Traffic Light Identification and Classification

For Non-Invasive Inclusion of Existing Stoplights in Smart City Infrastructure

Alec Resha
Computer Science
CSU: Sacramento
Sacramento, CA U.S.
alecresha@csus.edu

ABSTRACT

Smart Cities are the future, integrating real-time data for optimization, particularly for travel. Optimizing city travel can reduce commute times, cut down on pollution, providing more accurate mapping and suggestions, and many other benefits. The problem being addressed by this project is how to include existing traffic light intersections with smart cities without fully replacing them. The existing problem is that a lot of intersections do not have any way of transmitting their current status, and often do not have an accessible control box. The solution is a non-invasive way to include any intersection given power and an internet connection through collecting the real-time status from a video feed. A Machine Learning model has been used for identifying and classifying stoplights from images, and can be applied to video feeds.

Introduction

The main problem being addresses is regarding future smart cities plans, specifically optimizing stop lights. The end goal is to have real-time data for all traffic intersections so they can eventually be synced to reduce traffic wait times. The direct goal of my senior project is to capture traffic light data in real-time from a raspberry pi which is drop-in compatible with any intersection. This project is to actually capture the traffic data from images. This data will later be encoded and sent to a central database, but the focus here is just data collection first.

The approach for this project was a modification of standard object detection algorithms. Most object detection models are for classifying a large array of objects from images, usually based on a COCO dataset. The issue with these general models is that they will classify everything which is slower than we need, and they classify all stoplights together rather than red/yellow/green. I utilized the algorithm called YOLOv5 for this purpose. My dataset was of annotated images that I had to convert to a YOLO data format.

This project was done alone so all contributions are mine.

- Organize and interpret dataset
- Create algorithm for converting data and folders to YOLO format
- Create an index based label encoder
- Make and test YOLO models to determine best one for project
- Vary parameters, primarily number of images and epochs

The remainder of this paper will further discuss the problem, each part of the algorithm, an analysis of the results, a brief discussion on related works, and a conclusion.

Problem Formulation

Precisely define problem

This section will further define the specific focus of this project, including the goals and the dataset that was used for training.

My senior project is focused on being able to get trafic signal data in near-real time, about every 100ms was requested, and encode it in a standardized format. We will be capturing the data through a camera attached to a raspberry pi which will send a snapshot to the ML model every 100ms. This model is used to interpret the images.

The main problem that is here is that there are not any public machine learning models for both identifying and classifying stoplights. There are plenty of object detection models that can find stoplights in an image, and others for classifying an image of just a stoplight. We were unable to find any for identifying and classifying stoplights at the same time, and in an efficient way so this project was built specifically for this purpose.

The functional inputs for this project will be images from trafic intersections which include trafic signals. The output is a list of stoplights, including the classification and location in the image.

System/Algorithm Design

There are multiple modules at play in this project : converting the dataset, encoding all the labels(0-13), passing each image into the ML model, and interpreting the output.

The YOLOv5 model group was decided on primarily because of its size. There are not many trainable object detection models due to the complexity that they require. The other model that I looked into a good deal was a Faster RCNN model. The main reason for going with YOLOv5 over Faster RCNN was the speed that they process images. Since we need near real-time data, our overall project set a contraint of 100ms for capturing, encoding, encrypting, and sending the traffic light data. The YOLO model achieves a speed of around 20ms to 40ms for processing an image whereas the Faster RCNN model description says it can do it in roughly 1 second, which is far too slow for our purposes.

Dataset Conversion

The dataset did not come in a format that was compatible with most models, including a YOLO model. All of the labels and annotations were listed in a YAML file in the following format.

Image with no box :

- boxes : []

path : ./rgb/train/2017-02-03-11-44-56_los_altos_mountain_view_traffic_lights_bag/207374.jpg

Image with 3 boxes :

- boxes:

  - {label: Yellow, occluded: true, x_max: 615.75, x_min: 610.625, y_max: 358.625,

    y_min: 351.5}

  - {label: Yellow, occluded: false, x_max: 638.125, x_min: 633.875, y_max: 351.0,

    y_min: 342.25}

  - {label: Yellow, occluded: false, x_max: 655.0, x_min: 649.5, y_max: 360.75, y_min: 350.375}

path: ./rgb/train/2017-02-03-11-44-56_los_altos_mountain_view_traffic_lights_bag/207386.png

The problem with this dataset was that it is not compatible with any ML models. The required format is two folders, one labelled images, the other named labels. Each image file must have a corresponding txt file with the same name (with a different extension). An example of this would be images/img1.jpg and labels/img1.txt. The main conversion here is to copy images out of their path into the images folder, and create a new txt file for the relevant labels.

