Object Detection in Videos

Akash Gaonkar
University of Colorado
akga1284@colorado.edu

Jacob Munoz
University of Colorado
jamu0075@colorado.edu

Avery Olson
University of Colorado
avol4799@colorado.edu

1 Abstract

1.1 Questions we sought to answer:
There are many interesting questions related to object identification and classification, mostly because it is a relatively new field.

1.1.1 - Is it possible to modify the current best object classification algorithm to run quickly and accurately without needing a super computer?
1.1.2 - Can we find a way to reliably test an object detection algorithm on videos of varying speed and environments?
1.1.3 - Can we mine YouTube videos for interesting clips?
1.1.4 - Can we use these clips to improve an object detection algorithm?

These are the main questions we hope to answer by tweaking an algorithm that already exists and making faster without sacrificing accuracy.

1.2 Brief summary of our results: In our project, we managed to take an algorithm called YOLOv2, the fastest algorithm currently for object identification/ classification, and we made it ~20 times faster on a weak machine without a huge hit to classification ability. This an accomplishment because so far with object detection and classification researchers often have to choose between number of identifiable classes and processing speed. In other words, if the algorithm is fast, the accuracy is low, and vice versa. The rest of the paper will explain how we managed to achieve both.

2 Introduction

2.1 Description of our questions: With artificial intelligence and machine learning improving at such a rapid rate, we find it important that the average person be able to run an object detection algorithm on their own computer.

Question 1.1.2 is one of the main issues of object detection, the algorithm can seem “confused” if a video is too quick or the environment is unknown. This is why we chose YouTube videos with a variety of environments and speeds.

Question 1.1.3 was created because YouTube is such an easily accessible source for free data. If we can search through hours of footage and organize every frame in a way that it is easy to select testing clips it becomes much easier to test our algorithm.

Question 1.1.4 brings the project full circle. Being able to use the clips we mined from YouTube allows us to easily and accurately manage our testing. Being able to marginally increase the amount of motion in a testing clip is extremely helpful.

2.2 Why our questions are important:
The reason we chose the questions listed above in 1.1 is because it was hard to pick specific questions for our project. We could not accurately predict what we would be to accomplish, so we chose to generally advance this field. We see the future of object recognition to be helpful for humans in many different industries to monitor a wide variety of things. For example, people make mistakes and may miss a harmful weapon while watching a busy terminal. In the future, airport security cameras could identify threats if the algorithm could work in close to real time. Human error is unfortunate and also one of the most profound reasons that many events happen. Therefore, making computers more reliable to accompany what humans are capable of is the key to future. But in order for object recognition tools to be able to function in real time, they must be fast. Which is why our main objective for this project is to improve on what already exists and to provide a way to more reliably tests these algorithms.

3 Related Work
In March of 2017, Google announced its Video Intelligence API. With this, developers can make applications for searching for specific objects in videos and tagging scene changes. It automatically detects objects in videos, which before this, only could still images could be used for object detection. It can also recognize words and characters of several languages, detect major landmarks and logos, and has a component called safe search, which filters inappropriate content. The safe search feature does not give the percentage but says whether it is very unlikely, unlikely, possible, likely or very likely for the object to be in the video. The Video Intelligence API uses Google Cloud Storage to hold the images and videos. To the right of the video/image it will give you the percentage probability an object is in it.

Similar to this is a website called Clarifai, which does a similar thing with the percentage likelihood of an object in the image/video, filtering inappropriate content, and having the ability to tag and search the objects found.

Another important part of the all these object detection algorithms, is that there is two parts to each: object detection and object identification. *Object detection* usually consists of placing what are called bounding boxes around objects, to separate individual items. This may seem like an easy task, but we must remember that all a computer sees is individual pixels. Second is *object classification*, which is then identifying what object is in that bounding box. Knowing these terms will help you understand our work.

Our work builds on work done by researchers at the University of Washington, on an algorithm called YOLO, short for You Only Look Once. YOLO combines object detection and classification into a single network. This lets the classifier combine two types of datasets:

- Sets for object detection that have ~50k examples and multiple carefully drawn and labeled bounding boxes on each example
- Sets for classification that have 1M+ examples of single objects where the whole image can be considered a bounding box.

