COMP9517 Project - Vehicles Detection

1st Changyong Yang *z5172587* | 2nd Harry Kha *z5161310* | 3rd Henry Fang *z5163370* | 4th Minh Khai Tran *z5168080* | 5th Tiankuang Zhang *z5236826*

I. Introduction

A. Our understanding of the task specification

- Auto-driving car is a theme that you cannot avoid these days on the website, newspaper, magazines, etc. Our ears are flooded with names like Elon Musk and his superstar company Tesla. We also heard rumors that Apple might produce its own self-driving car.
- The self-driving technology is so popular not only because it can relieve the burden of driving from our human hands, but also it fully utilises machine learning algorithms to avoid human error that often result in disastrous car accidents. However, this exciting technology can be realised only if a car can successfully detect other vehicles. Not only must it detect other vehicles, it must be able to detect their relative velocity and corresponding positions.
- Besides the detection of vehicles and estimation of their relative velocity and positions, another big challenge is the lane detection. Lane detection is essential and crucial to safety especially "when high volumes of cars are driving on narrow roads at fast speed. In complex situations like this, road colour extraction and texture detection as well as road boundary and lane marking are the main perceptual clues of human driving." [1].
- The tasks of this group project are:
  1) Develop a model that detects vehicles of given image;
  2) Estimate relative position of vehicles detected to image plane;
  3) Calculate relative velocity of detected vehicles;
  4) Detect lane.

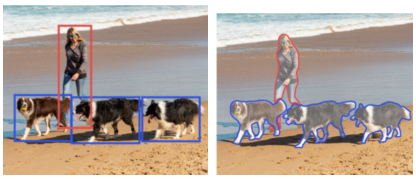
B. A brief discovery of Data sets

- The data set for velocity detection uses "2017 TuSimple Velocity Estimation Challenge" [2]. It includes 1074 training data and 269 testing data. Each data is a 2-second clip that is formed by 40 frames. For each data's 40^{th} frame, we are also provided with the annotation file that records the bounding box coordinates, relative velocity as well as relative positions information on all cars detected.
- The data set for lane detection uses "2017 TuSimple Lane Detection Challenge" [3]. The training dataset includes 3626 video clips and annotated frames and the testing dataset includes 2782 video clips. Also, we are provided with a label data JSON file in which each JSON line in 'label_data_(date).json' is the label data for the last (20^{th}) frame of this clip.

II. Literature review

A. Car Detection related Literature review

- **Image Segmentation Object Detection**:
  - **Image Segmentation** Image segmentation refers to the extraction of specific image features like colours, shapes, and contours [4]. Using segmentation techniques, we are able to separate 'useless' information from 'useful' information [5]. Using image segmentation we would be able to extract certain features of an object and remove background elements and bring forward foreground elements which we can use to highlight objects of interest in an image [5].
  - **Object detection** Object detection is a combination of image segmentation and object localization which basically means a task involves drawing a bounding box around one or more object and assign them a class label [6].



Left: object detection [5], Right: object segmentation [5]

- **Machine Learning and Deep Learning**:
  - The implementation of machine learning algorithms for object detection has been extensively used in literature in the past due to the fact that it is able to efficiently and accurately detect and classify objects within an image. For this group project, an efficient and accurate object detection algorithm that is able to classify and identify cars in an image is required.
  - The work of Gavrila and Philomin [7] proposes an efficient and simple shape-based object detection method that is based on the distance transforms. This method uses a template hierarchy to capture the variety of object shapes in order to classify and identify objects. While this method is successful at object detection, it requires pre-processing of the image beforehand which hinders the speed. Nevertheless, this method demonstrates the ability for computers to autonomously identify cars within an image frame.

– Finally, Alex et al. [8] introduces a large deep learning convolutional neural network architecture that is able to learn how to detect up to 1000 different objects with an overall accuracy of 85%. As opposed to previous methods, this new model is able to generalise features of objects more efficiently and accurately which allows it to be optimal in a more realistic setting, where smaller neural networks struggle.

B. Distance Estimation related Literature review

- There are various methods to estimate vehicle distance applied by other researchers. For example, Liu et al. [9] used MATLAB software provided by [10]to calibrate the camera parameters and proposed a method that uses a single-point laser rangefinder to measure the distance. Bougharriou et al. [11] also calibrated the camera and proposed the method that combines extraction of Point of Interest (PoI), roads detection and vanishing point intersection to determine vehicles' distance. Finally, Wu et al. [12] used the shadow underneath the front vehicle to identify its position, and then utilised a neurofuzzy network to estimate the real distance.
- In our method, since camera parameters are given, we assume the constant actual width of all vehicles, choose to apply triangle similarity between world coordinates and image coordinates, but also experiment advanced regression techniques and machine learning methods to further improve the accuracy of this simple method.

C. Velocity Estimation related Literature review

- In relation to velocity estimation, there are a variety of methods that have been used widely for vehicle safety regulation. For example, there have been GPS-based velocity calculation technology that have been fitted in cars for the purpose of recording driving accidents [13]. There was also a deep learning approach which made use of a dataset consisting of video clips with velocity labels for each time frame [14]. These approaches were less traditional and made use of existing technology like satellite technology and deep learning models.
- Ultimately, the approach we decided to go for was to go with a more traditional, formula-based method – where relative velocity of cars would be calculated by making use of our distance calculations in Task 1 to determine the relative change in distance of vehicles from the camera standpoint, and then applying the simple formula: velocity = distance / time [15].

