Object Detection

Shane Lafollette

Problem:

- New tech today in the automotive industry implements computer vision to help make driving safer and aid the driver in error prone situations. From lane assist, active cruise, active braking, and driver monitoring, to self-driving vehicles, computer vision is what has allowed these technologies to become standard across the market.
  - What machine learning models can work well with computer vision?
  - Can we create a small model that does the basic functions and makes predictions like these technologies?
  - Can we think of ways to improve upon these technologies and the models they use?

The Data:

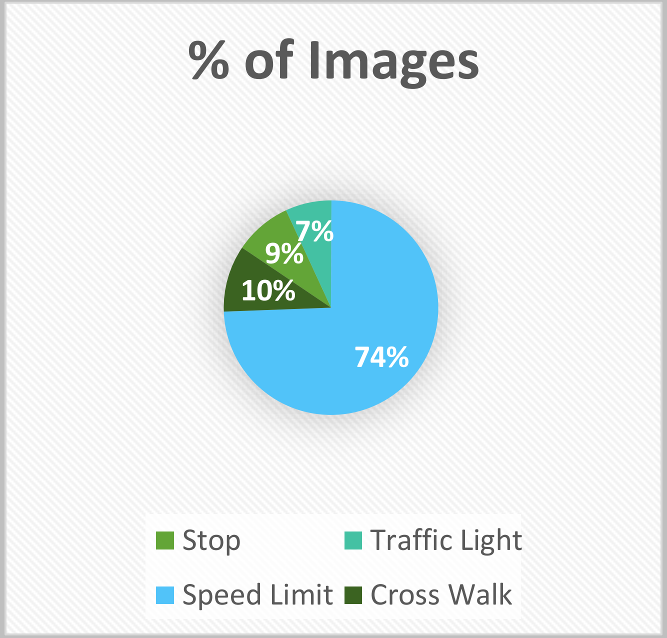
- One of the first problems of the self-driving car and computer vision was building a ML model that could interpret the vehicles' surroundings and perform reactions accordingly. A small part of this is road sign detection. Road sign data from Kaggle was acquired for use to create and train a model that can detect and classify signs and traffic lights. Building this model will help us relate to how the new technologies that aid drivers today work as well, as the same computer vision techniques would be used in those to detect the objects in question.

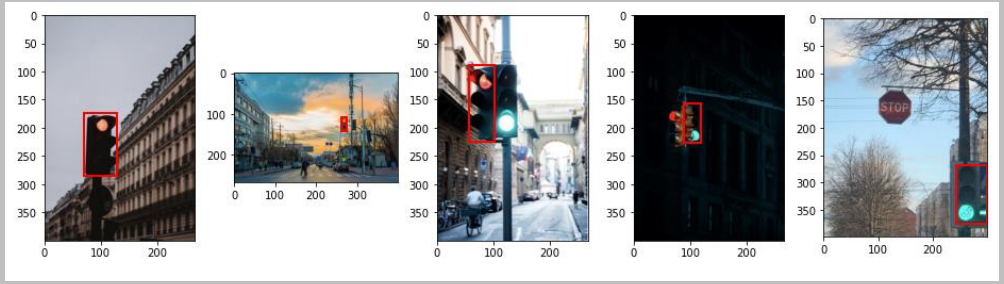
- https://www.kaggle.com/datasets/andrewmvd/road-sign-detection

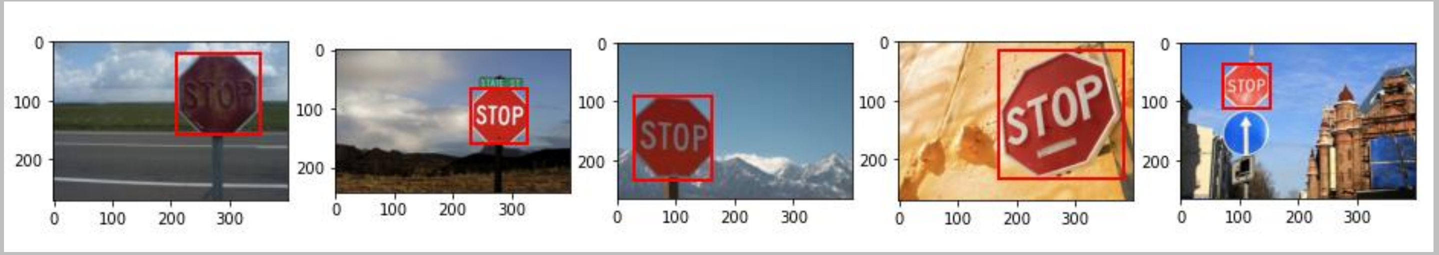
Data EDA:

