Video synopsis

Samuel Kostadinov
University of Trento
Trento, Italy
samuel.kostadinov@studenti.unitn.it

I. Introduction

Video synopsis is a method used in computer vision to summarize a video while keeping track of what happened in the field of view of the camera. This requires various different techniques, such as object recognition and motion tracking.
In this project, I implemented a video synopsis algorithm, provided some surveillance videos viewed from above of an indoor scenario.

II. External components

This project uses some components to work. In particular, in this section, there is a list of all the external components used in this project.
Python standard library modules:

- *os*
- *sys*
- *subprocess*
- *shutil*
- *sqlite3*
- class *Counter* from *collections*
- *sqrt* and *floor* from *math*
- *time*

External modules:

- *wget*
- *gitpython*
- *numpy*
- *tqdm*
- *python-magic*
- *Send2Trash*
- *opencv-python*
- *scikit-image*

Moreover, this project uses YOLOv5 for object identification, in particular the YOLOv5s6 set of weights.
All the required modules are listed in the file requirements.txt inside the “code” folder, so it is enough to launch `python -m pip install -r requirements.txt`.
Note that the script requires git and ffmpeg being accessible through the system PATH, since it relies on git to get the YOLOv5 repo and on ffmpeg to divide the video into frames and create the video output.

III. Usage

After cloning the repo from GitHub, it’s important to place the video on which the synopsis should be performed in the folder named “data”. It can be placed also in a subfolder. The requirements, listed on *requirements.txt*, should be installed with `python -m pip install -r requirements.txt` or a similar command, based on the system executable. The script gives the possibility to use the GPU for the object detection with YOLO. If the user decides to use the GPU, it’s also important to put the right command in the file called “command.txt” inside the “utils” folder. The script will read this file and execute the command that it’s contained inside it. To find the correct command for the CUDA support, it’s suggested to visit https://pytorch.org/get-started/locally/ and copy paste the command found there after selecting the setup present on the machine.
When launching the script, it will ask the user some questions:

- The script will ask if the user wants to use the GPU support for pytorch
- The script will ask if the user wants to delete the database result of the process. If the user decides to do it, the file should go through the entire pipeline another time for regenerating the output.
- The script will ask if the user wants to delete all the frames result of the middle phases of the execution of the script itself. Note that any frame, except for the background, will be deleted permanently at the beginning of a new run.
- The script will ask if the user wants to see on the console the output of the Object detection process with YOLO
- The script will ask to the user to input the name of the file to be examined. Please make sure that if the file is in a subfolder of the “data” folder the input is in the form `relative/path/to/video/`. The script also assumes that the video is in .mp4 format, so please input an mp4 file.

The script, in its usage, will also need *git* and *ffmpeg* available in the system path since makes use of both of them. The script also will need the permission to install python modules since it will try to install the requirements of YOLO and, if the user wants the CUDA support, the pytorch version that uses the GPU.

IV. Description of the script

The script can be divided into some parts:

- **Preliminary operations**:
  These operations are necessary to the script to work. First of all, it checks if the file provided as input is an mp4 video, then checks if it has already been processed by

checking if both the database and a background image are present. If both are present, the results are just used and the video is composed. If only one is present it is deleted and then the whole process is followed. After that, it checks if all the necessary files are present (such as YOLO and all the folders it will use). If they are not present, the script will download them and create the needed folders. As last step of this part, the script gets informations about the video, such as the fps and the size.