D. Lane Detection related Literature review

- Canny Edge Detection is algorithm used for edge detection in an image.
- Convolutional Neural Networks (CNN) is a Deep Learning algorithm which can take in an input image, assign importance to various aspects/objects in the image and be able to differentiate one from the other [16].
- Hough Transform is used to detect any shape in an image, if you can represent that shape in mathematical form [17].

III. Method Choice and Implementation

A. Methods Implemented

- Car Detection Model Choice:
  - **Car detection using Yolo**:
    - How it works: The yolo algorithm is based on regression. It is able to predict classes and bounding boxes for the whole image in a single run of the algorithm, instead of selecting certain parts of an image. The main advantage of this feature over its competitors is that it can perform high-accuracy detection in real time with a high frame per second [18].

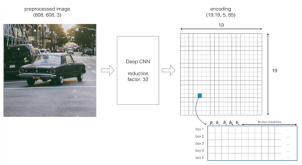


Figure 1: Splitting the image into $19 * 19$ cells, see ref [18]

- The YOLO algorithm is very simple in nature. The algorithm can be broken down into 3 main steps:
  1) **Resize the image**:Before the input image is passed into the model, it is first resized into the optimal shape of $448 * 448$. Once the image has been resized, it is then passed onto the convolutional neural network.
  2) **Run convolutional neural network**: During the pass of the forward propagation of YOLO, it determines the probability of each cell containing a certain class and predicts B number of bounding boxes. Each grid cell is only able to predict one class regardless of the number of bounding boxes that it finds [19]. This neural network architecture was inspired by the GoogLeNet model for the image classification [20]. More specifically, the network consists of 24 convolutional layers connected to 2 fully connected layers with the full network shown below in Figure 2.

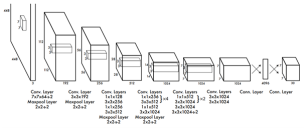


Figure 2: Fully connected Convolutional Network layers of YOLO [18]

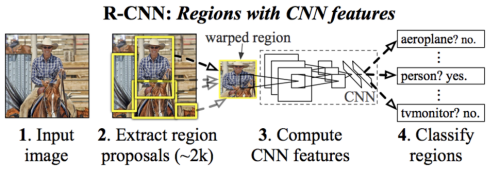
  3) **Non-max Suppression**:The YOLO algorithm predicts multiple bounding boxes per grid cell. This is done through non-maximum suppression, which performs Intersection over Union on the bounding boxes. The YOLO algorithm

continues this process with the remaining bounding boxes with the next highest class probability until left with different bounding boxes.

* **Reason for choice** : Although the requirements of this group project do not require real-time detection of objects, the YOLO algorithm will still be implemented due to YOLO's ability to generalise features of multiple objects simultaneously. In our case of car detection, this is useful as there may be many cars in a single frame image, each with different body shapes, sizes, and colours.

– **Car detection using RCNN**:

* How it works: R-CNN or regions with CNN, is a convolutional neural network-based model that combines selective search algorithm to extract region proposals [21].



see ref [21].

* Steps Taken: Our implementation was based on a 4-part series on deep learning and object detection tutorial by Adrian Rosebrock [22].

1) **Region Proposals Region of Interest**: In our implementation, we made use of a selective search algorithm provided by Rosebrock in his tutorial. This algorithm generates a number of proposal rectangles generated by OpenCV's Selective Search Segmentation function [23]. Conflicts of interest refer to region proposals that overlap with the ground truth bbox. In the code we have a function called computeIoU() which takes in two rectangles and spits out an output value corresponding to the ratio in which box A overlaps box B – intersection of union [23]. Using this value and a threshold of 0.7, we can separate positive classifications of vehicles from negative classifications of vehicles.
2) **Run convolutional neural network:** Rosebrock's tutorial uses the MobileNetV2 architecture fine-tuned with specific parameters that can be found in the tutorial [23].
3) **Non-max suppression:** Refer to 3) in Car Detection using Yolo.

* **Reason for choice**:In terms of why R-CNN would be a good candidate for vehicle detection is that it is a clever approach in that it is able to effectively extract only relevant region proposals of an image for positive and negative classification. The main downside to the R-CNN approach though is the computation time especially for images in our dataset which were $1280 * 760$, which meant that proposal boxes generated would take a really long time.