The dataset is 877 images of four classes. Stop sign, Speed limit sign, Cross walk sign, and traffic light images have an XML file grouped with them that lists class label and bounding box coordinates. The images were of different sizes of height and width averaging around 310x386. The dataset was also very unbalanced as 74% of the images are Speed limit signs. The chart on the left breaks down images by class.

The data was loaded into a data frame and the images were viewed with the bounding box overlayed on the image to make sure the coordinate data was correct. A couple issues noticed were that the dataset came with only one bounding box per image even if several target objects were in the image and several images also had multiple class objects but were only labeled with one class. This can be corrected by creating new bounding boxes and labels for the unbounded objects in the data. Before doing this, lets create a model and see how it predicts on the images and bounding boxes we have.



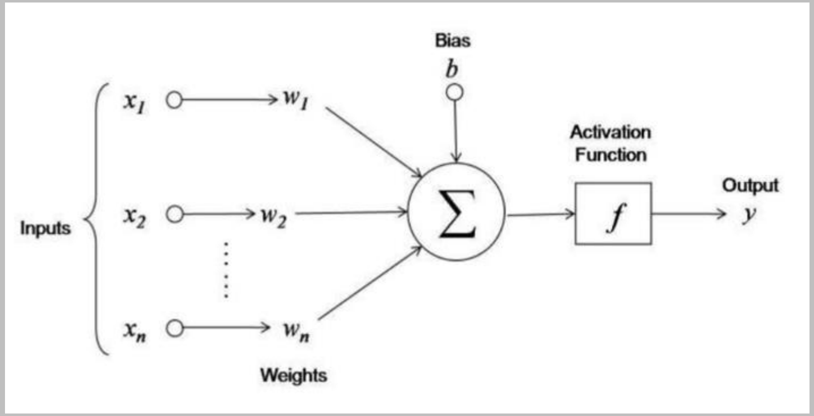




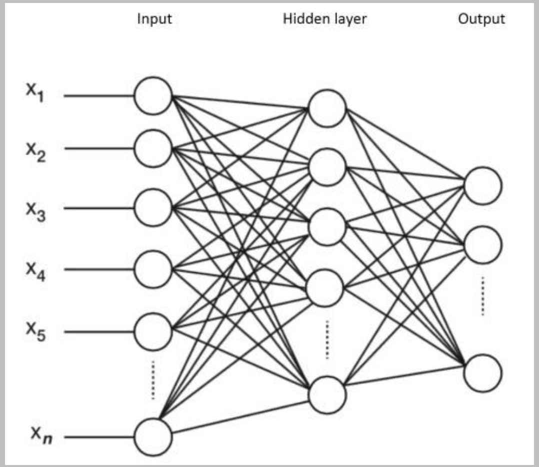
ML models for Object Detection:

The ML model most used for image processing and object detections is the Neural network. These models are architectures with many layers, taking an array from the pixel data as an input and using neurons (filters) in each layer to create feature maps that may be able to find the important characteristics of the target. These neurons apply weights to the data, then passes data through an activation function that can either act positively or negatively upon the data to help identify importance. After this process happens in the layer, it is passed to the next where it will go through the same process. After going through these calculations, the model will predict an output. When training, neural networks use backpropagation to adjust the weights applied in each layer to find the best model for the data. That's a quick review of how the model operates.

