Video Descriptor Project

Royce Yang

Please note that my ZIP file is on Google Drive at this link: https://drive.google.com/file/d/10DNS-eAQQhTO9Oe87eRHvOZsaRmEeXau/view?usp=sharing, because uploading to Canvas was really slow. The only big files are the outputs from training, which I though was necessary to include in the ZIP.

I. Introduction & Originally Proposed Method

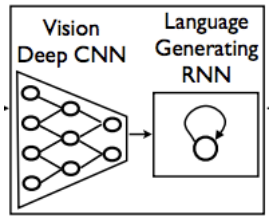
In general terms, the project that I propose is to be able to caption video and index the segments of that video related to the caption.

The flowchart is as follows:

1. For each frame in the video, use deep learning object detection to identify subjects present in the frame. We will use Darknet's YOLOv3 for being lightweight.
2. For each frame in the video, generate a list of potential captions. We will use a neural image caption generator consisting of a vision CNN and a language generating RNN.
3. Detect movement (will use kernelized correlation filter object tracking if time permits) from frame to frame and identify key objects. For a quick implementation, we will use coordinates of bounding-boxes from our object detector.
4. Choose best caption based on a voting procedure weighted by presence of key objects in captions.
5. Index and segment video based on similarity between final captions.

Specifically, I will use a pre-trained model, tiny YOLOv3 because I will be mainly running on my tiny and incapable laptop. Should I have better computing power, I would train my own model with labeled data from MS COCO and use an uncompressed version (not tiny).

In order to generate captions, I will use a neural image caption generator as proposed by Vinyals, Oriol, et al.:



Ensuing image processing with a CNN, this method uses LSTM-based sentence generation and therefore avoids the long-term dependency problem of general RNNs. Again, due to computational constraints, I will use a pre-trained model after training for 70 epochs (I know this is not nearly enough, but it will do for this project).

The latter steps of the flow chart are too trivial for this one-page-limit proposal.

For evaluation, I will manually index and segment a select few of the videos. I will then compare results with the manual labels using a confusion matrix that considers entry delays, exit delays, and jump discontinuities. In other words, my manual video segmentation will be used as ground-truth data for evaluation. The final evaluation result will be the percentage accuracy, or overlap, with ground-truth data. The captions will be evaluated on a percentage similarity basis when compared to ground-truth.

II. Method

In order to make it easier for you to follow my description of the method, here is my to-do list:

- Detect objects in each frame of the video
- Track objects from one frame to the next, identifying different instances and object classes separately
- Generate captions for each frame of the video
- Choose the best caption for each frame
- Segment video and choose the best caption for each video segment

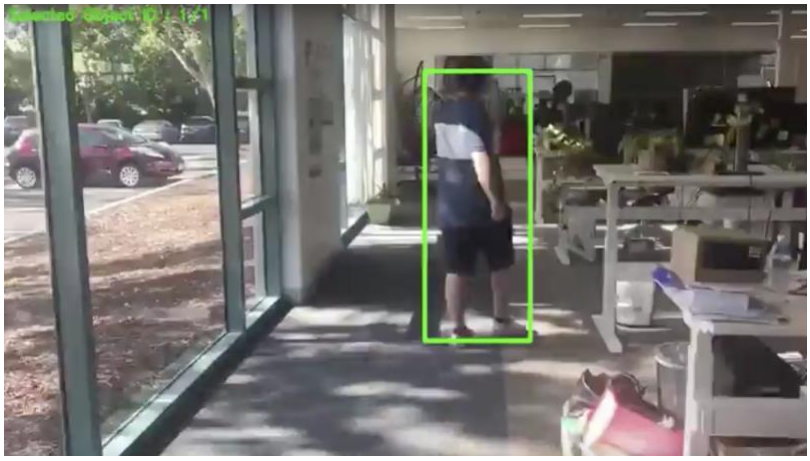
And just for some context, the video is of me walking around in an office. There are many people, as well as objects like computers, keyboards, books, chairs, etc. that serve as noise within the video.

Do note that words highlighted refer to directories that you can reference for code. I have written some READMEs for demos that you can run.