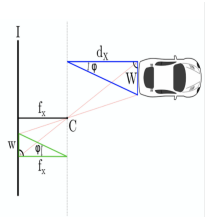
– **Car detection using Haar Cascade technique**: From the given velocity dataset and based on provided ground truth, we are able to extract two set of images including car images and no-car images. The approach is shown as below:

* **Neg and Pos folders:** After extracting two set of images (car and no-car images), we put them into two pos and neg folders.
* **Create descriptor files:** We need to create descriptor files for both pos and neg folders such as pos.txt neg.text.
* **Create vec file using OpenCV:** We need to create a vec file before we can actually train our model by running this command 'opencv_createsamples -info pos.text -num num_samples -w desire_width -h desire_height -vec pos.vec'
* **Train haar cascade model:** Before train our model, we need to create a directory named cascade and run this command 'opencv_traincascade -data cascade/ vec pos.vec -bg neg.txt -numPos num_cars -numNeg num_nocars -numStages 10 -w 24 -h 24'
* **Load model:** Once we complete the training part, we can load our model from jupyter notebook by using OpenCV2 'cascade = cv2.CascadeClassifier('./cascade.xml')'

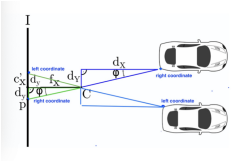
- Car Position Estimation Choice:

– **Position Estimation: constant width based**: this approach mainly deploys the method of triangle similarity and then carefully select a constant value W to estimate position values.
– **Triangle Similarity**: Given the camera calibration parameters, which are $fx = 714.1526px$, $fy = 710.3725px$, $cx = 713.85px$, $cy = 327px$ and camera height is $1.8m$, we are able to apply triangle similarity to calculate the distance from each vehicle ahead to the camera.In our method, we tried the approach "distance from width", **which is based on the source [24]**
– **Distance from width:** In this method, it is assumed that vehicles are of the same width(W). As shown in the picture, this method employs camera parameter fx. Based on the triangle similarity between two triangles, that is $\frac{dX}{W} = \frac{fx}{pixel\_width}$ where $pixel\_width$ is the width of the bounding box (pixel width). Then we can derive distance dx as $\frac{fx*W}{pixel\_width}$. Therefore, the only unknown parameter is the constant W that we need to assume. **In our method, we assume the constant width would be the average calculated width from the training set, which turns out to be** $2.156m$**. More details about how we obtain this**

value as well as its performance can be found in the section of experiments and results. Below image is cited from [24, p. 20].



- **Lateral distance:** To calculate the second value of cars' position, we apply triangle similarity again. The calculated distance dx is needed here. For dy, as depicted in the graph, when the vehicle is to the left of the camera we should calculate dy as the 'left coordinate minus cx', otherwise it should be 'right coordinate minus cx'. Since the 'Distance from width' method gives smaller error, we choose to employ dx calculated by this method and then calculate dY as $dY = \frac{dy*dX}{fx}$. After the experiment, the average error for the second position value in the training set is 1.14 and 1.2 in the test set. Below image is cited from [24, p. 23].



- In the above two methods, we assume the width of the car is constant. This constraint is not ideal given that the vehicles can appear in random positions in the image and car widths are not the same for different cars. Therefore, we further consider creating custom parameters for each vehicle like the one below.
- **Position Estimation: Linear Regression Model based**:
  - Linear Regression approach takes features from an image and predict the car width of a car. Then it uses triangle similarity method mentioned above to predict distance from width and lateral distance.
  - Reason of Choice: since triangle similarity can provide a reasonable estimation of distance based on bbox information, it indicates that there exist some linear relationship between the bbox and width/distance. This means that a linear model can be good enough to capture this pattern.

- **Velocity Estimation:**:
  - How it works: We deploy a simple formula where velocity is equal to the change in distance divided by time between frames.
    1) Relative Distance using frame 35 and 40: As discussed in the vehicle point estimation, this method of approach depends heavily on calculating the relative distance of a certain vehicle from the camera. In this approach our team decided to use the relative distances of vehicles in frame 35 with relative distances of vehicles in frame 40. From those distances we can find out the difference between those relative distances.
    2) Relative Time between frame 35 and 40: In terms of determining the time passed between frame 35 and frame 40, our team used a $\frac{2}{40}$ seconds per frame as our basis to calculate 5 frames:

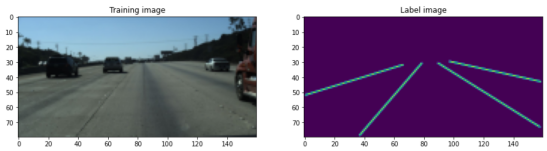
$$Time\_for\_5\_frames = 5 * \frac{2}{40} = 0.25s$$

    3) Relative Velocity between frame 35 and 40
    4) The final step of our velocity estimation approach is to apply the formula:

$$velocity = \frac{change\_in\_distance}{time}$$

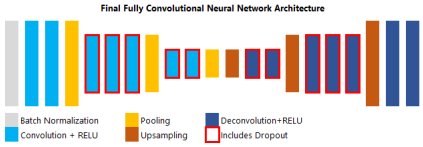
  - **Reason for choice** : Assuming that the distance calculations and the bbox dimensions detected by the YOLO algorithm are accurate, we are able to effectively calculate the change in relative distances between the vehicles detected by YOLO. Using the change in relative distances we can easily calculate the relative velocity of each vehicle in the frame. With the limitations of our dataset and annotated labels, such a simplistic approach is valued, especially considering computation time would be minimal compared to deep learning or machine learning approaches.

- **Lane detection:**:
  - **Lane detection using CNN model with raw images as input**: This approach mainly using raw input images (20^{th}) provided from lane detection tumsimple dataset and generated labels from the given label JSON file.
    - **Training images:** As in this method, we experienced train our CNN model by using the 20^{th} of every clips without applying any pre-processing techniques on the frames.
    - **Labels:** In order to train our CNN model, we need to generate one label for each clip from the JSON file. And the code used to generate label for a given clip is from `get_label` [25] function in laneDetectionHelperFunctions.py file.