- **Process phase**:
  This phase consists mainly in three steps: dividing the video into single frames, rotate every frame by 90, 180 and 270 degrees, and tun YOLO on all the frames obtained after the rotation. This is the longest phase of the script's execution.
  The first step of this phase is splitting the video into frames. This operation is done using ffmpeg and saving them in the folder called "frames". The saved frames are called with the format *frame_number_0.png* where number is the progressive number of frames.
  The second step of the phase consists in rotating the frames. To do so, the script uses opencv to read the every frame and rotate it by 90, 180 and 270 degrees and then save the results using opencv library methods. The function that does this part also cuts and pastes the frames from the "frames" folder into the "rotated_frames". The frames are named in the format *frame_number_rotation.png* where number is the progressive number of the frame and rotation is the rotation applied, which can be 0, 90, 180 or 270.
  After rotating every frame, the script uses the subprocess module to analyze all the frames in the "rotated_frames" folder using YOLO. The subprocess calls a modified version of the detect script given in the YOLOv5 repo. This script maintains the same structure as the original one, the main difference is that the results are then written on a sqlite database using as primary key the number of the frame and the rotation.
- **Post process**:
  This phase consists in some steps that are listed and explained in the following paragraph.
  First of all the data are retrieved by the database using sqlite and the results are transformed into a list. After that, the script will filter out repeated boxes. To do so, if more than one result have the same index, the one with more boxes is kept.
  After that, there is a step in which the script converts all the coordinates of the bounding boxes, applying some transformation to every box to transform the coordinates of the bounding box and understand in which position the box would be in case the image was not rotated.
  After transforming all the coordinates, the script will delete the bounding boxes that are identified as outliers. A box is identified as outlier if that bounding box has a computed distance over a certain threshold. The distance is computed as the mean between the distance of the top left corner and the bottom right corner.
  After removing the outliers, the script filters again the results so that only interesting boxes are kept. To do so, the script scans the results and if the currently examined result has not the same number of detections of the previous result the script will scan forward and backwards for a certain number of frames. The two directions are treated separately, and if in one of the two segments of results the majority of the frames has the same number of boxes as the current result and in the segments there are no results with a number of boxes different from both the number of current detections and the previous detections, the script will discard the boxes that are identified as non-interesting. The interest in the boxes is defined based on the distance between each other. The most distant bounding boxes are the one discarded.
  As last step of the post processing, the script scans the results and detects where the script skips some frames. If the number of frames in which there is no detection is lower than a certain amount, the script interpolates the positions linearly between two boxes based on a distance criterion. Lastly, the script saves the results in the database.
- **Background search**:
  The background search is conceptually simple, since the script scans the results in which there are no detections after the post processing and, using scikit-image, computes the structural similarity index. Based on this index, the script finds the background as the image with the highest structural similarity with the previous image without any detection. The image is then renamed and copied in the folder called "utils".
- **Tag association**:
  The tag association mechanism can be divided into two parts.
  The first step assigns the tags in the first place. To do so, the scripts first assigns a tag to each of the boxes appearing in the first frame, then for every other results there are 2 possibilities. If the results is of a consecutive frame, the script evaluates which tags remain from the ones in the previous result and eventually adds some new tags if the new frame has more boxes detected then how many tags are kept. To decide which tag should remain, the script evaluates a distance criterion from the current position of each box and the last position in which any box is found. Moreover, if a tag is not present for more than a certain number of frames, that tag is considered "dead" and it's not considered any more in the following results. If the frames are non consecutive then, given that there was an interpolation, the scripts assume all people corresponding to the bounding boxes are assumed to be of new people and so the script assigns a new tag to every bounding box, marking all the tags of the previous frames as "dead".
  The second step is a tag refinement procedure, since in the tag association the results usually have some tags that

appear just a few times. To solve this problem, the script checks all tags that appear less then a certain number of times used as threshold and tries to inspect if the should be substituted. To do so, the script checks if the tag has to be deleted or substituted. For example, possible tags to be deleted are the one associated with reflections on the elevator, while some examples are just to substitute, given by the fact that some people may walk faster in some cases or YOLO can be not very precise in recognizing people sometimes and the bounding boxes can have a distance higher than the threshold given in the tag association. The script decides which tag to use scanning the results to find the tags used in that particular "segment" of the video. In this case, a segment is the part of video between the previous and the following time fraction in which there are no detections by YOLO. The tag that will be used to substitute is chosen based on a distance criterion.
- **Output video creation**:
The last step of the script is generating the video output. The output video is created by building the images. To build the images, the script will create a dictionary in which saves the bounding boxes to insert in the examined frame, which is identified with the index of the frame itself. An extra care is given to the overlapping bounding boxes, since if the starting position of two boxes overlap more than a certain amount only one of the two boxes will be reported in the dictionary. Using this dictionary, the script creates images overlapping the background found in one of the previous steps and the bounding boxes that are marked in the previously created dictionary. The script also makes sure that if two bounding boxes are overlapping an alpha effect is applied. The generated images are saved in a folder and then the script calls ffmpeg to concatenate them in a video format. Lastly, another subprocess based on ffmpeg changes the codec of the video to reduce its dimension.

