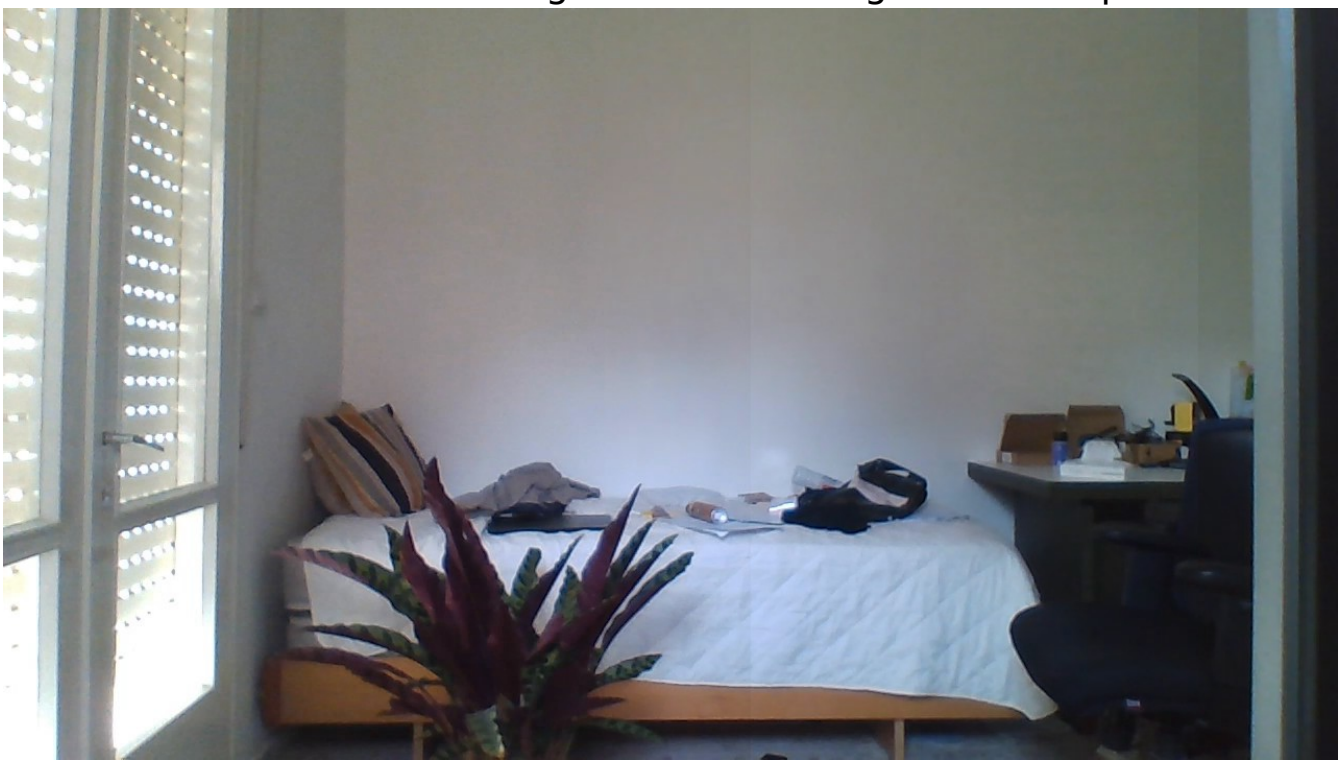
Problem: Using MobileNet object detector: not strong enough even for the simple videos, one missed detection and a portion of the person is in the combined image.

This image was created from a simple video of me crossing a room.



Solution: instead of MobileNet I tried to use
– Yolo3
– Enet semantic segmentation network (trained here:**https://www.pyimagesearch.com/2018/09/03/semantic-segmentation-with-opencv-and-deep-learning/** and described here: **https://arxiv.org/abs/1606.02147**)
The Enet segmentation were unreliable, and slower. YOLO3 worked great and dramatically better than MobileNet so the detector was switched for YOLO3.