Training image and its label image

    - **Create CNN model:** The creation of CNN model is based on SegNet's Fully Convolutional architecture and it is shown as the below diagram.



CNN architecture [26]

The snippet of code for creating CNN model comes from the `cnn_model` [27] function in `CNNModel.py` file

* **Train CNN model:** After creating our CNN model for lane detection task, there are a few things need to be setup before we fit the training images and labels into the model such as splitting train and test data, choosing values for a number of parameters. More details will be discussed in the next section.

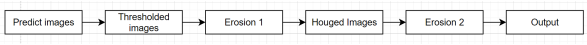
– **Lane detection using CNN model with preprocessed images as input**: In this approach, instead of using raw images fitting into our CNN model, I use a number of preprocess techniques to deal with our raw images before fitting into the model and once the model outputs a prediction, we need to do post processing on the prediction to get the desired output image.

* **Preprocess training images:** The pre-processing step is shown as the following diagram.

Pre-processing training images

At the `ROI` step, to get most insight information from the input images, I only consider the bottom half of the image because the top haft mostly contains irrelevant information such as tree, sky, buildings, etc...

* **Labels, Create CNN model and Train CNN model:** These steps are the same as the first method.
* **Post process prediction:** In order to get desired output images from predictions of our model, we need to apply post-processing step on the predictions.



Post process on prediction image

As we can see, from an output image of the model, we apply threshold to the image to compute binary image. From many experiments, in order to detect lines by using Hough line transform effectively, first we apply erosion morphological operation on the binary image to reduce the shapes. Once we detect lines from the image, we apply another erosion step to make the line thinner as it is supposed.

– **Final decision:** After doing a few experiments with these two approaches, the first method gave us nothing from the prediction of the model due to the model was overfitting even we have tried to use external training set from other resources. Eventually, we have decided to choose the second approach for this lane detection task. This will be discussed more detail in the next section.

IV. Experimental Setup

A. *Experiments on Car detection:*

- **Experimental setup of YOLOv3**:
  – The experiment was initially setup such that the YOLO model would be retrained on the training set that had been provided to the group by TuSimple [3]. Since the training set labels only provided bounding box values for the 40th image, training could only be conducted on 1074 images.
  – The code for training the YOLO algorithm was provided by Anton Mu which was taken from GitHub and was modified such that the model could be trained on the training set [28]. The YOLO model would be trained using batch size of 32 with an epoch of 102 which took approximately 4 hours to train the new YOLO model.
  – Upon testing out the new model, it was found to not detect cars as efficiently as previously thought. The main reason as to why it was not detecting cars efficiently was most likely due to not enough variation in the training set. As shown in Figure 3 below, we can see that the YOLO model was not able to detect the red car on the left as the training set did not have many cars that were labelled with the bounding box at this angle, and thus was not able to successfully learn how to detect cars that are on this angle.

Figure 3: Initial YOLO Model which had been retrained on the training set provided by TuSimple not having as high efficiency as previously thought

  – The next approach that the team took would be to train the model with the extra training set supplied by TuSimple. The code for training the model was then adapted to incorporate the extra training set. However, it was soon realised that training the new model with the extra training set resulted in a training time of roughly 12 hours and was decided that this would take too long.
  – The team then decided to just modify the predictions on the original YOLO model to only detect cars and

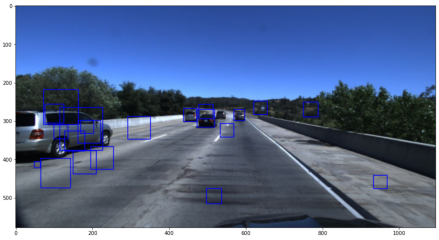
trucks in the image with a confidence threshold of 0.25 [19].

- **Experiments on R CNN:**
  - **Number of Proposal Boxes**: The suggested number of regions, by Ross Girshick was 2000 [29]. The more region proposals, the more regions that the selective search algorithm will analyse and extract relevant information from. Because of this relationship, we decided to vary the number of proposal boxes. What we saw was that when increasing the proposal boxes, the computation time becomes much greater, the performance did not change much and this may be because 2000 is an optimal number of regions for each image. Going further than 2000 would only create overlapping regions which provide the same information about an image region. Decreasing the number however we saw that computation time is faster, the performance did not rigorously change with an evaluation score of 98% compared to 99% at 2000 proposal boxes.
  - **Varying the Intersection of Union Threshold**: This threshold was used in the computeIoU() function which was adapted from Rosebrock's tutorial [22]. What was suggested by Rosebrock was an IoU of 0.7, but we found that it may have been chosen since most of the images in Rosebrock's training dataset had objects that were significant in size, thus having a 0.7 ratio would be able to take region proposals that cover even just 70% of the destination object. For the TuSimple dataset, vehicles are relatively small, and this is the case especially for the annotated vehicle. So, we would need a high IoU ratio which will eliminate the number of duplicate regions and just have the entirety of the car image as a positive example, rather than just a part of a vehicle – which we saw to have negative results in that the model, at times, predicted false positives.
  - **Varying the Non-Max Suppression Threshold**: This threshold was varied throughout our experiments seeing smaller thresholds to have fewer detection boxes in the end but tends to stretch across multiple positives. With higher thresholds, we can see more detected boxes on positive examples, but did not do its job on eliminating overlaps.

