VIP-IPA: Lane Detection Team

Final Report

Fall 2022

Pume Tuchinda, William Stevens, Mingyu Kim,

Justin Chan, and Daniyaal Rasheed

CONTENTS

1. Abstract

With the growing advancements in the field of autonomous driving, the computer vision algorithms utilized for such a task must be robust and accurate to correctly assess road features from images captured in real-time. In our previous work, we designed a neural network model according to the U-Net architecture and trained it on the BDD100k dataset to produce binary segmentations of the locations of lane lines in an image. With the success of last semester's model, we wanted to branch out towards other vision tasks for autonomous driving like object detection, and also develop a proof-of-concept model for multi-task learning. This process includes the development of a neural network to perform traffic object detection alongside lane line and drivable area segmentation, all within a single network. This will be achieved through modifying last semester's segmentation model and implementing an object detection model through utilizing the YOLO framework with a Darknet-53 backbone. The modifications to the model will include implementing a path aggregation network decoder through the works done in Feature Pyramid Networks and Spatial Pyramid Pooling. The decoder is designed to aggregate backbone features at multiple resolutions, allowing the model to selectively merge shallow and deep information to increase the receptive field of the model and allow better predictions. These developments will allow a single neural network model to perform the three crucial vision tasks for autonomous driving and create a new proof-of-concept model for future work to build upon.

2. Introduction

The goal of the project for this semester was to continue developing and testing an algorithm to successfully perform detection tasks for autonomous driving. Our work takes inspiration from the Fall 2021 Lane Detection team, which utilized the You Only Look Once (YOLO) object detection framework. We also built off progress made by the Spring 2022 Lane Detection team, which focused solely on lane detection and segmentation using the U-Net architecture. The initial objective of the team this semester was to successfully develop the first version of the YOLO framework, which is denoted YOLOv1. Our plan was to follow in the footsteps of the original developers of YOLO by working through each subsequent version of the framework to fully understand the thought processes and design decisions that were made with each version. Thus, the ultimate goal for this semester was to fully develop a modification of the YOLO network to be able to perform object detection, lane detection, and drivable area segmentation through a unified model for multi-task learning.

3. Dataset

We are continuing the use of the BDD100k dataset from previous semesters because it contains all the necessary labels needed for the three tasks outlined in our goal. The dataset contains 100,000 driving videos and labeled images collected from more than 50,000 rides covering New York City and San Francisco Bay Area year-round with multiple weather and lighting conditions. Due to the diversity of geography, environment, and weather, training on the dataset results in a robust and adaptable model for encountering new environments or changes of seasons. Containing thirteen different object classes shown in Table I and nine different lane classes shown in Table II, the BDD100k dataset encapsulates many of the classes that the model could potentially be tasked to analyze. Another component of the dataset that was utilized for our purposes is the labeling of drivable areas, which is divided into two classes: direct and alternative. Direct drivable area is defined as the lane the vehicle is currently driving on and thus the region where the driver has priority over surrounding cars. Alternative drivable area is defined as any lane that the vehicle is currently not driving on, but has the ability to do so through changing

lanes [1]. The BDD100k dataset with its large size, wide variety, paired and highly detailed labeling, makes it a perfect fit for the success of our training purposes.

Category ID	Class
1	pedestrian
2	rider
3	car
4	truck
5	bus
6	train
7	motorcycle
8	bicycle
9	traffic light
10	traffic sign
11	other vehicle
12	trailer
13	other person

TABLE I: Labeled classes for object detection in BDD100k. [1]

Category ID	Class
0	crosswalk
1	double other
2	double white
3	double yellow
4	road curb
5	single other
6	single white
7	single yellow
8	background

TABLE II: Labeled lane categories for lane marking in BDD100k. [1]

Fig. 1: Object detection visualization during the daytime. [1]

Fig. 2: Drivable area visualization during the nighttime. [1]

Fig. 3: Lane marking annotations during the nighttime. [1]

4. YOLOv1

A. *Introduction*

Introduced in 2015 by Joseph Redmon et. al [2], the You Only Look Once (YOLO) framework provided a new approach to object detection through re-framing the problem into a regression problem. The YOLO architecture is part the one-stage detector family, where classification and bounding box localization are performed simultaneously. This leads to a simpler and faster model, but introduces trade-offs including less accuracy and issues with the generalization of less represented classes. There are two core concepts behind the YOLO architecture. First, the input image is split into $S \times S$ grids, where S is a user-defined hyperparameter for the amount of grids to split the image into. Second, each object in the image is assigned to a grid in which their center lies, designating that grid cell to be solely responsible for the detection of that object. Within each grid cell the model will detect B bounding boxes, where B is a user-defined hyperparameter denoting the amount of bounding boxes generated per cell. Of these B bounding boxes within each grid cell, the model will choose the one most similar to the groundtruth data in order to quantify the accuracy when computing the loss.

