**Traffic Sign Detection**

Alex Dodd, Tom Casaletto

https://github.com/TomCasaletto/CMPE258

horizontal line

# Abstract/Introduction

This paper explores work related to object detection, tracking and classification in real time environments. Several articles were researched to determine the best algorithms for classification as well as how to most efficiently read incoming real time video feeds. In addition, tracking moving objects in a scene was investigated.

Our goal for this paper was to create a traffic sign detection program that could be used in the real world. It would use streaming video input to determine where a traffic sign was located within an image, place a bounding box around the sign, and label the bounding box with the appropriate sign designation. Applications for a program similar to this are speed limit detection, stop sign detection, and many other self-driving applications.

# Survey of Related Work

A survey of our related work can be found in the References section of this paper. Most of our research pertained to how to convert a tensorflow model to a frozen graph so that it could then be read into OpenCV for object detection. Unfortunately, after much effort was put into this endeavor, no solution was found as it appears many others on the Internet have the same issues loading their custom models into OpenCV through Tensorflow. This will be discussed more later in this paper. For now, the main consensus is to utilize PyTorch, convert to an ONYX model and then load into OpenCV in the future. This was not able to be explored during the timeframe of this paper.

It is worth noting that much of the code for object-detection was sourced from PyImageSearch.com. Several updates were made from the source code to allow it to read in video data from a camera connected to our computers.

# Technical Approach

The traffic sign dataset utilized for this project was sourced from Kaggle.com and contains images of a variety of different road signs along with a label indicating which type of road sign was present in the image. This data set contains 50,000 instances with more than 40 different classes. The entire dataset was 314 MB. This data had already been pre-processed to only include the cropped image of the sign and did not have any background included in the image.

The video data used in the project was taken from reference video tracking algorithms. Additionally a number of video sequences were obtained manually by the authors.

## Training an Object Detection CNN

First, a function was written to load all the data from the GTSRB dataset. We pulled all the images and image labels and appended them to our list for ingesting into the CNN. Prior to appending them to our list, we resized all images to 32 x 32 and performed an adaptive histogram equalization to help keep light levels and contrast levels equal across all images.

We then one-hot encoded our labels using the “to\_categorical” function. This forced our label vector to be the length of the number of labels available.

To further add variation to our dataset, we used an image data generator to augment our images. This would make our CNN more robust to all sorts of images in the future that it would need to classify. We set the image generator to rotate, zoom, change the width and height, and shear our images. This meant that if incoming pictures to our dataset were not pristine, these augmented images in the training set would help classify imperfect images.

For our training of the CNN, we used Adam as our learning rate optimizer and set our number of epochs to 30.

Our final accuracy on the training data was around 95%.

**Implementing the CNN into an Object Detection Algorithm**

We opted to use the OpenCV cv2.dnn.readNet module as our starting point for object detection. This was due to many sources online pointing to OpenCV as one of the premiere modules for computer vision optimization.

We found a tutorial on PyImageSearch that showed us the basic workings of Object Detection using OpenCV. It was written to classify images already existing on the user’s computer and not for real-time object detection. We made some updates to the script in order to load real-time frames from our computers’ front facing camera.

The general workflow of our process was reading the frame from the webcam, shrinking the image for processing speed, detecting a set of blobs from the image using cv2.dnn.blobFromImage, sending those blobs to our CNN and computing detections, and then plotting bounding boxes around the blobs that yielded a high enough confidence score (in our case 60% or above).

We utilized imutils.video.WebcamVideoStream to efficiently read in webcam frames to the Object detector. Some examples of the labeled webcam images can be seen below. With the current set up, we were able to process at around 25 frames per second.

Unfortunately, after extensive research and many hours of trial and error, we were not able to load our traffic sign model into the OpenCV DNN submodule for use for object detection. The way this is supposed to be done is via freezing a graph (model) in Tensorflow and then utilizing this frozen graph (.pb & .pbtxt files) to load into OpenCV for predicting blobs. Several different attempts were made at freezing the graph but none were successfully read into OpenCV. Various sources on the internet all stated this as a problem between Tensorflow and OpenCV, stating that many of the conversions were not fully built and many manual edits needed to be made to the files.

Due to project due dates, a decision was made to drop the traffic sign CNN from the object detection algorithm and to utilize a pre-made caffe model instead. The purpose of this decision was to show that we trained a CNN model correctly (traffic sign model) and implemented a model into an object detection algorithm (caffe model).