- **Experiments on Haar Cascade:**
  - we extracted area that contains car and no-car images. After all we collected more than 1074 car and non-car samples and resize them to `64x64` pixel by pixel and convert them to gray scale images.
  - Car and non-car samples are shown as below.

Car and non-car images for training Haar Cascade model

  - The next step is to create two descriptor files for car and no-car images named pos.txt and neg.txt respectively. The code for create two descriptors in the 'Create two descriptor files for car and no-car training set' section from HaarCascade jupyter notebook [30].
  - Run the commands as discussed in the previous section to train our Haar Cascade model.
  - Here is the result when the Haar Cascade model predicts on an input image.



Prediction from Haar cascade model

B. Experiments on Position Estimation

- Experiments on constant width based position estimation
  - Based on the triangle simularity method and the ground truth regarding the positions as well as bbox, we use the formula

$$W = \frac{dX * pixel\_width}{fx}$$

    to calculate all ground truth width for 1447 cars in the training data.
  - After the experiment, we average this 1447 width and get the average car width $2.156m$. Then we use this width as a constant to make prediction of cars' positions for all training and testing data.
  - The average distance error is 3.86 in the training data set and 4.08 in the test data set.
  - This error in distance is not ideal considering that we average this error over 1447 cars. This method doesn't give us a good result not only because cars widths are not a fixed constant in reality, more importantly, the pixel width we use to do estimation sometimes fail to reflect the width of the car in the image especially if a car is away from the center of the camera.
- Experiments on Linear Regression Model based position estimation

– **Experiment on predicting distance 1:**
  * 4 Features are selected: bbox top value, bbox left value, pixel width (bbox right value - bbox left value), pixel height (bbox bottom value - bbox top value).
  * After feeding linear regression model 1447 cars information, the coefficients we obtained are $[-0.0003, -0.03435, 0.02162, -0.0334, 14.2527]$.
  * Then for the width we obtain, we calculate the distance based on the formula
$$dX = \frac{fx * W}{pixel\_width}$$
  * **Observation on Experiment result:** The average distance error is 2.53 in the training data set and 2.62 in the test data set. The mean square error on training set is 0.04.
– **Experiment on predicting distance 2:**
  * 4 Features are selected: bbox left value, pixel width (bbox right value - bbox left value), pixel height, $pixel\_width * pixel\_height$.
  * After feeding linear regression model 1447 cars information, it returns a linear equation $[-0.0305, 0.01964, -0.0265, -1.3394e^{-05}, 12.5666]$
  * **Observation on Experiment result:** The weights on the area is very small, only $-1.3394e^{-05}$ but the overall performance improves. The average distance error is 2.46 in the training data set and 2.55 in the test data set. The mean square error on training set is also 0.04.
– **Experiment on predicting distance 3:**
  * After analysing the error pattern, the team discovered that most mismatch occurs on cars very far or very near to the camera. So instead of training one model for all data, we can train smaller models for cars based on their y-axis value.
  * Our team divide 1447 data points into 3 classes based on their ground truth distance: a car is deemed as far if its distance is bigger or equal than 45. It's near if its distance is smaller than 45 but bigger than 20. Otherwise it's deemed as near. After division, there are 305 data point in $training\_far$, 941 data point in $training\_middle$, 196 data point in $training\_near$.
  * For each of the 3 list, a linear regression model is trained.
    · Near $[-2.2321e^{-02}, 1.4238e^{-02}, -2.5765e^{-02}, 4.9736e^{-06}, 10.1432]$
    · Middle: $[-0.0392, 0.0380, -0.0296, -0.0001, 15.0486]$
    · Far $[-0.0281, 0.01575 - 0.0811, 0.0014, 12.18]$
  * In the prediction phase, we first use the linear model from the 2nd experiment as an initial evaluation of the distance, based on which we classify the cars as far, near or middle. Then we can choose the model accordingly.
  * **Observation on Experiment result:** The average distance error is 2.28 in the training data set and 2.42 in the test data set. The mean square error for model 1 on training set is 0.03. That for model 2 on training set is 0.02. For model 3 on training set is 0.03. All 3 are better than the initial model.
  *

- **Experiments on calculating the lateral distance**
  – As discussed in the constant width based approach, the calculation of lateral distance is based on the formula
$$dY = dy * dX/fx$$
where $dy = (left\_value - p1)$. To decide which value that we should set $p1$, we loop through a range of value $[600, 610, 620, 630, 640, 650, 660, 670, 680, 690, 700, 710, 720]$ and calculate the average error on lateral distance obtained on training data set.
  – The experiment result shows that when $p1 = 670$, we obtain the minimum average error 0.48 on training set and 0.50 on testing set.