The required txt file also has a very different data format. While the dataset came in the form {label, occluded, x_max, x_min, y_max, y_min}, the txt file must have each line be in the form « labelIndex x_center y_center boxwidth boxheight ». The coordinates in the dataset also are in pixels, while the coordinates needed for the model are between 0 and 1, relative to the img size. To find this, the algorithm is as follows :

- avg_x = (x_max + x_min) / 2
- avg_y = (y_max + y_min) / 2
- box height = (y_max − y_min)
- box width = (x_max − x_min)
- Divide avg_x and box witdth by imgwidth
- Divide avg_y and box height by imgheight
- Encode label with index-based encoding (see next module)
- write row to .txt file
- repeat for all boxes in label record for image

Label Encoding

The unique labels for the entire dataset are collected into a single list then assigned indices. When outputting the text to the txt file (from above), the label is encoded through the list then the index is placed in the txt file.

Config File

A config file is needed for the YOLO model. This model has to contain : a path to the dataset folder, the relative path to training, val, and test folders, the number of labels, and a list of all the labels in increasing index form.

In this project, the path is ‘/dataset/’, the training path is ‘train/’, the val path is ‘val/’, and the test path is ‘test/’. The number of labels is 13, and the labels are listed in the same orderas they were encoded. The order is listed here :

```
{'Yellow': 0, 'RedLeft': 1, 'Red': 2,
'GreenLeft': 3, 'Green': 4, 'off': 5,
'GreenRight': 6, 'GreenStraight': 7,
'GreenStraightRight': 8, 'RedRight': 9,
'RedStraight': 10, 'RedStraightLeft': 11,
'GreenStraightLeft': 12}
```

YOLO Model

The model takes the above mentiond config file as an argument, as well as the image resolution, the number of epochs, what weights to use, and what file name to use. The weights argument defines what version of the YOLOv5 model is used. In the case of this project I used the YOLOv5s as my final model, but also tested YOLOv5m and YOLOv5L. These are both more accurate than YOLOv5s, but they are much slower and take much longer to train. This is done through the command :

python train.py --img (img resolution to use) –epochs (number of epochs to use) – source (config file path) –weights (model to use).pt –name

For testing the model, the following command is used :

python test.py –weights (path to weights generated by above command) –img (img resolution to use) –conf (confidence used to accept a result) –source (path to images/videos or 0 for webcam.

The YOLOv5 model works by dividing an image into a grid and trying to detect an image in each grid square, and will make recursively smaller squares to find the borders of a full object. The different sizes (YOLOv5s, YOLOv5m, YOLOv5L, etc) all work the same way, just with more layers.

For most use cases, the creators of the YOLOv5 model recommend the YOLOv5s model since it is lightweight and fast, the higher tier models can be more accurate and with fewer training epochs and data, but take longer to classify images and the only real benefits come when trying to classify a large number of different kinds of objects with the same model.

Experimental Evaluation

Methodology

The data used was from annotated still images from a dashcam in mutliple locations. All visible stoplights were annotated in the dataset. The data was manually split into separate train/val/test folders based on a random selection from the overall dataset.

The primary changes that were tested were different YOLOv5 models, different number of training images, and different number of epochs for training.. Three different models were tested, namely YOLOv5s, YOLOv5n, and YOLOv5l. YOLOv5s was multiple times faster both for training and for evaluating each image when compared to the YOLOv5n and YOLOv5l models. It was also nearly as accurate, so the overall speed outweighed the marginal benefits of the larger models.

In regards to training images and epochs, adding more training data was not useful unless there was a sufficiently large number of epochs (>50). When trained on 3 epochs for all 5,000 images in the dataset, the training took upwards of 13 hours and could only barely detect even some of the stoplights. Conversely, 500 images with 200 training epochs took 11 hours to train, but is extremely accurate in both detecting and classifying stoplights. The 500 image and 200 epoch training is the one that I plan to use going forward, eventually training it on night images as well (since none were included in the data set).

Results

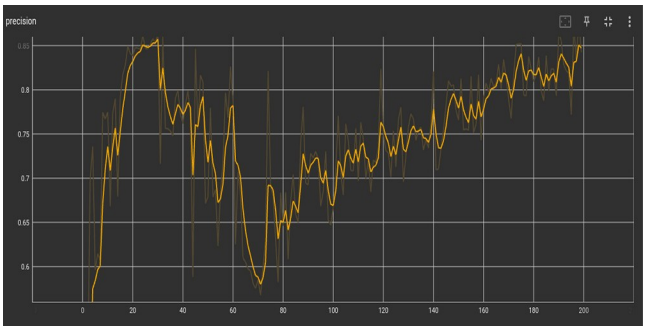

Figure 1: Precision vs Epoch # for YOLOv5s with 500 images and 200 epochs

As shown in Figure 1 above, of the precision for this training, the precision got to it's max of 0.8645 around 30 epochs then dropped to about a 0.6 precision before increasing towards 0.86 again. The highest precision was an 0.8645 which was the best weight saved.

