Project 5: CarND Vehicle Detection and Tracking

Vehicle Detection and Tracking Project

The goals / steps of this project are the following:

- Perform a Histogram of Oriented Gradients (HOG) feature extraction on a labeled training set of images and train a classifier Linear SVM classifier
- Optionally, you can also apply a color transform and append binned color features, as well as histograms of color, to your HOG feature vector.
- Note: for those first two steps don't forget to normalize your features and randomize a selection for training and testing.
- Implement a sliding-window technique and use your trained classifier to search for vehicles in images.
- Run your pipeline on a video stream (start with the test_video.mp4 and later implement on full project_video.mp4) and create a heat map of recurring detections frame by frame to reject outliers and follow detected vehicles.
- Estimate a bounding box for vehicles detected.

Overview

The goal of this project is to understand and implement pipeline for Vehicle/Car detecting solution for the recorded video as input. To start with understand the SK learn Support Vector Machines (SVM). Using sklearn.svm algorithms in doing Support Vector Classification (SVC) on images provided part of the projects as Vehicles and Not Vehicles category.

In this project, we are using the Histogram of Oriented Gradients (HOG) a popular model for object detection. HOG algorithm uses multiple steps computing the gradient image in x & y, computing gradient histograms, normalizing across blocks and flattening the feature vector using ravels using scikit-image APIs.

Also used new techniques sliding windows with overlapping sliding sub-sampling of mages to detect the vehicles. Avoided some of the false positives using the Heat map method and applying threshold. Finally, the single image process image pipeline is used for the provided test and project video.

Files Submitted & Code Quality

Required Files:

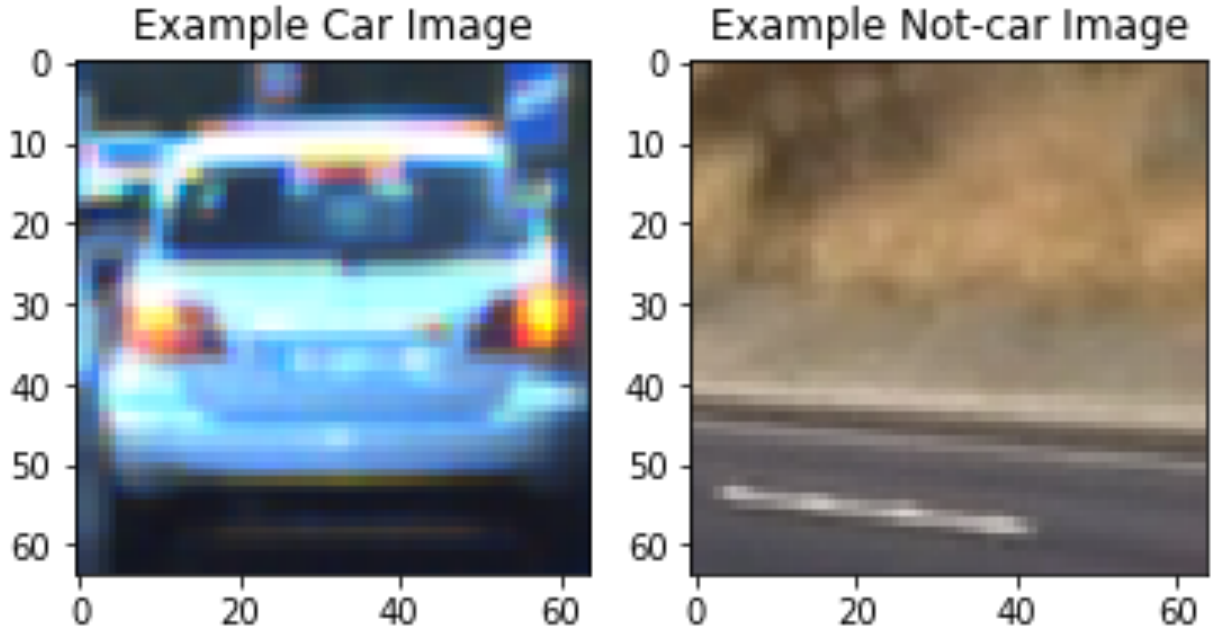
My project includes the following files:

- CarND-Vehicle-Detection.pynb containing the code for the classification, image pipeline and video creation
- VehicleDetectionAndTracking.html file with the test images
- writeup_report.pdf summarizing the results
- project_video_out.mp4 is the recorded video of the final image processing using the pipeline
- Test images are provided part of CarND-Vehicle-Detection.pynb.pynb as well as in writeup_report.pdf for various stages of the pipeline

Histogram of Oriented Gradients (HOG)

HOG image extraction from training set:

Used the Vehicles and Non-Vehicles images provided part of the Udacity repository. Extracted the image of cars and notcars using the glob.glob API and added it to cars and notcars list.