C. Experiments on Velocity Estimation

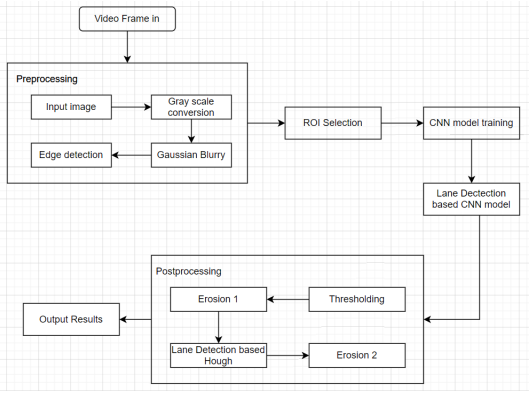
- **Different Frames Times**: We varied the number of frames between frame 40 and the beginning frame. We tried frame variations like [1, 40], [30, 40], and [39, 40]. However we came across a study on vehicle velocity estimation which suggested that the most optimal range is 5 frames – [35, 40]. We also decided that the correct time between frames would be:
$$\frac{seconds\_per\_clip}{frames\_per\_clip} = \frac{2}{40}$$
Therefore, our final time would be:
$$5\ frames * \frac{2}{40} = 0.25$$
- **Manual Experiment**: The next experiment our team performed was to manually determine the bbox dimensions of vehicles in between frames. The rationale was that the distance and velocity precision depends heavily on the accuracy of the bbox. It would yield much greater results and accuracy if the bbox were pinpoint accurate to the pixels. Thus, we drew the bboxes by hand to retrieve the most accurate dimensions to the human eye.
- **Using Different Distance Calculations**: Since Task 2 has a high dependency on Task 1, we decided to estimate the velocity using various types of distance estimation experiments as proposed in the earlier section. There were cases where the velocity estimation error grew with the distance estimation error which is suggestive of the positive relationship between velocity and distance. Thus, this implied that we need perfect distance calculations to achieve accurate velocities.

- **Backtracking for Validation:** Our team attempted to backtrack from the ground truth velocities to compute the position of the vehicle from which we could calculate the distance as well as the bbox dimensions of the target vehicle. If results closely matched with the bbox and position labels of the annotated vehicle, it is suggestive to our approach being one that is appropriate or not.

D. Experiments on Lane Detection

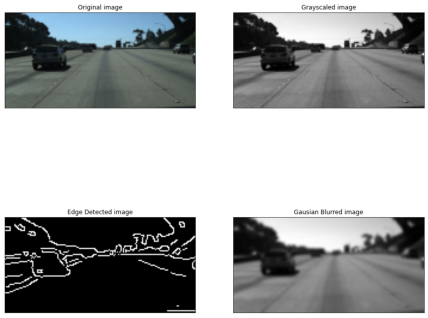
In this section, due to the similarity of the two methods discussed in the previous sections and the first method's outcome was not successful on the lane prediction. Therefore, I will only discuss the second method in this part.

- **Overview of the proposed method:** In this approach, the lane detection task is divided into 4 steps: (1) image preprocessing, (2) ROI selectin, (3) model training and (4) image postprocessing. The steps are shown as the diagram below.



Overivew of the Lane Detection approach

- **Preprocess on training images:** The preprocessing step is implemented in the `pre_process` function in the `laneDetectionHelperFunctions.py` file.

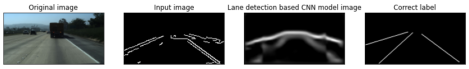


Preprocessing steps for an input image

- **ROI selection:** By observation from all the training set, most of the lanes are appeared in the bottom half of their images. That is why we decided to do this ROI selection step in order to remove irrelevant information as much as possible from the input image. And the `region_of_interest` function [31] in the `laneDetectionHelperFunctions.py` file implements this step. Here is the result after we apply ROI selection step on the preprocessed image.

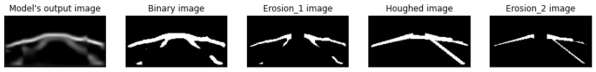
ROI selection step

- **Training CNN model:** The code from `cnn_model` function [27] in `CNNModel.py` file and the code from `Train CNN model` section [27] in the `LaneDetection` jupyter notebook are used to create and train our CNN model. Here we have been using `batch_size=128 epochs=10 pool_size=(2,2)` as our parameters for the model.
- **Lane Detection based CNN model:** Once we finish training our model, here is the result when we input an image to the model as shown below.



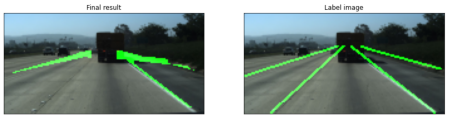
Lane detection based on CNN model vs its label

- **Postprocess on the model's output:** As we can see, the prediction from the model is quite blur, so that we need to perform post processing on the output to make it more accuracy when comparing with its label. After experimenting number of ways in this step, we came up with this order: Threshold $\rightarrow$ erosion 1 $\rightarrow$ hough transform $\rightarrow$ erosion 2 $\rightarrow$ final output. The whole process is implemented in the `post_process` function in the `laneDetectionHelperFunctions.py` file while the hough tranform implementation is from the `hough_lines` function [31] in the `laneDetectionHelperFUnctions.py` file. Please refer to the previous section to see the steps in the post processing prediction image part. Here is the results of every step in the post processing prediction image part as shown below.



Post processing steps

- **Put things together:** Once we finish our post processing step, putting our predict result on the input image and compare to its label then we got as the follow.