The creation of the clean image after the change to YOLO3 perfect:



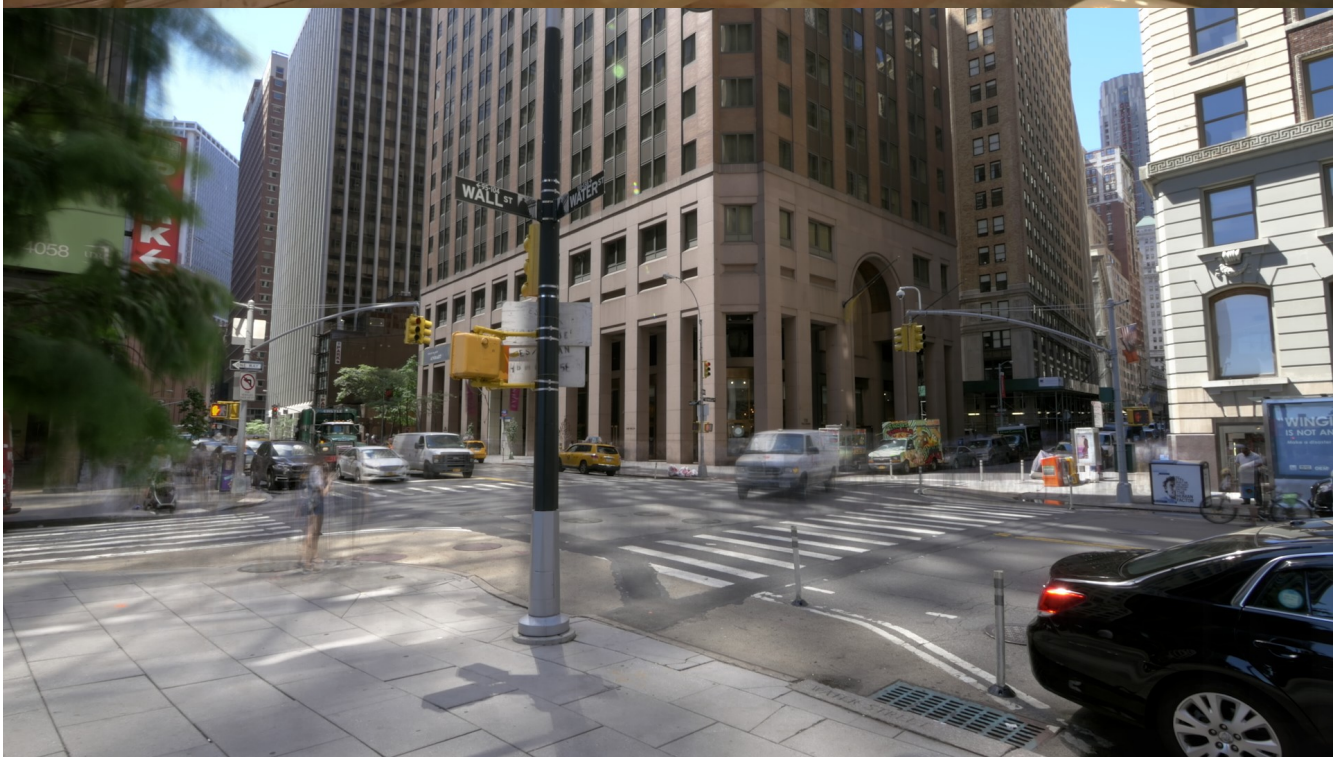
Problem: Taking the first color out of detection for each pixel proved to be unstable and generate mistakes due missed detections here and there. Processing the data while the video was running was also heavy on the CPU.
Solution: Process the entire video first and then choose the most common color out of detection. Later this was also changed due to problems. The idea of doing a live frame combination was dropped and the idea was changed to try and find the best ‘summarization’ of the video.

Problem: Using the entire video frames was slow and heavy and unnecessary since the movements in each frame are very small.
Solution: after some experiments started to sample a 1/8 of the frames.

Stage 2: After changing direction to processing all frames before running the reconstruction algorithm there was a need for utilizing the pixels data gathered in an effective way. A few solutions were tested until we concluded to a graph−labeling optimization algorithm.

