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 Problem Statement/motivation

The contents of a video are clear to any human that watches, but is it possible for a computer to understand what it is seeing? Can an algorithm learn to recognize objects in a video and alert humans of a potential threat? These are the questions we hope to answer by training and tweaking an algorithm that can recognize objects in each frame of a video.

If a computer could identify threats reliably it could help prevent violent events before they occur. While security agencies (such as those at the airport) do their best, people make mistakes and may miss a harmful weapon while watching a busy terminal. Having a computer to watch alongside people can confirm a human's suspicion or alert them of an unknown threat. 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 increase the speed of what already exists.

2 Literature survey

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.

3 Proposed Work

Right now with image detection in videos you can chose between high accuracy and speed of it running. You shouldn't have choose, and as mentioned previously, our main objective is to improve what already exists. To begin, we plan on replicating what has been done so far, and if we can, make it faster, therefore helping decrease the gap between speed and accuracy.

After we successfully replicate the object detection aspect, we can create larger bounding boxes and try to predict where an object will move in order to make it faster. And usually object detection and classification are trained with different data sets, but if we could combine them into one data set, it could improve the accuracy of our algorithm.

For our particular project, we will use the Lord of the Rings movies as our data set. First we need to pre-process the data so that we can work with it. The combined length of the Lord of the Rings series is 654 minutes, which multiplied by seconds and 30 frames per second, gives us 1,177,200 images after the movies are converted to frames. This conversion will be the first part of the code we add.

Then, the next objective is to use the movies in order to predict movement of objects. By predicting where an object is going to move, you don't have to analyze every frame, but rather, every 30 frames. That would already give a 30 times increase in the speed of the algorithm. Some accuracy would be lost since it is only a prediction of where the object would move, but the dramatic increase in speed would help close the gap between speed and accuracy.

3.1 Data Preprocessing

3.1.1 *Scaling*. The first step is to rescale the images to be of consistent form. The pre-existing algorithm we are working worth uses a frame size of 416 x 416, so we need to convert all the movie frames to this size.

It is important for all the images to be lined up exactly so the algorithm can compare images pixel by pixel without having to worry about varying image sizes.

3.1.2 *Outliers*. The next step is to clean the data by removing infrequent classes (image with objects). If an object only appears a few times within millions of images it is best to remove said image to improve the learning algorithm. Leaving outliers in the training data may actually decreases the overall performance of the algorithm.

3.2 Algorithm Training

Having acquired a sufficient corpus of labeled images, we will partition it into training, validation, and test datasets. The training set is for teaching our classifier to properly predict bounding boxes and object labels, while the validation set is for tweaking classifier parameters and the test set is for comparing classification algorithms if we have time.

We have several options for our project's object-detector. Aside from the built-in classifiers described later (in our Tools section), we can choose between various ML techniques and systems. Deep learning is the current champion at object detection, but even within that category there are several algorithms. We're particularly

intrigued by YOLOv2 (You only look once), which can process up to 45 frames per second under the right conditions, fast enough to detect objects in real time.

3.3 Evaluation

After running the algorithm through the training dataset it is important to check for reliability. One example is to ensure the algorithm can recognize a chair from multiple angles or lighting conditions. After the initial training has proved successful can we push the algorithm to recognize more obscure objects such as a man holding knife.

4 Data set

Like previously mentioned, we will be using all three of the Lord of the Rings movies, meaning there will be over a million frames to work with. But, our algorithm could have been used with any movie or video.

Below are two sets of images that we might use to further enhance our algorithm if we have time.

Open Images - 9 million imges - 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.

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.

5 Evaluation Methods

We will test videos in our version and compare the results to the Google Cloud Vision API and the Clarifai website. This will tell us how close or far we are from the current standard and if we can get more accurate results in the same timing, then we know we have made it better.

More specifically, to test our pre-processing of the data, we will test smaller videos of about 5 minutes length before running it on the movies. We will do this to see if we can successfully convert them to the correct size of 416 x 416 and also cut them up into frames.

From this point, we will also use these shorter videos to test if we can successfully predict the movement of objects.