Final result vs Label image

V. Results and Discussion

- Evaluation on Car Detection
  - Results of Haar Cascade:
    - The Haar Cascade when implemented returned reasonably low accuracy when trying to detect cars. Shown in Figure 4 below, some cars in the distance have been captured through the Haar Cascade model. There are a lot of false positives being captured on the image. This low

accuracy is likely due to the amount of training data supplied by TuSimple being insufficient for successful learning.

Figure 4: Haar Cascade detection algorithm

- Results of YOLOv3:
  - The YOLO algorithm produced very high accuracy when detecting cars in an image with minimal false positives, detecting most if not all cars in the image, even successfully detecting cars that are occluded with a reasonably high accuracy as shown in Figure 5 and Figure 6 below.

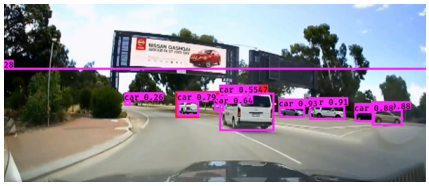


Figure 5: Final YOLO Model which utilised the original YOLO weights, with outputs only showing cars and trucks

  - This is due to the YOLO Algorithm being trained on the COCO dataset [32], which is a large-scale object detection, segmentation, and captioning dataset with over 200,000 labelled images and 80 object categories. Since the model is being trained on such a large dataset, the YOLO model is able to successfully generalise the features of a car and thus, is able to obtain such high accuracy. Comparing the results of the YOLO model being trained on the TuSimple training set shown in Figure 3 as opposed to the COCO dataset, we can clearly see the difference in ability to detect vehicles.

Figure 6: YOLO Detection algorithm

- Results of R-CNN:
  - The RCNN-based vehicle detection model had an accuracy that bested the HaaR cascade method of vehicle detection. One main reason would be the fact that this model utilises selective region of interest search to extract accurate car and non-car images. However, since in each image we were only given access to only one annotated vehicle, we were not able to fully train the model with vehicles with features that were dissimilar to that of those annotated by the bboxes. As a result, on some images there may be instances where vehicles were not correctly predicted with several false positives, reducing the overall accuracy of the vehicle detection model.

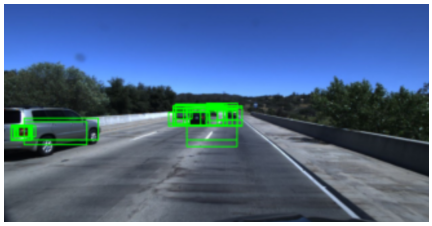
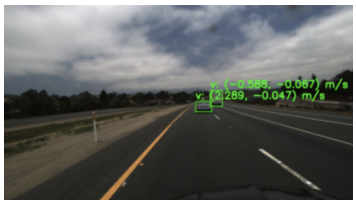


Figure 7: R-CNN Detection Model

- Evaluation on Position Estimation
  - Results: Based on the demo code on position evaluation, the model we developed has the below performance.
    - Position Estimation error (Near): 1.79919
    - Position Estimation error (Medium): 6.28537
    - Position Estimation error (Far): 34.53903
    - Position Estimation error (total): 14.20787
    - Result Showcase

  - Discussion
    - The above result on evaluation shows that the model does great on cars that are near to the camera, reasonably acceptable on the medium distance and worse on the far distance.
- Evaluation on Velocity Estimation
  - Results
    - Using the demo code provided to us by TuSimple velocity estimation challenge as the basis of our evaluation approach, we were able to achieve the below performance.
      - Velocity Estimation error (Near): 16.90006
      - Velocity Estimation error (Medium): 179.26929
      - Velocity Estimation error (Far): 1123.73019
      - Velocity Estimation error total: 439.966511
    - Result Showcase



  - Discussion

    - The conclusion we make from the evaluation results are that our approach performed significantly much better on vehicles that were close compared to vehicles that were further away. Our belief as to why this happens may be that vehicles that are further away are expected to only move by single digit pixels so thus in calculating the velocities, we need pinpoint pixel accuracy in terms of bbox dimensions as well as distance calculations.
- Evaluation on Lane Detection
  - Evaluation: for Lane Detection, the way we evaluate is we taking the final prediction label image and the ground truth label image, from those binary images, we calculate true positives, false positives, true negatives and false negatives according to their pixel vales. This is implemented in the `compare_binary_images` function from the `LaneDetectionEval` jupyter notebook. The function outputs precision, recall and accuracy values to evaluate how accuracy of the predicted label image compares to its ground truth label.
  - Results:
    - Average precision: 0.30859
    - Average recall: 0.23950
    - Average accuracy: 0.30859
  - Discussion: Most of the time, our model predict really well the outer lane, for the inner lane, our edge detector mostly fail to detect due to it is hard to distinguish between the inner lanes and the road. Therefore, the model will not learn much about the inner lanes.

VI. Conclusion

A. What worked

- Position Estimation: the triangle similarity works as expected. The linear regression approach improves the accuracy of prediction as expected. The result is consistent with our analysis in the section of method.
- Velocity Estimation: Our method of choice which was making use of a simple velocity formula derived from distance and time changes worked satisfactorily for vehicles that were considered 'near'.
- Lane Detection: with the preprocessing step, the CNN model is pretty good at detecting outer lanes from a given image.