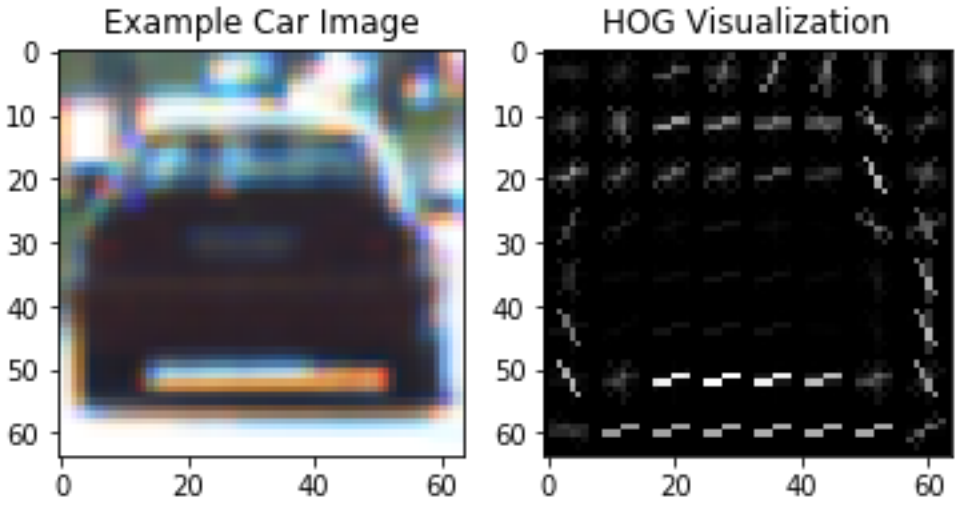


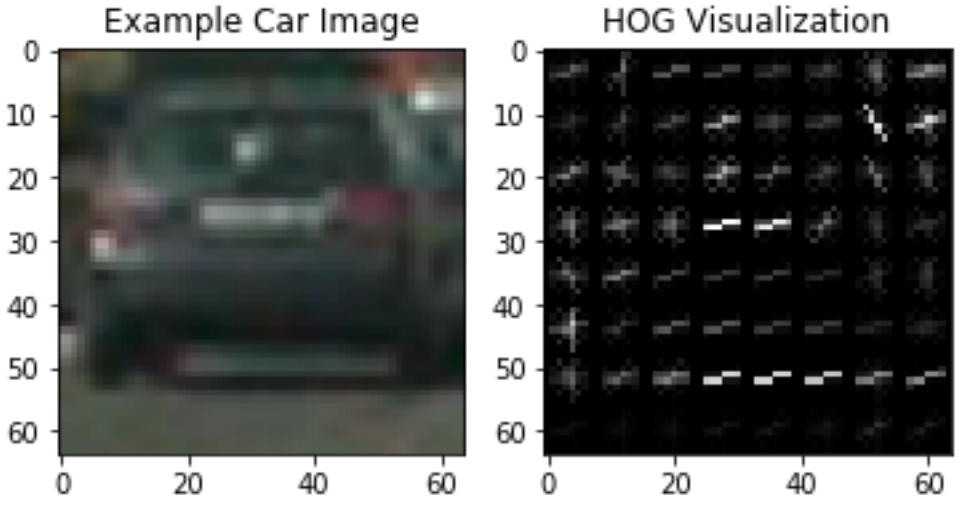
```
print('Function returned a count of',
      data_info["n_cars"], ' cars and',
      data_info["n_notcars"], ' non-cars')
print('of size: ',data_info["image_shape"], ' and data type:',
      data_info["data_type"])
```

Below are the number of cars and notcars images extracted from the repository, and the images of size (64, 64, 3).

```
Function returned a count of 8792  cars and 8968  non-cars
of size:  (64, 64, 3)  and data type: float32
```

Used the skimage hog API and created a get_hog_features function to extract the hog image and features. Below are couple of HOG images samples.