V. Critical aspects and possible improvements

Although the algorithm works and the results are quite good, it's not perfect. For example, when there is more than one person on the field of view, YOLO may not be very precise if the people are near each other, so the algorithm, since it relies on YOLO's object detection is not that precise. One possible way to improve the results could be using a fine-tuned version of YOLO to find people or to fine tune it directly with the dataset on which it should be used onto. Unfortunately, there were not enough data to fine tune it properly so in this project the base version of YOLOv5 was used.

Another problem regards the video themselves. If the video is not fluid and sometimes gets blocked, the algorithm may not work properly. Depending on where the video is blocked and what position did the people have when the video was blocked. Unfortunately, this happens in most of the videos. In the video named "2018-12-03 13-00-00 13-04-54.avi" there is a person, around 1:30 in the video, going from the top of the view to the bottom that stops near the elevator. In this person's case, the video stops some times and one of these is in a pose in which YOLO does not recognize him. Since the person stops for less than a certain number of frames, the interpolation mechanism comes into play and interpolates the positions of the person, but that results in an image in which the interpolated bounding box contains nothing since in the original video the person is blocked. Otherwise, if the person stops for a number of frames higher then a certain threshold, the algorithm will treat that bounding box as the bounding box of a person that exited the field of view, and will associate another tag to the next time the person appears in another position in the field of view. Moreover, the script may have problems in case of splitting of bounding boxes. In Figure 1 there is an example of, respectively, a good and a bad frame. Thhe bad frame, which is the one on the right, has clearly two different detections that are the same person but identified two times since the script could not refine the tag association enough to understand that these two people are the same. On the contrary, the image on the left is a good example of how the algorithm should work, since all the detections are of different people.

It's also important to note that the script requires git and ffmpeg being installed and accessible through the system path. It also requires python installed to install the requirements of YOLOv5.

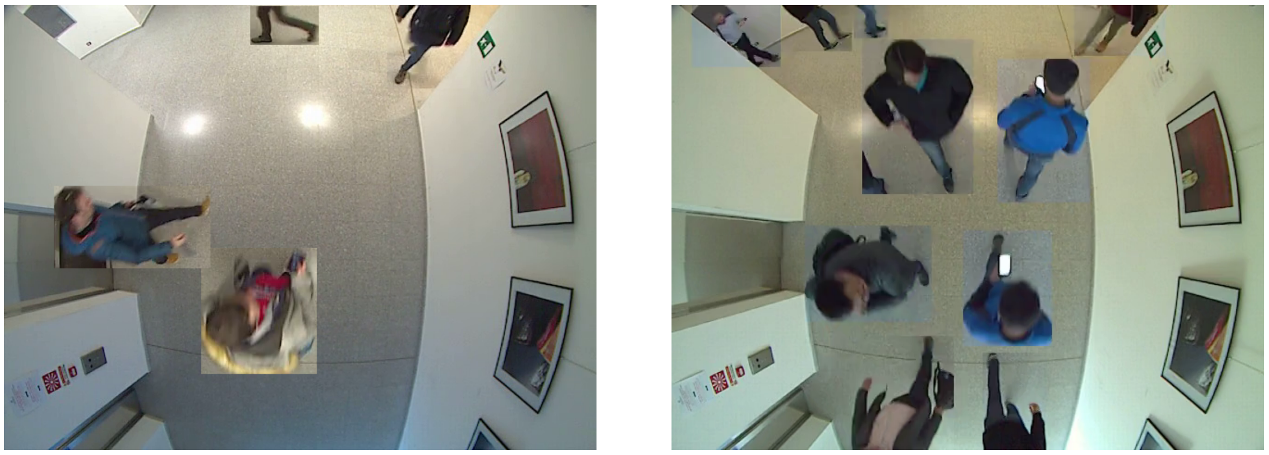
Fig. 1: Example of edge detection

References

[1] *YOLOv5 gitHub page*. Last accessed: 2022-08-16. URL: https://github.com/ultralytics/yolov5.