6 Tools

We plan to use Python3 as the main language for our project, and Numpy will be used to help us properly format our images. From this launching point, we have various tool sets depending on how we choose to focus our project.

The simplest way to get object detection running would be to hook up an existing tool, such as Google Cloud Vision API, Tensorflow Object Detection API, or

perhaps Darknet, a C program that would make YOLO available from a command line interface.

In this case, we'd want a toolset useful for building something on top of object detection, so we might use a Python/Flask backend serving pages with a Polymer 2 or Angular 4 frontend, maybe with npm and bower for dependency management.

Alternatively, we could focus on training our own instance of one of the neural net algorithms, creating each layer and running the images through our model. This would involve us much more deeply with the actual construction of an object detection tool, and so we'd be more inclined to use pure Tensorflow (as opposed to the apis it makes available). This would make us rely much less heavily on the web toolkit above, since we'd be creating a more generic detector rather than an application.

7 Milestones

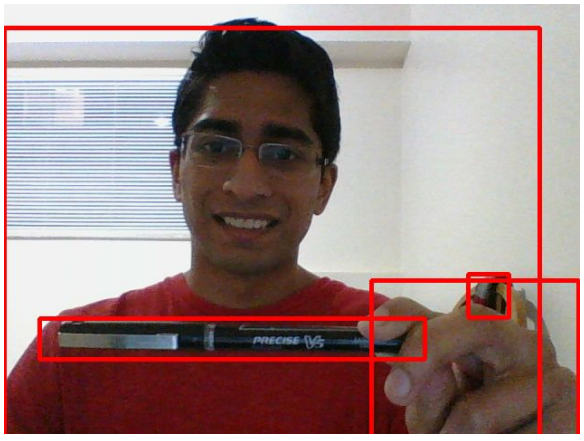
7.1 Milestones: We would like to try and pursue an agile-ish methodology with our project, and so have made detailed plans for near timepoints, but would like to be more flexible after that.

Starting 3/11, there are approximately 7 weeks before the end of the project. We will break the project up into three phases. Phases 1 and 2 will focus on getting object detection working by both pre-processing our data and training our algorithm. We expect to iterate over the two phases several times before arriving at something stable, so this process should take 3.5-4 weeks (milestone on 4/2). The remaining 3 weeks, phase 3, are reserved for taking a next step to our tool, be it finding an application or focusing even more on streamlining our detection algorithm. Phase 1's first tasks are to 1) find a way to loop through our dataset, since we can't just download every image, 2) properly shape each image to have the same dimensions, 3) test existing object detection tools and see if we can get a working system prior to integrating it with our dataset.

7.2 Milestones Completed: First, we tested several object detection tools and determined the one we plan on replicating. We also have successfully written python code to convert video to frames, which we have tested on smaller videos. In addition, we successfully replicated the object detection aspect, and have begun testing several speed increases. We are currently working though some errors in combining these two aspects, but expect to be moving along to the next part within this week.

7.3 Milestones To Do: After we finish working though some errors, we will be able to test the code we have written that added onto YOLOv2, and see if any improvement has been made at that point. From there we we can focus on predicting object movements.

8 Results So Far



Akash holding a pen as an example image.

As mentioned above, we have successfully got the object detection aspect working (shown in the image below). More to come!

9 Citations

https://techcrunch.com/2017/03/08/googles-new-machine-learning-api-recognizes-objects-in-videos/

https://cloud.google.com/blog/big-data/2016/09/experience-googles-machine-learning-on-your-own-images-voice-and-text

https://cloud.google.com/blog/big-data/2016/08/filtering-inappropriate-content-with-the-cloud-vision-api

https://research.googleblog.com/2017/06/supercharge-your-computer-vision-models.html

https://www.cv-foundation.org/openaccess/content_cvpr_2016/papers/Redmon_You_Only_Look_CVPR_2016_paper.pdf

https://pjreddie.com/darknet/

https://blog.miguelgrinberg.com/post/the-flask-mega-tutorial-part-i-hello-world

https://www.polymer-project.org/about

https://pjreddie.com/darknet/yolo/

https://arxiv.org/pdf/1612.08242.pdf