HOG parameters:

Tried various combinations of HOG parameters, including:

- Color Space
- Channel (ALL, 0, 1, 2)
- Orientation
- Pixels per Cell
- Cells Per Block

The below tables list the various HOG parameters and values, and the Extraction time for the Sample Size of 500 in seconds.

#	ColorSpace	Channel	Orient	Pixels per Cell	Cells Per Block	Extraction Time (In Seconds)
1	RGB	ALL	11	16	2	2.87
2	RGB	0	11	16	2	1.29
3	RGB	2	11	16	2	1.31
4	YUV	ALL	11	16	2	3.06
5	YUV	0	11	16	2	1.57
6	YUV	1	11	16	2	1.35
7	YUV	2	11	16	2	1.35
8	YUV	ALL	9	8	2	3.64
9	YUV	0	9	8	2	1.93
10	YUV	1	9	8	2	1.99
11	YUV	2	9	8	2	1.69

Trained Classifier with HOG features:

Finally used the YUV colorspace with 'ALL' hog channel. Below are the list of Parameters in training and testing the Classifier:

- **ystart = 400**
- **ystop = 720**
- **scale = 1.5**
- **color_space = 'YUV'**
- **orient = 11**
- **pix_per_cell = 16**
- **cell_per_block = 2**
- **hog_channel = 'ALL'**

Used sample size of 500.

```
car_features = extract_features(cars, color_space=color_space, orient=orient,
                    pix_per_cell=pix_per_cell, cell_per_block=cell_per_block,
                    hog_channel=hog_channel)

notcar_features = extract_features(notcars, color_space=color_space,
orient=orient,
                    pix_per_cell=pix_per_cell, cell_per_block=cell_per_block,
                    hog_channel=hog_channel)
```

```
3.26 Seconds to extract HOG features...
colorspace: YUV orient: 11 pix_per_cell: 16 cell_per_block: 2
```

Used SVC classifier by utlilizing the repository images to train the classifier, 20% data for Testing the classifier. SVC classifier was able to predict almost 100% with this model.

```
Using: 11 orientations 16 pixels per cell and 2 cells per block
Feature vector length: 1188
0.04 Seconds to train SVC...
Test Accuracy of SVC =  1.0
My SVC predicts:  [ 1.  1.  0.  0.  0.  0.  0.  0.  1.  0.]
For these 10 labels:  [ 1.  1.  0.  0.  0.  0.  0.  0.  1.  0.]
0.0018 Seconds to predict 10 labels with SVC
```

Sliding Window Search

Sliding window techniques:

Sliding window is a technique used to search the windows that matches the vehic le. It takes various parameters and fine tuning in terminal the correct window.

The parameter includes:

- Scales
- Overlaps
- Start and Stop points
- Colors for the windows

Below are the parameters used to determine the windows that fit in the lower part / interesting part of the image.

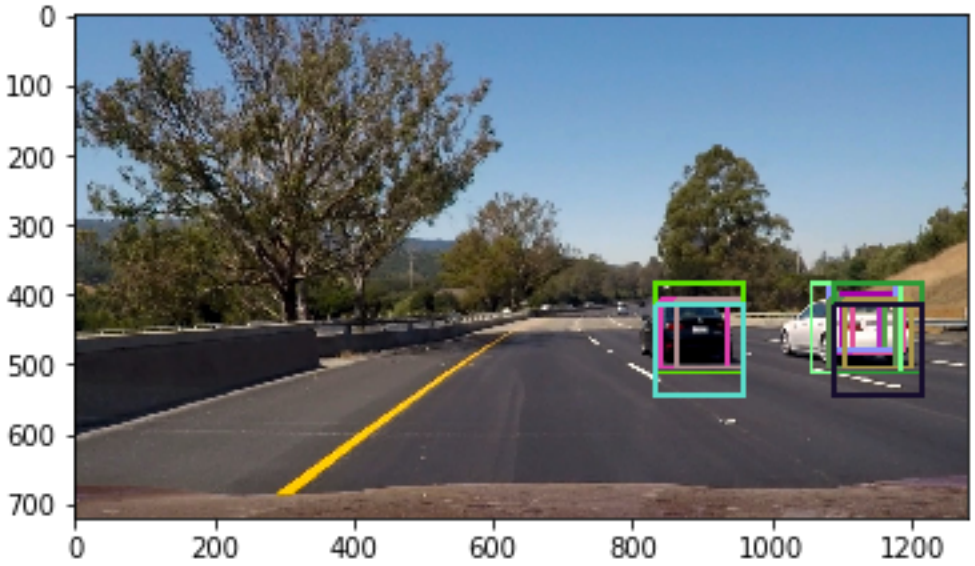
```
image = cv2.imread('test_images/test1.jpg')
image = image.astype(np.float32)/255

scales = [(40,40), (60,60), (90,90), (120,120), (170,170)]
overlaps = [0.25, 0.25, 0.5, 0.75, 0.75]
y_stops = [500, 550, 600, 650, None]
x_start_stops = [[400,850], [360,890], [320,930], [None, None], [Non
e, None]]
colors = [(0,255,0), (255,0,0), (128,128,0), (0,128,128), (128,0,128
)]
```