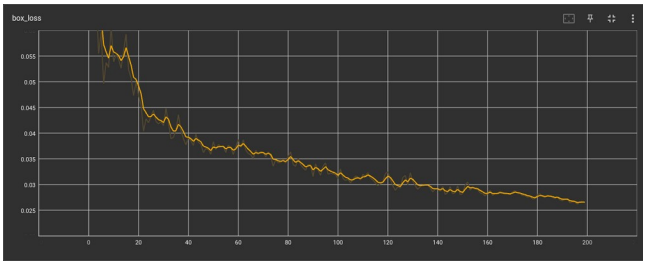

Figure 2: Box/val los vs epoch # for YOLOv5s with 500 images and 200 epochs.

As shown in figure 2, the loss was continually decreasing as the model progressed, indicating improvement in the model.

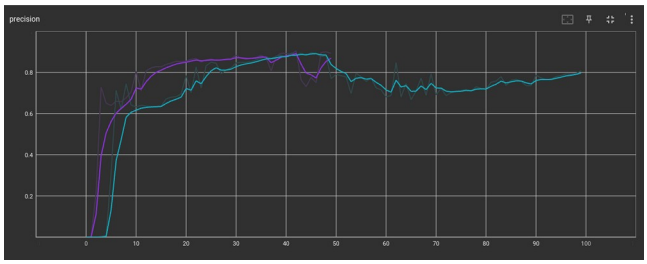

Figure 3: Precision for 50 epochs with 1000 images on YOLOv5n (purple) and 100 epochs with 500 images YOLOv5s (blue)

When I experimented with increasing the number of images and decreasing the number of epochs, the values were similar. In this case, having more images lead to a faster increase in the precision, but ultimately did not lead to a particularly more precise model than with 500 images with 100 epochs. The precision for 500 images and 200 epochs was not shown on the graph due to scaling making the graph hard to read, but all three models came out to similar accuracies. With more images and more epochs the model could potentially become more precise, but at the expensive of a very long training time. The run with 50 epochs and 1000 images took several hours longer than another run with the same number ofepochs and images but with the YOLOv5s. This equivalent run on YOLOv5s did not save correctly unfortunately so the results were lost.

Another experiment was run with 5000 images and 3 epochs, the precision was only slightly above 0.5 despite 14 hours of training.

Comparing speed and accuracy, the 200 epoch model with 500 images took a long time to train, but has been slightly more accurate than the other models that were trained, though all were accurate with a suffcient number of epochs and images.

Related Work

There are not too many projects that address the specific problem that I needed to solve in this project. A standard object detection model (such as the ones that are built into YOLOv5) do not differentiate the different stoplight phases, it just detects that a stoplight exists. After researching for a pre-trained model specifically for identifying and classifying stoplights, I wasn't able to find anything that works beyond a simple CNN for an image of just a stoplight, which is not helpful for the context of this problem

Conclusion

Summarize results anc conclusions

With sufficient training, many of the YOLOv5 models achieve the same degree of precision. That being said, the training must include a minimum of around 10-20 epochs in order for the model to become confident, whether this is with 500 images or 5000.

Since all the models can achieve similar degrees of accuracy, YOLOv5s is the most efficient model for this use. This is because it can process images faster than the other YOLOv5 models due to its smaller size.

Work Division

All work was done by me (Alec Resha) as the project was done alone.

Learning Experience

The biggest thing that I learned, was something that was reinforced by trying to train the model : more data is not always the solution. When trying to train the data, I initially had it at 3 epochs and kept increasing the data until I was using 5000 images, and the results were still not particularly good (precision of about 0.65). After that, I moved to using 500 images and 50, 100, then 200 epochs, and the model went much better (see figures 1, 2, and 3). My best model so far took 11 hours to train, but has an overall precision of around 0.95. I intend to continue training the model with more data over winter break since this project may get real world use, but I will likely pay for an ec2 instance with vpu support rather than have my laptop run overnight.

The other thing I learned was how the object identification algorithm works. This was not covered in the ML class, but my project required it so I did a lot of research regarding it. It is a very interesting way of breaking up an image to search for objects, and the model I chose to use does it very quickly as well. The data format is also an interesting part of the model as it defines a center and width and height of the bounding box rather than x and y coordinates.