B. What did not worked

- Position Estimation: the prediction on the cars that is far from the camera has low accuracy compared with those near the camera or has medium distance from the camera.
- Velocity Estimation: Our method failed to perform above satisfactory levels on vehicles that were further away. We believe it to be because of the precision we needed from prior tasks so that we could better calculate the velocity of these vehicles.
- Lane Detection: the model is not good at detecting inner lanes and sometime it detects safety barriers as lanes.

C. Potential Improvement

- Position Estimation: The overall performance can be improved if we have more data, especially on cars far from the camera. Also we can consider other machine learning methods like CNN. Other experiments that can be done include testing on different combination of features.
- Velocity Estimation: In terms of improvements, our team have decided on some considerations: using image segmentation to get better pixel coordinate accuracy, rather than object detection with bounding boxes as well as making use of machine learning or neural network architectures.
- Lane Detection: There are lot of rooms to improve this task such as using more data from external sources. Also due the properties of thin lanes so that the pixels belong to lanes are lot fewer than the background pixels. Therefore, we might need to develop a REcurrent Feature-Shift Aggregator (RESA) [33] to gather information within feature maps and pass spatial information more directly and efficiently [33].

VII. Contribution of Group Members

- Changyong researched basic methods for task 1 and 2, and helped experiment with Ty. Report writing literature review and basic methods of Task 1.
- Harry was tasked on implementing YOLOv3 for car detection for Task 1 and Task 2. Worked with Dino, Ty and Henry. Contributed to report for YOLO.
- Henry implemented the R-CNN model and task 2. Worked with Ty to review methods for task 1 and 2. Report writing and presentation for assigned sections.
- Minh implemented Haar Cascade model for car detection part and lane detection part for task 3. Report writing and presentation for assigned sections.
- Tiankuang(Ty) develops the linear regression model for task 1 and helps Henry on task 2. She writes the report part she is in charge of and also makes presentation.

REFERENCES

[1] Min Jiang Mingfa Li, Yuanyuan Li. Lane detection based on connection of various feature extraction methods, 2018.

[2] TuSimple. Tusimple velocity estimation challenge.

[3] TuSimple. Tusimple lane detection challenge.

[4] COMP 9517 Lecture. webcms week 5 lecture.

[5] TuSimple. Classification, object detection and image segmentation.

[6] Jason Brownlee. A gentle introduction to object recognition with deep learning.

[7] D. M. Gavrila and V. Philomin. Real-time object detection for "smart" vehicles. In *Proceedings of the Seventh IEEE International Conference on Computer Vision*, volume 1, pages 87–93, 1999.

[8] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. *Commun. ACM*, 60(6):84–90, May 2017.

[9] Lu Dongming Qian Weixian Ren Kan Zhang Jun Xu Liwei Liu, Zewei. Vision-based inter-vehicle distance estimation for driver alarm system. *IET intelligent transport systems*, 13(6):927–932, 2019.

[10] J. Bouguet. Camera calibration toolbox for matlab.

[11] Hamdaoui Fayçal Mtibaa Abdellatif Bougharriou, Sana. Vehicles distance estimation using detection of vanishing point. *Engineering computations*, 36(9):3070–3093, 2019.

[12] Cheng-Jian Lin Chi-Yung Lee Chi-Feng Wu. Applying a functional neurofuzzy network to real-time lane detection and front-vehicle distance measurement. *IEEE transactions on systems, man and cybernetics. Part C, Applications and reviews*, 42(4):577–589, 2012.

[13] K. Osamura, A. Yumoto, and O. Nakayama. Vehicle speed estimation using video data and acceleration information of a drive recorder. In *2013 13th International Conference on ITS Telecommunications (ITST)*, pages 157–162, 2013.

[14] Sharif Elfouly. Vehicle speed estimation from video using deep learning, 2020.

[15] C Dominik M Mateusz. Velocity calculator, 2021.

[16] Sumit Saha. A comprehensive guide to convolutional neural networks — the eli5 way.

[17] Alexander Mordvintsev Abid K. Hough line transform.

[18] Manishgupta. Yolo - you only look once.

[19] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv*, 2018.

[20] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[21] Rohith Gandhi. R-cnn, fast r-cnn, faster r-cnn, yolo — object detection algorithms, 2018.

[22] Adrian Rosebrock. R-cnn object detection with keras, tensorflow, and deep learning, 2020.

[23] Opencv documentation on image segmentation.

[24] Moritz Kampelmu hler. Estimating the velocity of vehicles relative to a moving camera.

[25] Anand Kummari. Detecting lanes using deep neural networks.

[26] Michael Virgo. Lane detection with deep learning (part 2).

[27] mvirgo. Mlnd-capstone.

[28] Anton Muehlemann. Trainyourownyolo: Building a custom object detector from scratch, 2019.

[29] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation, 2014.

[30] sentdex. Creating your own haar cascade opencv python tutorial.

[31] KDnuggets. Road lane line detection using computer vision models.

[32] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft coco: Common objects in context. *Computer Vision – ECCV 2014 Lecture Notes in Computer Science*, page 740–755, 2014.

[33] Tu Zheng, Hao Fang, Yi Zhang, Wenjian Tang, Zheng Yang, Haifeng Liu, and Deng Cai. Resa: Recurrent feature-shift aggregator for lane detection, 2021.