HOG sub-sampling Window search:

The more efficient method of doing the sliding window approach, it allows us to use not only the HOG features but also helps to extract features and make predictions.

The find_cars has to extract hog features once and then can be sub sampled to get the overlaying windows. Each window is defined by the scale factor. Here is a test image.



Video Implementation

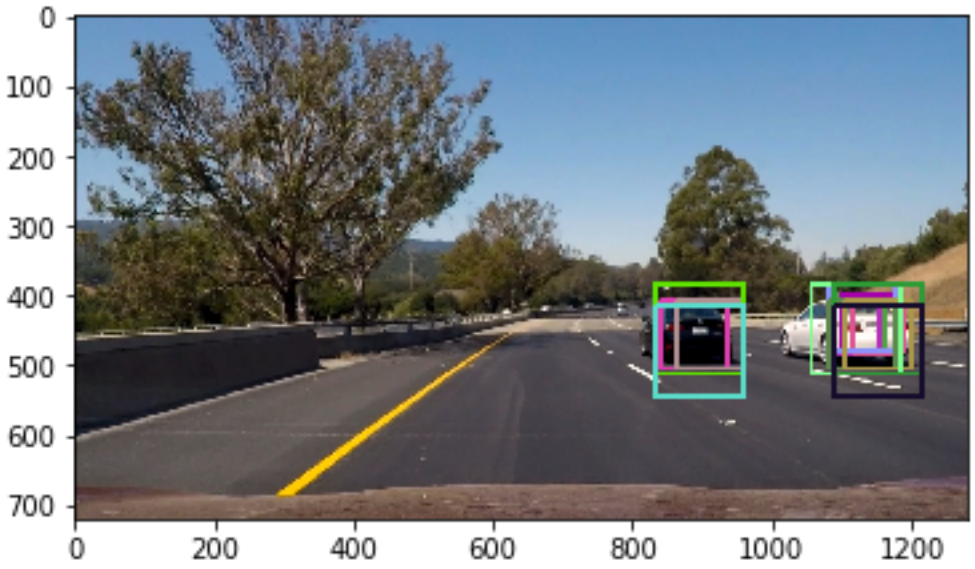
Sliding window and classifier:

The sliding window search plus classifier has been used to search and identify the vehicle.

Used colorspace = YUV, orient =11, pix_per_cell =16, cell_per_block =2 and the hog_channel = 'ALL'. Used the find_cars providing the image, and various hog parameters and SVC classifier as inputs.

The find cars uses the trained SVC classifier model, with the hog features to do the test prediction. find_cars returns the list of boxes or the windows that is identiried.

The same find cars has been called multiple times to get the list of box lists.



Used the bbox_list found during the multiple iterations of find_cars with the variou s parameters of ystart, ystop and scale values.

Sample bbox_list windows.