As is often the case with machine learning, this vastly increased sample count gives the classifier significantly more power than was traditionally the case. Incremental updates to the classifier's design led to the release of YOLOv2 and most recently (released in March 2018) YOLOv3. We integrated YOLOv2 into our system as v3 is still extremely new and tools for using it have yet to be built.

4.1 Data Set

4.1 Where is our data from?

Although there is already a copious amount of data in the world, we chose to create our own data. The origin of our data is from three youtube videos of considerable length, from the links below:

4.1.1 Boat moving through a harbor:
https://youtu.be/CXECg509GJk?t=33s
4.1.2 Biker riding through Indonesian countryside:
https://youtu.be/V98uMzYBlxU
4.1.3 Cars driving through an intersection in New York City:
https://youtu.be/MBxv-OnUZhU

4.1.4 Raw data with jupyter notebook report:
https://github.com/jamu0075/DM_Video_Analysis/tree/master/code/data_processing

4.2 Why This Data Set?

The reason we chose the videos above is because we wanted to have a variety of objects and speeds in the videos. The main goal of our data is to be able to identify clips of varying speed: slow, medium, and fast. By doing so, we can incrementally test our algorithm, starting with nearly motionless clips and slowly selecting faster clips as improvements are made to the algorithm. Essentially we want to extract a few clips (roughly ten seconds long) out of ten hours of video.

4.3 Data Collection

We extracted every frame from ten hours of video to accumulate 1.1 million frames. We then took the frames and converted them to the grayscale version of the image. This is important because we need to take the difference between each adjacent frame, and it is much more efficient to find the difference between a grayscale image than a full color image. Every frame has an associated number that represents the relationship of the colors of the pixels in it. This is why we are able to essentially 'subtract' two frames to get a frame difference value. A frame difference value with a high number implies lots of movement in between frames while a low number means less movement. When you play the video with the diffence value as frames rather than the original frame it creates a black canvas with moving objects shown as white. This essentially creates a visual that represents how the computer determines a frame difference score.

Here you can see a house in the background and a man moving in the middle of the frame as an example.

From this point we have obtained 1.1 million data points that represent the differences between each subsequent frame. The data starts out as three separate text files, each containing the frame differences for our three videos.

4.4 Data Cleaning

The next step is to transform this data into something we can actually work with. The goal is to get all these points into one Pandas Dataframe that is easily interpretable. Unfortunately you can't just throw each text file into a Dataframe because you need to be able to identify time frames from different videos and as is we just have the frame difference values.

To accomplish this we loaded each text file into its own Dataframe and gave them attributes for source video ("Boat", "Car", "Bike") and an additional column for index (i.e 1 - 97860) to maintain the frame order for each video. These attributes are crucial because without them there is no way to distinguish where the desired clip originates from.

With these additional attributes we can safely concatenate our various Dataframes into one which contains every frame difference from every video with attributes allowing us to easily find the desired clips from the source video.

Provided is the link to the GitHub repository where the program to collect frame differences is held:

4.4.1 Frame difference calculations/ collections - https://github.com/jamu0075/DM_Video_Analysis/blob/master/code/clip_finder/frameDiff.py

5.1 Mining the Data for Clips

Now that we have a workable Dataframe we can extract clips to test our improved algorithm.

To start we parsed the ten hours of frame difference data into ten second clips, we chose ten seconds because it is a short enough clip that it won't take long to run an object detection algorithm on it yet it is long enough to measure any increase or decrease in algorithm efficiency.

Next we computed the average frame difference of those ten second clips, and stored this information in a list to manipulate so as to not affect the raw data.

Now that we have all of our clips associated with a value it is possible to search for clips we want.

The plot of the clips shows that the majority of the clips are moving rather quickly and skews the graph. This is to be expected since roughly 70% of our clips originate from a constant motion video. However, this skew does not negatively affect us because we only need clips for testing, not training, therefore we only need a few.