The caffe model was trained on several different objects including: people, chairs, airplanes, bicycles, birds, sofas, etc. It was fairly accurate at detecting several of these objects. Several of the following images show those detections.

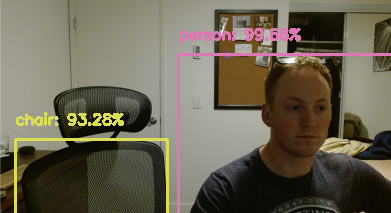


Figure 1 - Caffe Object Detection



Figure 2 - Caffe Object Detection

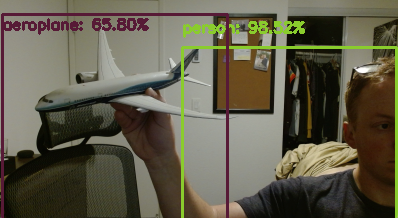
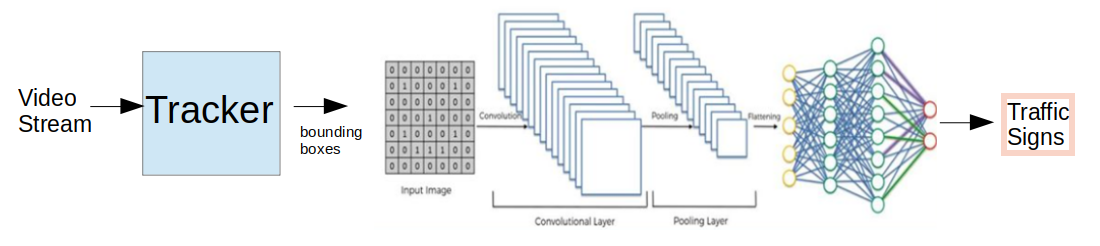


Figure 3 - Caffe Object Detection

## Using a video tracker with an object detector

An alternative approach to classify street signs would be to extract regions of interest (ROI) from an input sequence of video images and then pass those ROIs to a CNN for object classification.. To that end, various tracking algorithms were investigated with the intention of merging the tracking output (bounding boxes) and the object detection (street sign image). This approach leads to the following processing pipeline:



The following tracking algorithms were investigated:

* Standard detect-then-track
* Kernelized correlation filter (KCF)

In the standard “detect-then-track” approach, an object detector is used to extract points or regions of interest from a scene and then correlate these detections over time. This is appropriate where the form and response of what is desired to be tracked is known a priori. Also the dynamics of the objects of interest are usually known as well.

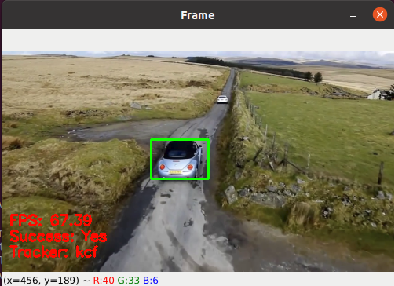
In the KCF approach, the tracker is given a region of interest in one frame and its job is to find that region in the next frame. This is often done in a tight, real-time control loop at data rates between 10-1000 Hz.

Various video sequences were run through either:

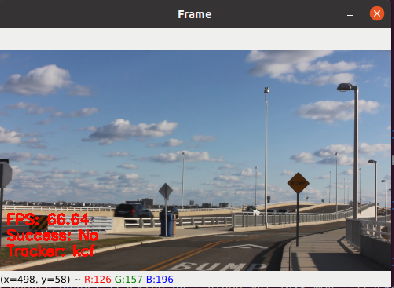
1. a KCF tracker (https://www.pyimagesearch.com/2018/07/30/opencv-object-tracking/)
2. a “detect-then-track” tracker (https://pysource.com/2021/01/28/object-tracking-with-opencv-and-python/)

The results for 4 different cases are discussed below::

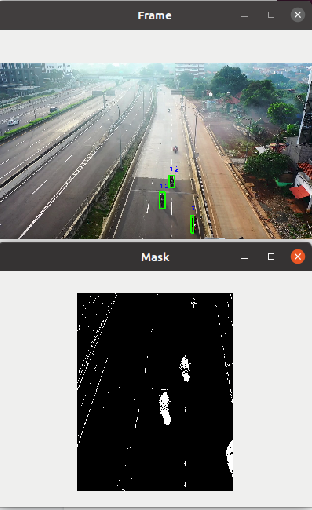
1. Drone following a car (KCF tracker). in the video sequence below a moving car is being followed by an overhead drone. The aspect does not change much and the tracker is able to maintain track. Similarly, the background does not change much from frame-to-frame making the correlation job easier.