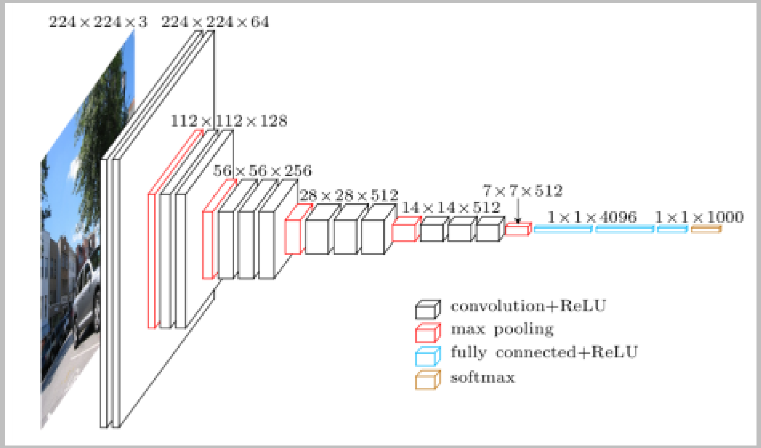
Neuron Example



Small Neural Net with single hidden layer



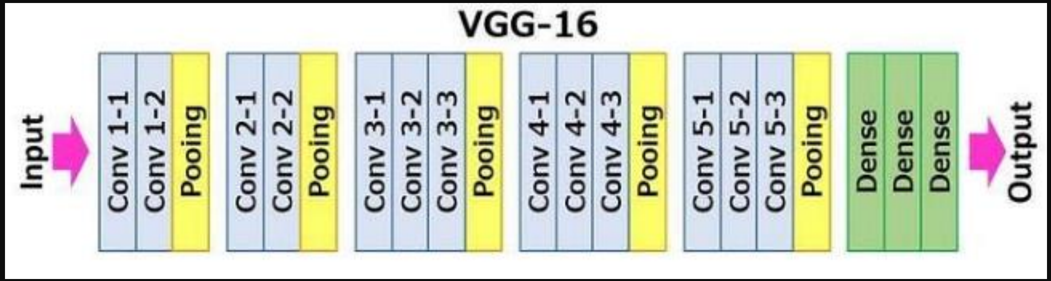
Full Neural Net Example


$224 \times 224 \times 3$
$224 \times 224 \times 64$
$112 \times 112 \times 128$
$56 \times 56 \times 256$
$28 \times 28 \times 512$
$14 \times 14 \times 512$
$7 \times 7 \times 512$
$1 \times 1 \times 4096$
$1 \times 1 \times 1000$
convolution+ReLU
max pooling
fully connected+ReLU
softmax

Model 1: VGG-16 Neural Net

Model Architecture

- 13 convolutional layers
- 5 pooling layers
- Can use model pre-trained on ImageNet
- Accepts input size of 224 x 224 x 3
- Neuron counts of 64,128,256,and 512 are used
- Kernal size is 3 x 3
- Strides of 1 on conv. Layers and 2 on pooling layers
- Last 3 dense layers can be dropped to add custom layers for your data