1) The simple solution: for each pixel, averaging its colors over time (without people detection colors).
Problem: Result was unsatisfying. Generally, crowded videos has allot of noise, and miss-detection effected heavily with "ghosts". Also, places which had people standing for long had the people or their shades in the image. Maybe also have motion blur for pixels which belong to moving wanted objects (like plants, cars, etc...)

Average (with sensitivity to people detection)



2) Taking the most common color which was not detected.
Problem: Overall seemed like a better idea but suffered similar problems as averaging. The main problem was there were many colors and at this point they were not clustered.

3) Trying to pick the color, which was closest to the neighbor's decision, by iterating pixels in a spiral.
Problem: Total failure. Mostly enlarged mistakes made by the algorithms.

At this point it seemed that a good idea will be giving each color a 'cluster' meaning, for each color finding how much colors are close to it with Euclidean metric. After applying this and averaging the cluster colors we got a pretty good result (depending on threshold distance). The reason we did not use it was that the clusetring was taking in a 'lazy' way. Every color coming in was checked to be close enough to the current clusters, if it was it was assigned to a cluster and if not it opened a new one. The problem that bothered us then was that we should create a cluster for each color and then pick the right one. Now it seems to me this gave better results.

Taken with a color distance threshold of 65

Stage 3: After the previous results we tried to reduce the color count for each pixel and use a graph labeling algorithm.

Idea 1: Reduce colors by quantizing frames after detection.
Problem: Reducing the colors by using frame quantization (quantizing each frame) seemed not to give the expected results, no improvement at all.

Idea 2: quantize colors over time, meaning cluster all colors of a pixel gathered over time to 4 colors, world much better, but still had some color difference problems between neighbor pixels where there was high color difference.

Graph-Labeling – The graph labeling idea came after understanding that out Wanted objects are very noisy and has holes in them. The idea was to add info from neighbors colors in order to complete the holes found in objects.
The error function was described as:

$$\sum_{p \in pixels} [\max(all\ color\ cluster\ size) - chosen\ color\ cluster\ size] + ¿ \sum_{4-neighbors} d(p, n) ¿$$

after a few iterations it seemed that the best results came with alpha=0.99, meaning we want as much weight on neighbor color as possible while leaving some effect to the cluster size of the color in order to prevent blurry edges.
The results seemed to have a much grainier output, but objects where much more refined, which is possibly the reason the output seemed better than stage 2.


NEW HAVEN LINE DEPARTURES
TIME TRK DESTINATION REMARKS
4:34 19 STAMFORD MOUNT VERNON - 1ST STOP
4:37 24 NEW HAVEN STAMFORD - 1ST STOP
5:04 108 STAMFORD MOUNT VERNON - 1ST STOP
5:08 25 NEW HAVEN CONNECTION TO N.CANAAN, DANBURY & WTBY
5:34 110 STAMFORD MOUNT VERNON - 1ST STOP
NEW HAVEN LINE DEPARTURES
TIME TRK DESTINATION REMARKS
NEW HAVEN LINE DEPARTURES
TICKET VENDING MACHINES

Idea: In order to get better results for the algorithm a few general things were tested, like changing the number of frames used, with no luck.

IMPORTANT NOTE: It seems now that we may leave stage 3 behind and use the next stages (Patch optimization part) after stage 2, this may give better results and will shorten the algorithm dramatically.

Stage 4: Similar Patches\Frames use for refining wanted objects.
Initial Idea – After stage 3 we have an image which is clear enough to understand. The idea was to sort frames by their distance from the generated image and from there to run some sort of reconstruction algorithm which respects that order and uses it to find better patches. The pixels were graded for certainty (what is the max difference between two colors for the pixels, small difference means certain) and the comparison was made first by uncertain pixels and then by certain pixels.

Problem: The sorted frames seemed to be close but not precise enough, because we were looking on a big subset of pixels.

Second Idea – After that came an understanding that we should compare smaller patches. The best way to check if this can work was running another object detection and finding the closest patches for each object in order to perfect them in the image.

First Iteration – for each wanted object, similar frames were found and sorted by similarity. The generated patch consisted of their average. It was a success, the objects seemed whole and refined but had a little bit color, a leftover from patches with interference.