We started by sampling clips that are close to Q1 because we know these will provide clips that are slow moving and easier on an object detection algorithm. The closest clip to Q1 was of a Police Officer stopping traffic in the New York City traffic intersection video. This is good because when viewing the clip with the frame difference filter we can clearly see that a majority of the frames are black, indicating we accurately identified a ten second clip with a desired speed.

Here you can make out the officer while a few cars pass by. Notice the majority of the image is however black, indicating not much movement.

After testing our improved object detection on this clip we found it had no problem identifying everything in this clip and we could increase the threshold to find a slightly faster clip.

This process could be repeated until you are unhappy with the performance of the object detection and you are forced to make adjustments to improve the speed.

With our own studies we found our improved algorithm did fairly will with clips around the mean and had noticeable performance drops with clips around Q3.

The importance of all this is that we now have the ability to accurately test for marginal improvements on object detection. It is possible to find a ten second clip out of ten hours and then find another ten second clip that is marginally faster than the previous to see how algorithms handle them.

For more details regarding this process and source code see this report: https://github.com/jamu0075/DM_Video_Analysis/blob/master/code/data_processing/Video_Breakdown.ipynb

5.2 Improving Our Algorithm

The first milestone that we completed was first to download and install things in order to get YOLOv2 working (our chosen pre-existing algorithm). This took more time than we had originally planned for, but eventually we got it working and each of our computers and could begin our work on it.
Then, having extracted our clips as described in the previous section, we proceeded to test and modify our algorithm to try and better predict bounding boxes.

Our first iteration was running on YOLOv2 directly on the main thread loop. This was extremely slow, with Akash’s computer barely managing 0.5fps. Since the YOLO algorithms, like most machine learning algorithms are designed for high processing systems with many GPUs, it was reasonable that it was very inefficient on a laptop with 4 physical CPU cores and Intel Integrated Graphics (eg no GPU at all).

We next isolated the YOLOv2 algorithm to its own process, using the fact that we had a multicore machine to separate the CPU intensive machine learning from the input/output and cpu intensive video loading and processing. This managed around 4fps.

After that we tried to perform basic motion prediction by sending two consecutive frames of the video to the predictor, and then performing some physics to predict the velocity of the object to move the bounding box alongside it. This did not affect the frame rate, since the predictor was in a separate process, but did double the predictors time to response, and also produced unusable velocities. We scrapped this work and reverted to the previous attempt.

What was eventually successful for our team was looking into OpenCV trackers, and we found we could take the bounding box results from YOLOv2 and produce a tracker for each detected object. This was extremely accurate, but at large numbers of detected objects, reduced our framerate back to 0.5fps (and it was inefficient to subprocess our trackers — we tried it). To solve this problem, we adjusted the algorithm so that only one tracker was updated per frame, giving us more accurate data when fewer objects are detected and less accurate data when more are (since on average, each is less important).

5.3 Tools Used
The simplest way to get object detection running is to hook up an existing tool, such as Google Cloud Vision API, Tensorflow Object Detection API, or Darkflow, a Python/Tensorflow program that would make YOLO-style classifiers available from both a command line interface and via Python imports. Specifically, we chose to import Darkflow's YOLOv2. The main tool for coding our project was Python 3. We used openCV in order to analyze and parse our source videos into the data we needed (frame difference values), as well as in parts of our algorithm. We used a Python Jupyter Notebook to create our histogram and analyze our final cleaned and formatted data set.

6 Key Results

6.1 What did we discover? We have found a way to reliably find clips of videos with a desired amount of motion and variation. We discovered that a video's "score" has a direct relationship with the amount of movement in video. This was a key point for us to run our analysis, create histograms, and determine where to find clips of the desired speeds for our test set.

6.2 Cheapo compared to other versions: Cheapo, our algorithm, detects around 30 classes accurately, runs at about 8-15 frames per second (on a laptop with no GPU), has medium bounding box accuracy and low consistency.

tinyYOLO detects around 5 classes accurately, runs at about 8 frames per second, has high bounding box accuracy and high consistency.