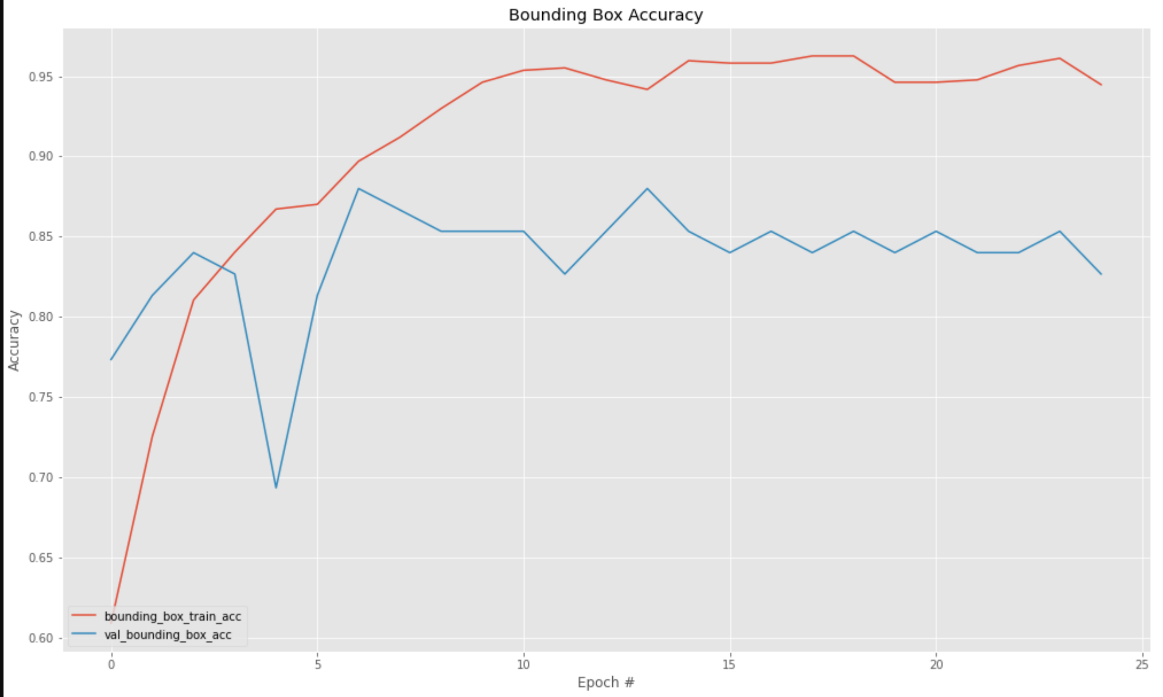


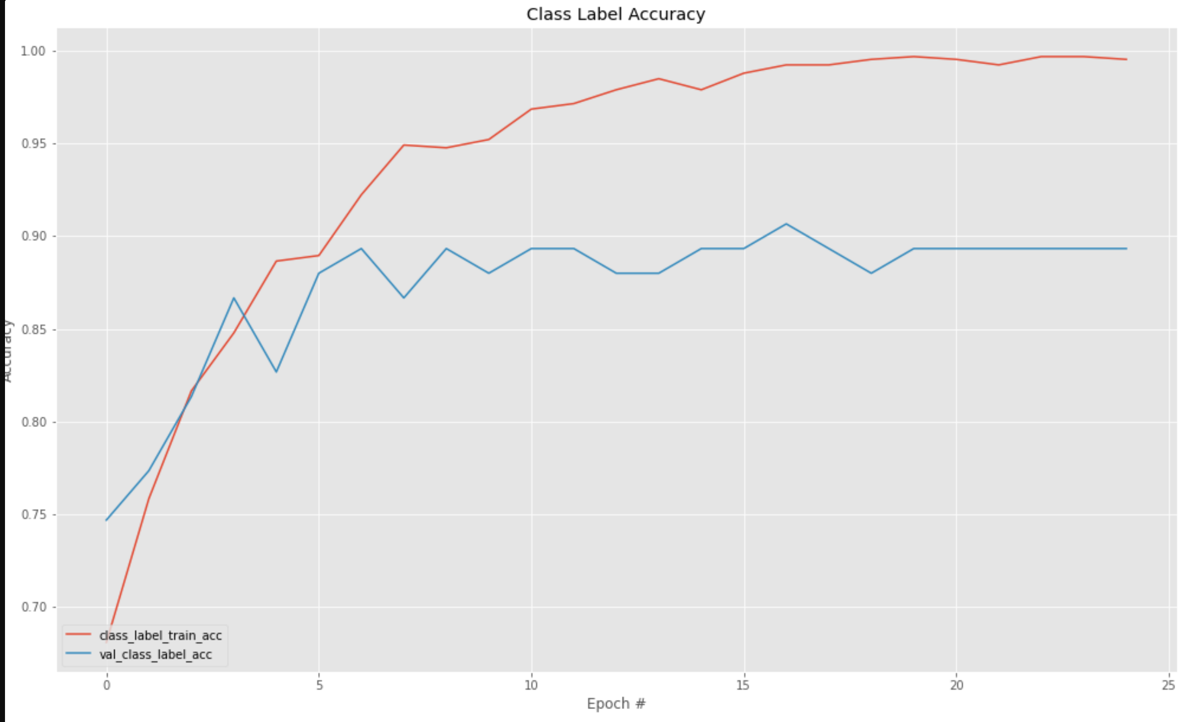
VGG-16 Parameters:

- Imported pre-trained model
- Dropped Top (Dense) layers
- Froze all pre-trained weights
- Resized images and boxes to 224 x224 x 3 input
- Normalized pixel data
- Transformed data into arrays
- Split data into train (76 %), validation (9%), and testing (15%)sets
- Created Neural Net model to take VGG output as an input, pass it through the network of dense and dropout layers and predict the outputs of class label and bounding box location
- These Dense layers were trainable.
- Optimizer chosen was ADAM
- Learning rate was 0.0001
- Loss functions were categorical cross entropy for labels and MSE for bounding boxes
- Batch size set at 16
- Epochs were set at 50

VGG-16 Metrics:

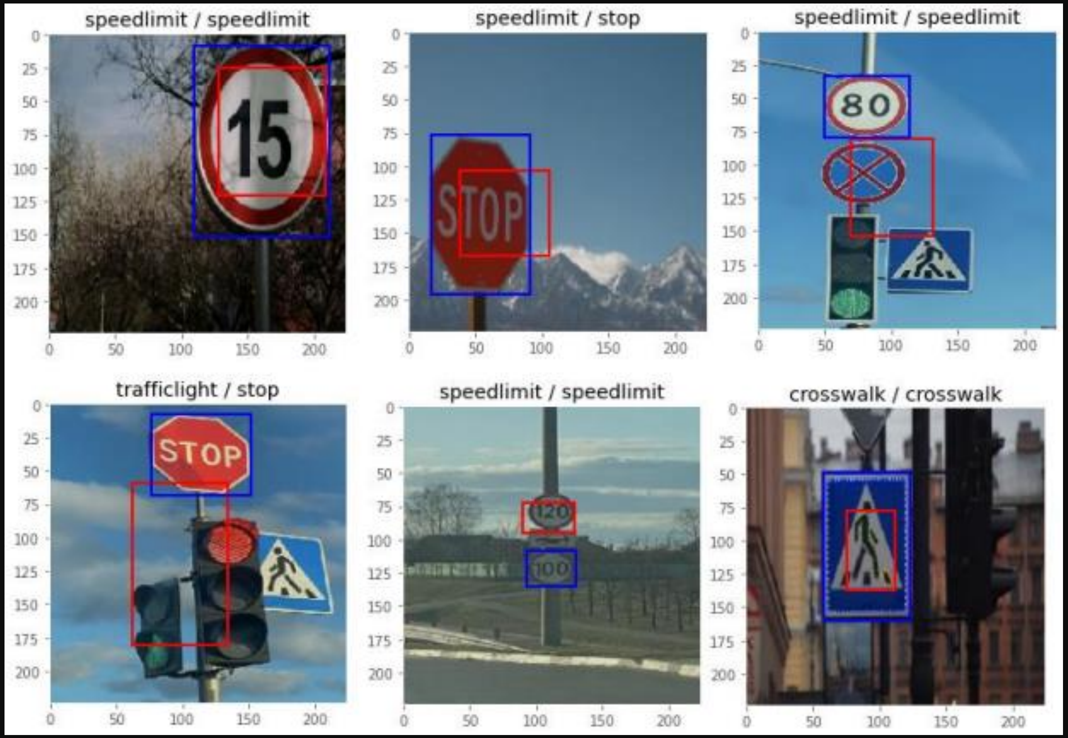
- The model created over 31 million parameters.
- After 25 epochs, the model had over 99% accuracy on class labels and 96% on bounding boxes
- The validation accuracy was 89% on labels and 85% on boxes
- Small tuning adjustments were tried before deciding on learning rate, batch size





VGG Model Predictions:

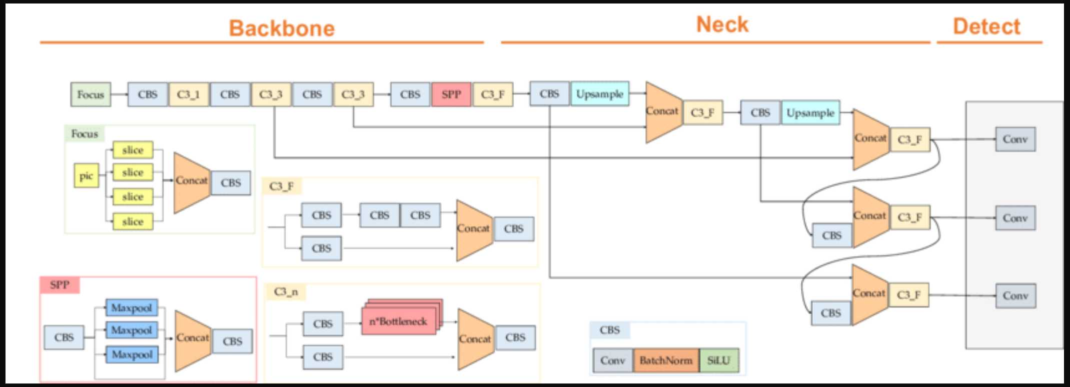
- The model accuracy on the test set was 83%.
- There were some issues with this model and the dataset :
    - The model had trouble predicting bounding boxes and class labels on images with multiple signs
    - While the bounding boxes may have been on the correct sign, several would cut off edges and pieces of the object
    - We would like to create a model that predicts several bounding boxes if needed
    - We need better prediction of bounding box size

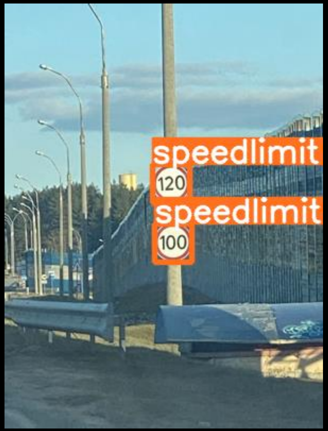


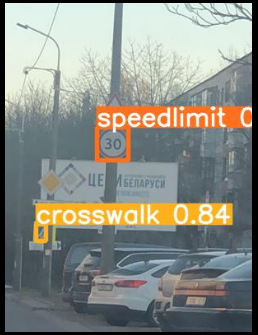
Model 2: YOLOv5 Neural Net

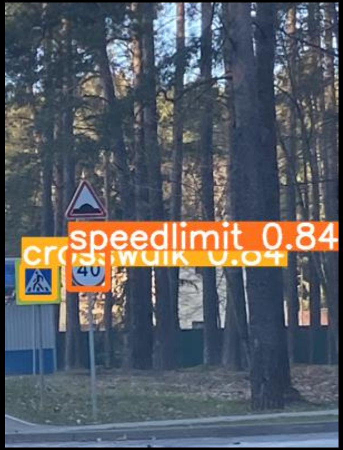
- **Model Architecture**

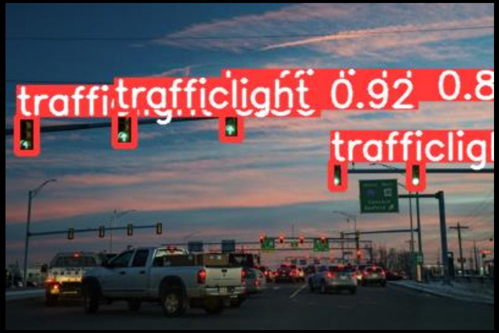
- The model is very dense, with the backbone of the model creating the feature maps through convolutional layers, pooling layers, concatenation, batch normalization and more
- The neck of model combines the info from different maps
- The head of the model detects the objects
- Breaks up images into large, medium, and small grids to search out objects
- Will apply bounding box to any object with probability greater than threshold
- Can be imported pre-trained
- It will output the bounding box along with the class label and class probability over the image
- Uses mAP (mean average precision) and IoU (intersection over union) as metrics of accuracy