Let's begin by discussing the first task of my to-do list: object detection. I decided to go with Darknet's YOLOv3 because it is faster and more lightweight than other methods like Mask R-CNN. Rather than applying the model to multiple locations and scales within an image, YOLO applies a single neural network (53 layers) to the full image and then divides the image into regions, predicting bounding boxes and probabilities for each region. Because this model looks at the entire image at once, its predictions take in more global context. But the best part is that it makes predictions with only one network evaluation, making it 1000x faster than the basic RNN and 100x faster than the Fast R-CNN. Admittedly, YOLOv3 in some cases is slower than YOLOv2, but the increase in performance outweighs the slight loss in speed by far. I began by testing the most compressed pre-trained model, called YOLOv3-tiny. Although it was able to detect people in the video, confidence was low and smaller objects such as keyboards, computers, and (awkward shaped) office chairs were unable to be detected. Therefore, I decided to go with the full model (uncompressed), called YOLOv3. However, this model was slower to process (~60x slower from my estimates), so I let it run for some time. By the way, Darknet on its own wasn't designed to process video data, so I did some tweaking and made AutoLabelImages, which takes in and outputs results in video format (actually image format, but then uses ffmpeg to convert to video) as well as saves the coordinates of bounding boxes in .txt (this is useful in the later object tracking step). This also helps with visual evaluation needs, so that I can see whether the objects and labeled and bound correctly. The results turned out okay in that people, computers, etc. were able to be detected but chairs were still lacking. Therefore, I used the Grid at UChicago Booth to train a new model (150 epoches), using YOLOv3 as a starting point in training. I took data of people, chairs, and office supplies (computers, mugs, etc) from the 2012 VOC-Pascal training set. However, the VOC-Pascal dataset did not fit the format of training data that Darknet required, so I created Training-Preprocessing in order to get that ready. Training-Preprocessing is able to convert .xml labels to .txt and vice versa. It then organizes the folder structure required by Darknet and you can immediately begin training after running the program.

The second on my to-do list is object tracking. As I have already mentioned, while detecting all the images for each frame of the video, I had already saved the coordinates of the detected bounding boxes in the form of .txt files. For object tracking, I decided to use background-aware correlation filter tracking. I use correlation filters for tracking because it can learn how the object is changing over time and adapt to that. The background-aware correlation filter also makes use of histogram of oriented gradients in both the foreground and background of the area of focus. This is important because I need to distinguish one object from another, even if they are of the same class and have overlaps. BACF is implemented in the folder called Tracking. I also included some of my own additions into this implementation that would help with the overall accuracy. For one, the tracker is initialized based on the coordinates of objects detected from YOLO (as mentioned above). When the tracker's own coordinates are too far from that of the bounding boxes, the tracker resets onto the nearest bounding box of the same class. Despite the drawbacks to this approach, it is simple and most reasonable given my time constraints. Another thing I added is reverse tracking. I thought that by adding a feature to track objects backwards as well, we would be able to better refine the data. Since the input is a video, all this meant was to reverse the video and retrack objects, trying to match reverse tracking data and forward tracking data, purging out data that is unreliable. Each object we track is given an ID, so we can calculate how much an object has moved by looking at bounding box/tracker coordinate changes for the object of that ID. Although theoretically unlimited IDs are supported, I have limited it to 6 because of computational constraints. Do note that actually another good use of this tracking method is that I can save each processed frame as an image, and since we have bounding boxes (more accurate bounding boxes that is,

since we used the tracker), we are able to use this set of images and bounding boxes as groundtruth for training a new and improved model. This works especially well for weird objects that public datasets don't have. …although we would need a lot of videos to avoid overfitting.



That's me walking around. This is the demo you get from running what's inside the "Tracking" directory.

The third on my to-do list is generating captions for the image. Actually, this is where I ran out of time / got stuck, as the deadline for this project was approaching. I was planning to use the method proposed by Vinyals, Oriol, et al. (they also provided an implementation, so I just had to do some training), but I was unable to get the implementation to work due to conflicts in version with Keras and Tensorflow (don't have enough time to sort out this issue). However, on this note of image captioning, I was able to do some training using the Flickr 8k Dataset provided, through my request, by the computer science department at UIUC. Weights and implementation (mostly provided, but I made a few changes) are saved in the directory called Captioning.