YOLO detects around 30 classes accurately, runs at about 0.25 frames per second, has high bounding box accuracy and high consistency.

We managed to get the speed of tinyYOLO with the class count of YOLO, an extremely significant feat, if there were no other side effects. We feel that the side effects of our method, which does have worse bounding boxes and may sometimes lag them relative to the frame are well worth the costs for full strength, real time object detection.

7 Applications

7.1 How can the knowledge we gained be applied? Using the method of data mining that we did, creating our own data set, it becomes much easier to test for algorithm improvements. You can see even slight increases in algorithm speed, or analyze clips that are slowing the algorithm down. Different and useful test clips had a great impact on our ability to test and develop our algorithm.

7.2 Casual Real Time Object Detection
While real time object detection has been achieved, it's often unusable in most applications because of the need for specialized hardware. Our system runs quickly and without sacrificing detection power, bringing the world of real time detection to casual machines!

7.3 Further work to be done:
Potentially, you could take our work a step further by teaching the algorithm to speed itself up by rewarding speed increases. We also need to reduce a bounding box lag problem we're experiencing due to techniques we used.

Below are some additional data sets that we feel would help improve the accuracy of our algorithm if we had the time to incorporate them. This was a part of our original project intention.

7.2.1 Open Images - 9 Million Images
https://github.com/openimages/dataset
The data contains URLs to images that have been annotated with image-level labels and bounding boxes spanning thousands of classes. All of the image-level labels are automatically generated by a computer vision model similar to the Google Cloud Vision API mentioned above. This data is split into three sets, the training set, the validation set, and the test set. The labels in the validation and test set are all human-verified image-level labels (done by Google annotators) as well as some in the training set. And the bounding boxes have also been manually drawn by Google annotators for all the human-verified image-level labels.

7.2.2 ImageNet - 14 million images
http://image-net.org/index
This data contains images for nouns, organized into a tree-like structure to become more specific leading the images for the more specific category.

8 Visualization

8.1 Demonstrations of our project:

Akash holding a pen as an example image.

The above image was from the beginning of our project, to show how we got our bounding boxes (in red) to successfully identify separate objects.

Below are a few more images from our project.

8.1.1 YouTube Video Link - https://youtu.be/jGrqurk3ef0

Citations

Huang, Johnathan. “Supercharge Your Computer Vision Models with the TensorFlow Object Detection API.” *Google Research Blog*, Research @ Google, 15 June 2017, research.googleblog.com/2017/06/supercharge-your-computer-vision-models.html.

Lardinois, Frederic. “Google’s New Machine Learning API Recognizes Objects in Videos.” *TechCrunch*, 8 Mar. 2017, techcrunch.com/2017/03/08/googles-new-machine-learning-api-recognizes-objects-in-videos/.

Mallick, Satya. “Home.” *Learn OpenCV*, Big Vision LLC, 13 Feb. 2017, www.learnopencv.com/object-tracking-using-opencv-cpp-python/.

Redmon, Joseph, and Ali Farhadi. “YOLO9000: Better, Faster, Stronger.” *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, doi:10.1109/cvpr.2017.690.

Redmon, Joseph, et al. “You Only Look Once: Unified, Real-Time Object Detection.” *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, doi:10.1109/cvpr.2016.91.

Redmon, Joseph. “YOLO: Real-Time Object Detection.” *Pjreddie.com*, Joseph Redmon, 2016, pjreddie.com/darknet/yolov2/.

Robinson, Sara. “Filtering Inappropriate Content with the Cloud Vision API.” *Google Cloud Big Data and Machine Learning Blog*, Google Cloud Platform, 17 Aug. 2016, cloud.google.com/blog/big-data/2016/08/filtering-inappropriate-content-with-the-cloud-vision-api.

Sato, Kaz. “Experience Google’s Machine Learning on Your Own Images, Voice and Text.” *Google Cloud Big Data and Machine Learning Blog*, Google Cloud Platform, 16 Sept. 2016, cloud.google.com/blog/big-data/2016/09/experience-googles-machine-learning-on-your-own-images-voice-and-text.