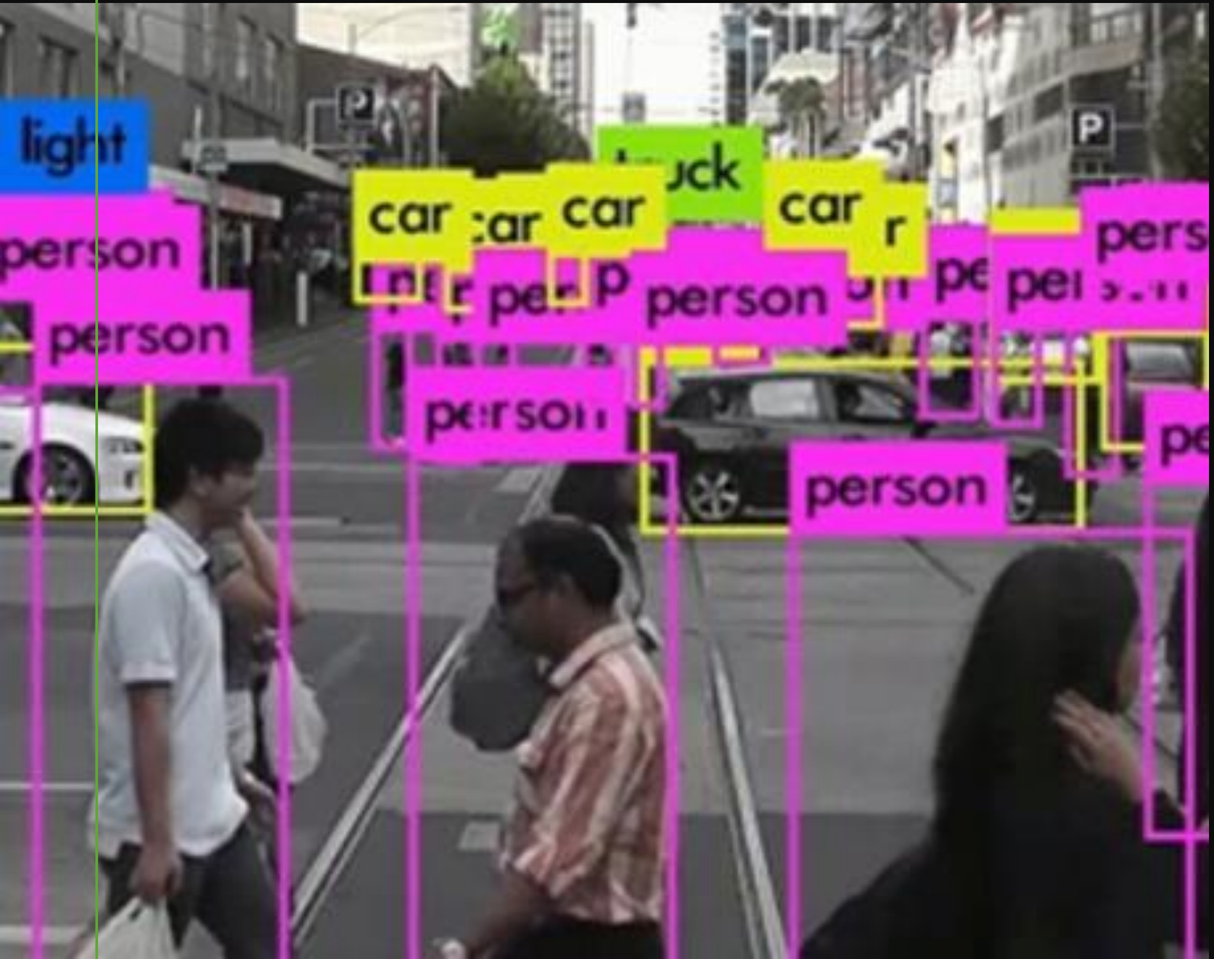






YOLOv5 Model Metrics

- The YOLOv5s (small) model was imported with pre-trained weights
- Trained model on our dataset
- Batch size was set to 32
- Image size was set to yolo standard 640 x 640
- Initiated training for 100 epochs but early stoppage happened at epoch 56 due to no substantial gain in accuracy
- The model showed accuracy of over 96% on our dataset



Summary:

- The models created have great accuracy and the YOLO model implemented works very well with the dataset used. These models can be adjusted, restructured, and retrained in thousands of way to make the models more precise and accurate. The YOLO model we used may be more advanced than some of the computer vision models used by automotive tech today. Adaptive cruise looks for objects within several hundred feet in front of the vehicle, not having to classify them, but just detect them.
- These models have given a good idea of how accurate they can be with little tuning and adjustment. As there is a multitude of hyperparameters and architecture that can be changed, at times we may not need all the performance that comes with the pre-trained models that can be imported. Some small problems can just be solved with a small network of a couple layers.