Second Iteration – Instead of taking all patches into to average, it was probably better to take a certain percent of the similar sorted frames into the average and do a weighted average by similarity. This was also very successful, and the objects seemed very good, maybe perfect.
The best percentage checked for similarity averaging seemed to be around 15%.

Previous result after refining wanted objects (cars).

Stage 5: Try the same method for the background patches. This was experimental but actually worked to better than expected. The

difference here is that we do not know what the patch size we should use.

First Iteration – Run experiment for patch sizes from 10 to 120 (squared). The results were surprisingly well, it seemed the patches generate to king of noise:
– Using small patches made small noise patches due to sensitive small color changes (every patch could pick a different color due to 'Salt Pepper' light difference noise.
– Using large patches had very small noise but did have 'ghosts' due to people passing in big frames.
A good patch size seemed to be 40 for both examples, but some uncertainty was felt due to it being sensitive to different king of noise which can vary for different videos.

Second Iteration – In an effort to generalize the patch algorithm, I tried to run on all patch sizes iteratively in two ways:
– Up – [10, 20, 40, 60, 90, 100, 120]
– Down – [120, 100, 90, 60, 40, 20, 10]

The output of 'up' was a failure and seemed to be no difference from the output of the 120 sized patch, which suffered from ghosts.
Down, seemed to work very well and produced better results than the 40 sized patch tried before. I think it worked well because the larger iteration first eliminated all the small noise areas, and later the small patches eliminating the ghosts while not creating noise like before since it was just eliminated by the big patches.

The images look much better but there a two problems:
1) The images have noticeable patch light differences. This differences are minor but still apparent.
2) The two videos get their best output from stopping on different path sizes:
Nyc street – best when applying [120, 100, 90, 60, 40, 20, 10]
Train station – [120, 100, 90, 60, 40] – going further created more noise

Remaining Challenges

1) fixing noise from people not moving enough.

2) find a solution for people not moving at all (inpainting or treating them like wanted objects)

3) handling colliding objects. (a group of colliding people will be easier to reconstruct from the same frame

4) Maybe – trying to find a more efficient way to decide which patch sizes should be used in each are of the image for stage 5 instead of running all patch sizes.

5) making algorithm more efficient.

Thoughts and Conclusions

1) Stage 3, which is the heaviest runtime part, maybe redundant. Images I found from stage 2 seems to have good results, maybe good enough to jump so stage 4, and maybe even better than what our Graph-Labeling algorithm outputs.

2) If (1) works, we may want to find a more efficient way to calculate the clusters, maybe combining it with some quantization.

3) For stage 4,5, currently the frames are used without discarding pixels with detections. Changing this may make results much better.

4) We may solve challenge (1) by refining detection by checking similarity where we find breaks in detections. Example: if we detection a person in frame #5 and then again only in frame #10 we can compare the patches, and if we see they are similar it may be safe to assume there should be a detection also in frames #6-9.

5) We may find a better way to pick patch sizes (optimization, ML) in order to get the best patch selection.

6) In order to solve problem (6) we may want to check the detections in stage 4 and if we see a detection contains a pixel which has no background frames, we may want to use one of the methods suggested. This will better work if we upgrade our detection method as suggested before in order to be able to truly detect unmoving objects in the image.