In regards to the fourth item on my to-do list, which is choosing the best caption, I make use of the temporal domain. A lot of captions are generated, but I need to pick out the one that best fits. So, I look at the motion of objects relative to one another, to see which object has moved the most. For example, if 10 objects move in exactly the same way, we can assume that it is only the camera moving and not the object. In this way, we identify the class of focus and we search for that class name within the multiple captions produced, upvoting captions with that class name. Next, I use intersection of different objects (IOU of bounding boxes) as a proxy for interaction with those objects, especially if both objects have actively moved within the recent frames. When an interaction is identified, both objects of the interaction are deemed "key objects" and we filter out captions that don't contain those objects/classes.

The fifth item on my to-do list is segmenting the video. This step is simple. Since in the last step we already decided the captions, we simply compare the similarity of nearby captions and combine if similar enough (how similar is a threshold to be decided).

III. Experimental Result

In order to evaluate the results, I use a confusion matrix, as shown below:

```
***-------------- Aggregate Data --------------***

Average Accuracy (across all 105 files) : 0.837
Average Jump Error (across all 105 files) : 4.190
Average Entry Delay Error (across all 105 files) : 34.095
Average Exit Delay Error (across all 105 files) : -51.076


Confusion Matrix :
[34596, 804, 3354, 842]
[3312, 17144, 0, 16]
[4043, 521, 25028, 0]
[3992, 222, 65, 11621]

Skip Matrix :
[122, 0, 318]

Delay Matrix :
[1614, 392, 1574]
[-615, -2676, -2072]
```

The code I wrote for this is in the directory called CompareClassification. Essentially, what is does is that it compares what I labeled by hand (in a format like "0:00 – 0:45 is a person sitting down") with auto-generated video captions (in a format like "Frame 1: Person is standing up … Frame 30: Person is sitting down"). It gives data for each video, as well a summed total. In the image above, that is actually aggregate data from about 100 videos that I processed. The number of columns indicate the number of vastly different captions. In my videos, there are 3 primarily different actions, which is why there are 3 columns (the 4th column in the confusion matrix is the "doing nothing" column). How a confusion matrix works is self-explanatory. The skip matrix is simply the number of groundtruth frames that were skipped and classified wrong. The delay matrix tells us about both entry delay and exit delay, where positive numbers represent later and negative numbers represent earlier (i.e. beginning of action is too early, or beginning of action is too late, end of action is … etc.).

So anyways, as I mentioned above, I was able to make these individual components of the big project working, but unable to combine them due to issues with the version of Tensorflow/Keras compability.

The end result is that I have a pretty good tool for tracking specific objects throughout videos, which is refined through reverse tracking as well as compare to detected bounding boxes from YOLO. I also ended with a pretty successful metric of evaluation, as described above.