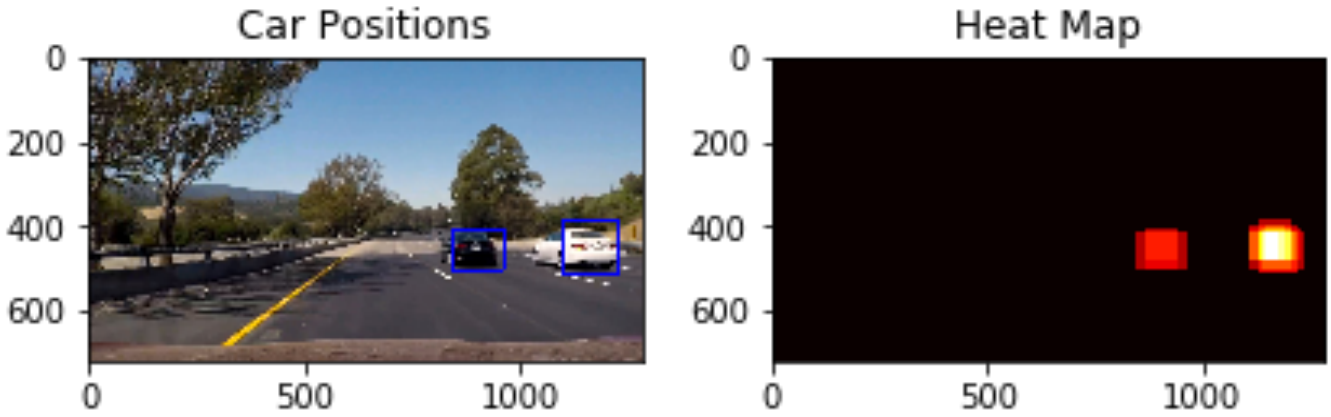
```
[((112, 400), (176, 464)), ((864, 410), (936, 482)), ((1098, 410), (
1170, 482)), ((1134, 410), (1206, 482)), ((1100, 400), (1180, 480)),
 ((1120, 400), (1200, 480)), ((1140, 400), (1220, 480)), ((840, 408)
, (936, 504)), ((864, 408), (960, 504)), ((1104, 384), (1200, 480)),
 ((1104, 408), (1200, 504)), ((1128, 408), (1224, 504)), ((576, 416)
```

```
, (704, 544)), ((608, 416), (736, 544)), ((832, 384), (960, 512)), (
(832, 416), (960, 544)), ((1088, 384), (1216, 512)), ((1088, 416), (
1216, 544)), ((1120, 384), (1248, 512)), ((1120, 416), (1248, 544))]
((112, 400), (176, 464))
((864, 410), (936, 482))
((1098, 410), (1170, 482))
((1134, 410), (1206, 482))
((1100, 400), (1180, 480))
((1120, 400), (1200, 480))
((1140, 400), (1220, 480))
((840, 408), (936, 504))
((864, 408), (960, 504))
((1104, 384), (1200, 480))
((1104, 408), (1200, 504))
((1128, 408), (1224, 504))
((576, 416), (704, 544))
((608, 416), (736, 544))
((832, 384), (960, 512))
((832, 416), (960, 544))
((1088, 384), (1216, 512))
((1088, 416), (1216, 544))
((1120, 384), (1248, 512))
((1120, 416), (1248, 544))
```

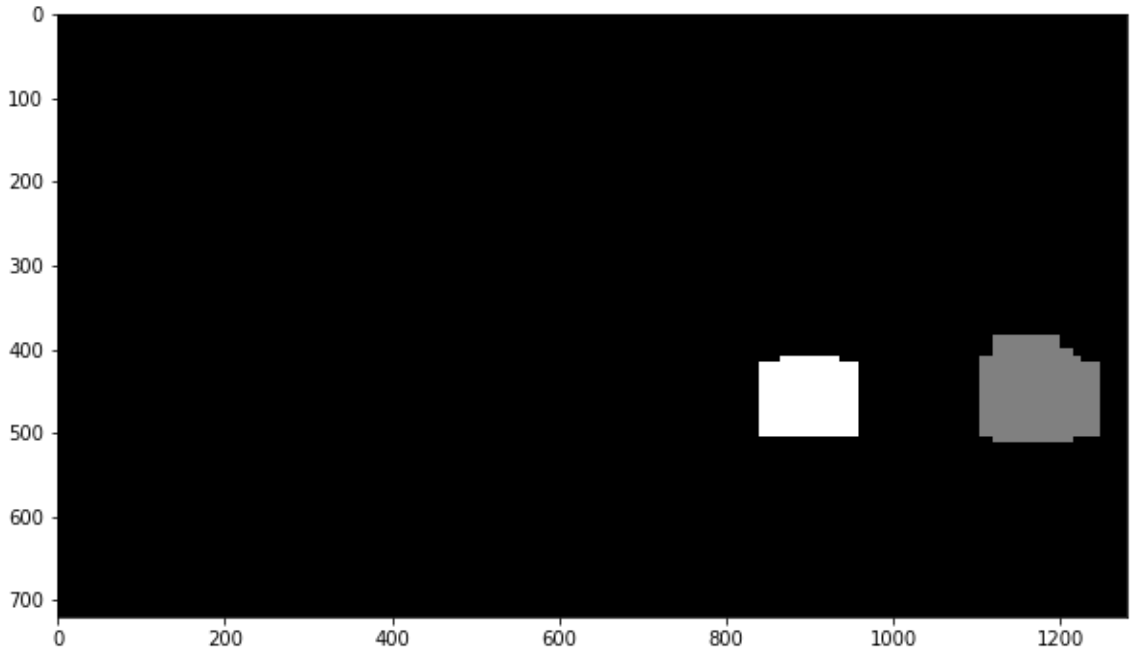
Avoiding false positives & Multiple detection:

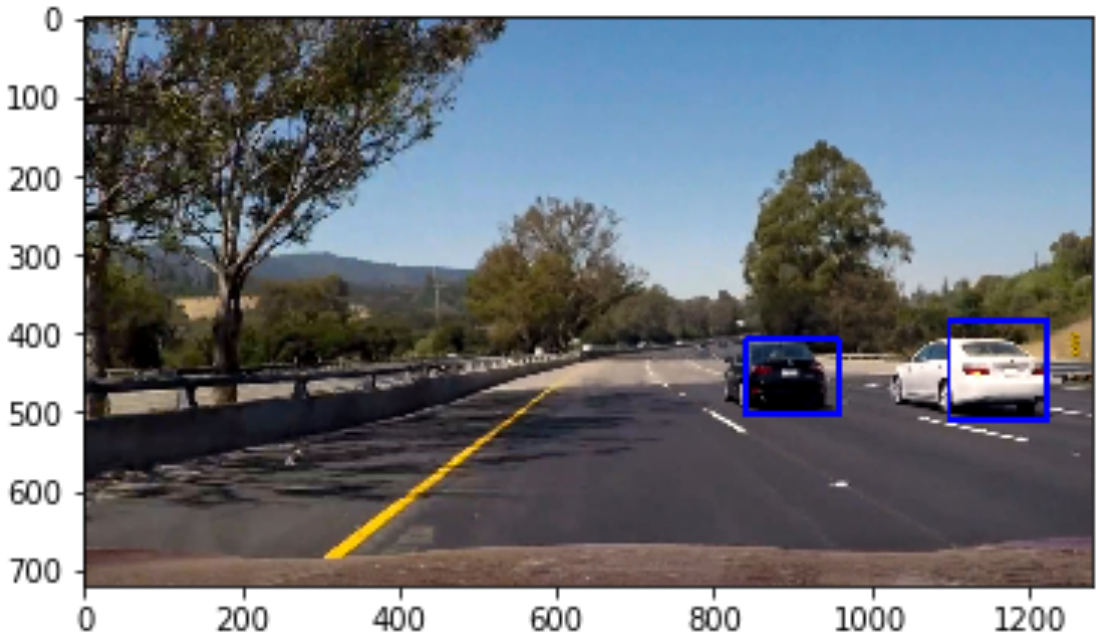
In order to avoid the false positives used the heat-map function by increasing the value of heat +=1 for all the pixels within the windows, where a positive direction is reported by the classifier.

Below is the heat map of a test image, and associated vehicle detection.



Here is the final output of the heat map image, that removes any false positives a nd and used the labelled detection in plotting the car boxes.





Used the moviepy.editor function VideoFileClip to read the given image project_video.mp4 and used the function to read each images in the video file and used the *process_image_new* and saved the filed to project_video_out.mp4 . Also used subclip to identify and address some of the false positive detection.

Conclusion, Future Work

- Was able to use the various concepts taught during the course SVM, SVC functions and various parameters and new concepts of heat maps and sliding window detections
- As an improvement to the code, will try to avoid few false positives see in the final video
- Will also try with the different videos in identifying the vehicles, and see how this code scales. Also will try using spatial and histogram features in refining the video.