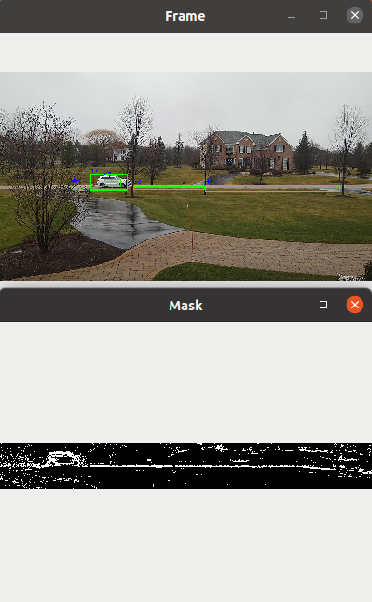
1. Highway scene from stable camera (KCF tracker). The video sequence below was taken manually with the aspect changing rapidly. Hence the cars are not tracked successfully. A tripod-mounted SLR camera was set up ~5 meters on the side of a highway. There were street signs and clouds in the background. As cars streamed by, the tracker was able to track a car for a few frames but as the aspect angle changed dramatically it was difficult for the tracker to correlate successive images.



1. Highway from stable camera (Standard tracker). The video below was obtained from a stable camera on an overpass viewing cars/trucks/motorcycles. In this tracking approach, frame differencing is used to determine moving objects. Pixels showing a difference combined to obtain bounding boxes for detections and an spatial filter was used to eliminate smaller detections (noise). The ‘Mask’ figure below shows the result of a difference between 2 consecutive frames showing larger detections (cars) and smaller ones (noise). The valid bounding box detections were then passed to a correlator which matches detections that are “close enough” on subsequent frames.



1. Car from unstable cell phone (Standard tracker). The video sequence below was taken of a neighborhood street from a hand-held cell phone camera. The white car was tracked successfully across the scene however there were many additional “tracks”. This is due to a combination of the frame differencing for detections coupled with a jittery hand-held camera setup.



Lessons learned from this tracker analysis include:

* Kernelized Correlation Filters (KCF) can operate at high camera data rates
* KCF trackers typically do not track multiple objects
* Dynamic scenes with moving background can confound detection/tracking
* Object occlusion/variation can lead to dropped tracks for the KCF tracker
* Frame differencing for object detection is susceptible to camera jitter
* Multiple hypothesis tracking (MHT) could be used to reduce false tracks

## Results

While training our traffic sign CNN, we were able to obtain a 95% accuracy on the GTSRB dataset.

We were not able to deploy the traffic sign CNN to our object detection framework but we did deploy a standard mobilenetSSD. While using this CNN, we were able to achieve frame rates of 25 fps with fairly good object detection.

## Conclusions

Our team learned a lot from this project. Implementing neural networks into an object detection algorithm is not as straightforward as it sounds. Many of the modules available for python do not readily speak to one another and several kludgy interfaces have been developed. In order to truly implement an interface, one needs to be well versed in the modules at hand for the solution to work.

We did learn a lot about the speed at which these algorithms work. Training our traffic sign neural net took around 30 mins but using the net to predict an image classification only took a fraction of a second. This speed lends itself to the capability of tracking objects within videos due to videos typically running around 30 frames per second.

We also learned a lot about different aspects of video tracking and some of the advantages and drawbacks to various approaches.

## References

GTSRB DataSet

<https://www.kaggle.com/meowmeowmeowmeowmeow/gtsrb-german-traffic-sign/activity>

Converting TF2 to Frozen Graph

<https://giters.com/tensorflow/tensorflow/issues/46740>

How to export a TF2 Keras model to a frozen and optimized graph

<https://medium.com/@sebastingarcaacosta/how-to-export-a-tensorflow-2-x-keras-model-to-a-frozen-and-optimized-graph-39740846d9eb>

Tensorflow Documentation

<https://www.tensorflow.org/guide>

Several TensorFlow Issues

<https://github.com/tensorflow/tensorflow/issues>

Several OpenCV Issues

<https://github.com/opencv/opencv/issues>

Pytorch Tutorials

<https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html>

PyImageSearch (Several Articles)

<https://www.pyimagesearch.com/2017/09/11/object-detection-with-deep-learning-and-opencv/>

PyImageSearch (KCF Object tracking)

<https://www.pyimagesearch.com/2018/07/30/opencv-object-tracking/>

Detect-then-track tracking

https://pysource.com/2021/01/28/object-tracking-with-opencv-and-python/