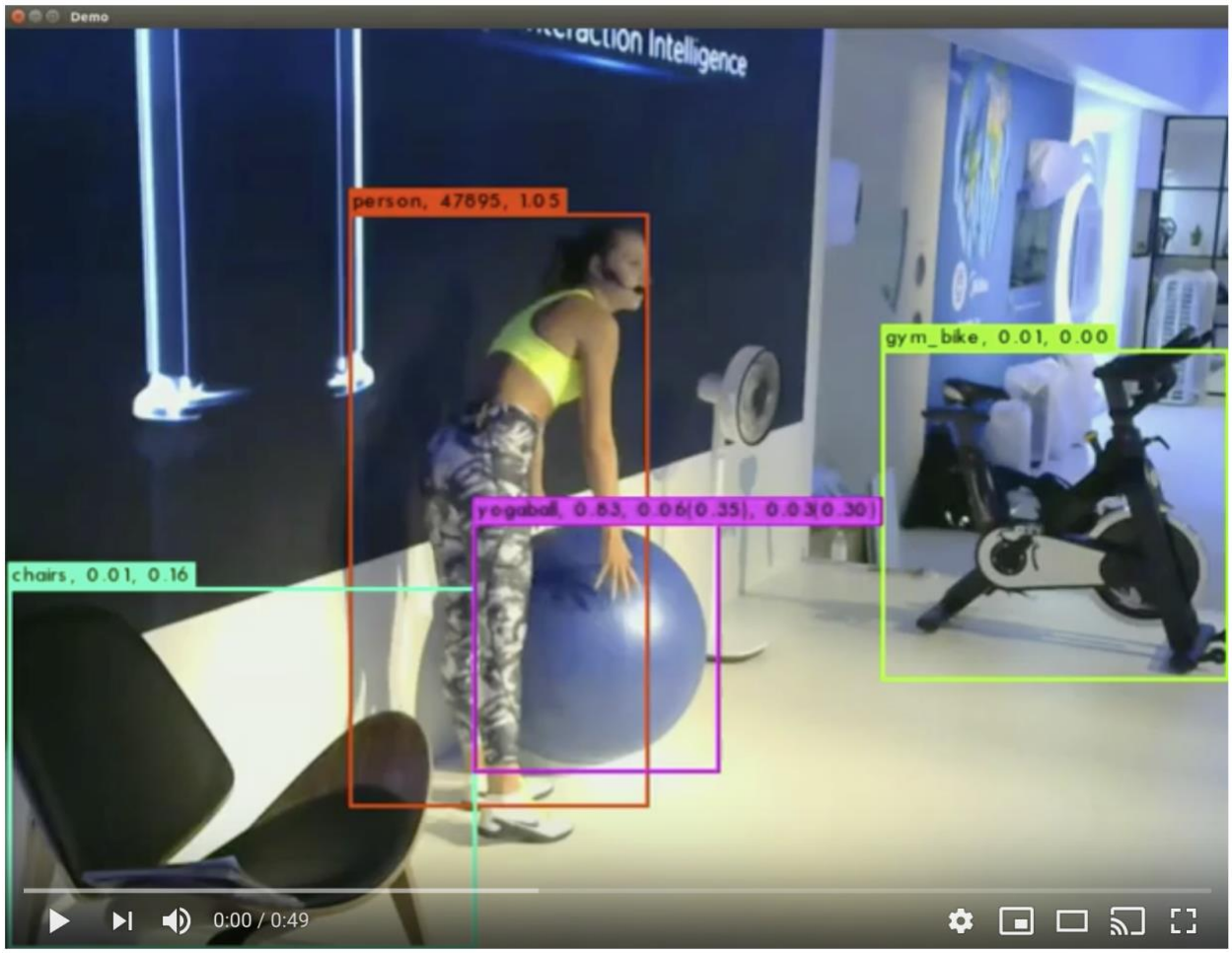
View a very rough demo in the appendix.

IV. Discussion

My experimental results tell us that it is very possible that in the future, video can be converted to text and automatically summarized. However, I did take several shortcuts, limiting my input to a very narrow range of videos. Nevertheless, the shortcuts that I took (and are mentioned above) I believe are reasonable given the human labor and time constraints of this project. My system performs okay, and can be improved by not taking the shortcuts that I did. But, since this field of research is largely underexplored, I believe that the fact that my system is functional is quite impressive. I also believe that with a little additional time and work, continuing this project can become quite meaningful and a big step in the field of video action recognition. This is useful is many applications as well, such as video surveillance.

V. Appendix

Very very rough demo (with minimal LP):



(If clicking the image doesn't work, go to this link: https://www.youtube.com/watch?v=TagiFFpsf_8)

VI. References

1. YOLO: https://arxiv.org/abs/1506.02640
2. Background-Aware Correlation Filter: https://arxiv.org/pdf/1703.04590.pdf
3. Image Captioning: https://www.cv-foundation.org/openaccess/content_cvpr_2015/papers/Vinyals_Show_and_Tell_2015_CVPR_paper.pdf
4. Confusion Matrix: https://en.wikipedia.org/wiki/Confusion_matrix