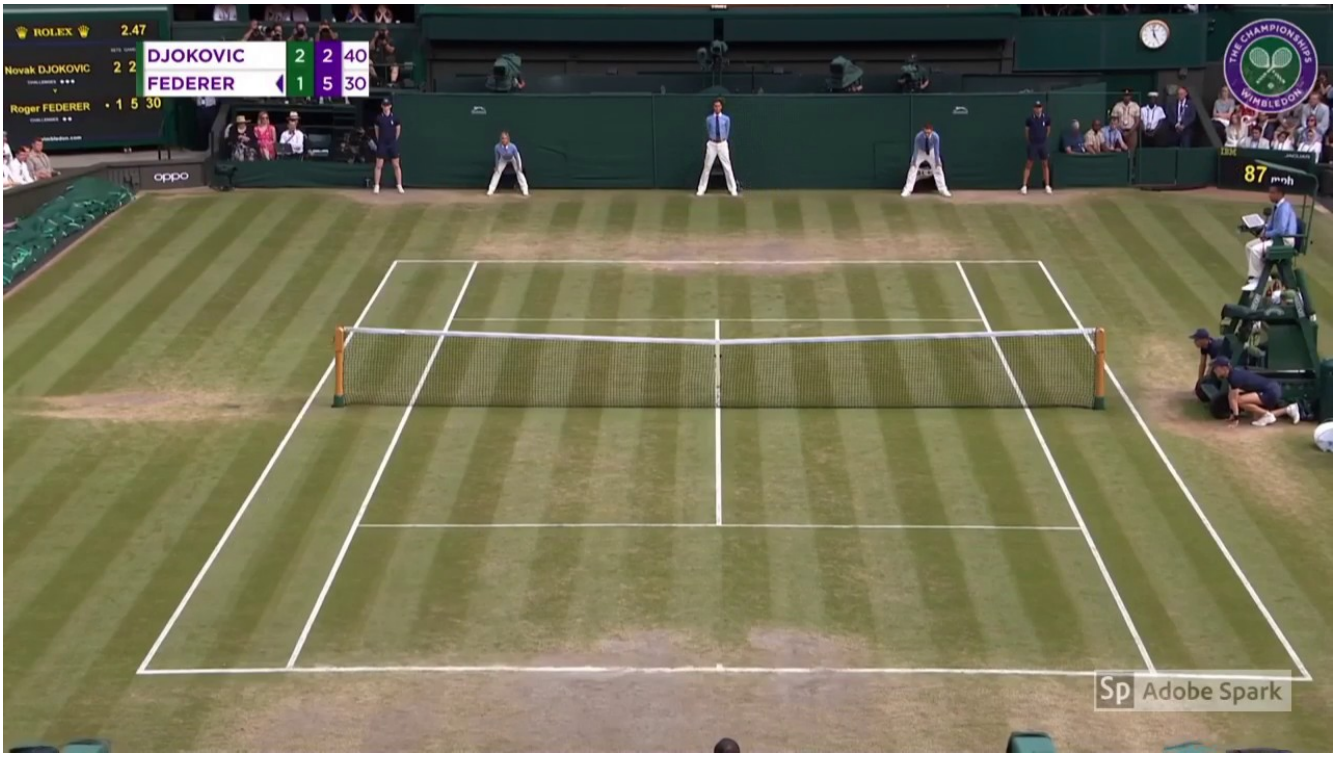
7) In order to optimize output and stage 5 we can apply stage 5 only for pixels which had at least one detection over the entire video. Pixels which did not have a detection may be taken from a single frame (or averaged) such that the frame of their patch matches the frame of the other pixels at the best way.

Update (30/11)

Stage 6: In an effort to try and simplify things we decided to use a median of the gray intensity values for each pixel, while removing colors detected on moving objects. To take care of people which still appear in the image we decided to reconstruct them using the same method we used for wanted objects before.
The results seem to be not bad at all although they are not perfect. I think that a bit less crowded videos would do pretty good.

It remains a question if this result is truly better that doing the whole optimization process. The optimization process may be a bit better overall but we still need to think of the rumtime and simplicity factors: the media way gives a results within less than 5 minutes where the optimization full process takes around 10 hours.


WALL ST
WATER ST


ROLEX
2.47
Novak DJOKOVIC
2 2
Roger FEDERER
1 5 30
DJOKOVIC 2 2 40
FEDERER 1 5 30
oppo
THE CHAMPIONSHIPS WIMBLEDON
87 mph
Adobe Spark

New Examples Results:

Success:

Basta in the Machne Yehuda Market

The Mashbir

Street under my balcony in Talbia neighborhood

Main Machnea Yehuda Market st.

Machne Yehuda “Etz Haim” st.

Failures:

This is a good output since it was very crowded but is is still a failure since the person in the image was regenerated in two places on the ambulance

The angle the video was taken had people all the time in the far end of the street. That part came out very noisy.