B. Architecture

The YOLOv1 architecture utilizes a convolutional neural network (CNN) to perform its detections. It should be noted that the CNNs used by most developers today actually use cross-correlation rather than convolution to execute operations on the image matrices. The term "convolutional neural network" is quite a prevalent misnomer in the field, and it is important to understand the difference between the two operations. Nonetheless, we will use the term CNN to represent this concept. The CNN described in the framework contains 24 convolutional layers followed by two fully-connected layers shown in Figure 4. Each convolutional layer is followed by a batch-normalization layer and a leaky ReLU activation function [2]. These convolutional layers are grouped into sections, and after each section of layers is applied, the architecture downsamples the image by applying a 2x2 max-pooling operation. The output of the last convolutional layer is passed into two fully-connected layers to achieve the final output from the model, which is a tensor of shape $S \times S \times (5 * B + C)$ containing all the detections, where $S \times S$ = the number of grids the image is split into, B = the number of detected bounding boxes per grid, and C = the number of classes in the dataset [2].

Our configuration of YOLOv1 for training on the BDD100k dataset sets the hyperparameters as, $S = 7, B = 2$, and $C = 13$. Therefore, the final output of our model is a $7 \times 7 \times 23$ tensor. This output represents a 1-dimensional tensor of size 23 for each of the 49 grid cells, where the 23 values in the tensor corresponds to the model's predictions for the class probability distribution and two bounding boxes inside each grid cell, as shown in Figure 5. For each of the B bounding boxes, there are five data points containing the x-coordinate and y-coordinate of the bounding box's center, the width and height of the bounding box, and the objectness confidence for each bounding box. Following the bounding box parameters is the class probability distribution signifying how the probability that the object belongs to each class.

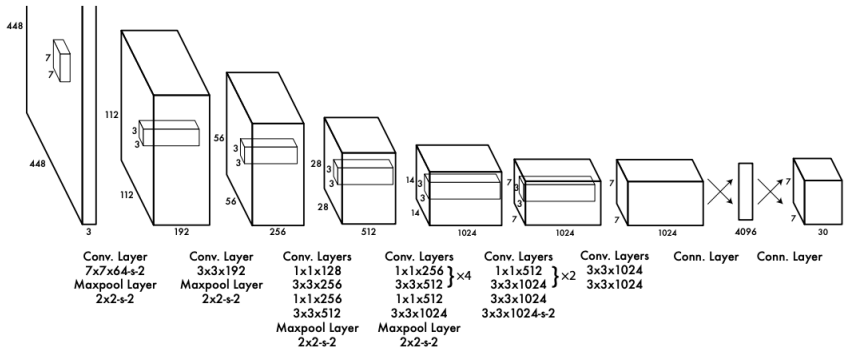


Fig. 4: YOLOv1 Architecture [2]

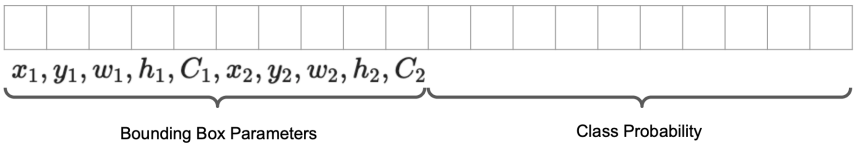


Fig. 5: YOLOv1 Output

C. Loss Function

To quantify the accuracy of the model for a given set of weights, the loss function presented in the YOLOv1 paper is utilized [2]. This function will take the output of the model at each iteration and compute a scalar value representing the loss of the model for that specific set of weights. This loss represents the amount that the model should be penalized for its predictions. If the loss is very high, the weights are incorrect and must be changed drastically. If the loss is approaching zero, the model is approaching a set of weights that are optimal for the task. To compute the loss for the YOLOv1 model, Equation 1 shows the entire loss function which was derived from the combination of the three main aspects of the model's output [2]. These are the bounding box coordinates, the objectness confidence, and the class probabilities. Each of these loss components use sum-squared error loss when comparing the detections with the ground truth data. The detection's data points are retrieved from the prediction tensor for each of the B detected bounding boxes in each grid cell. For choosing which of the B bounding boxes to compute loss on, our architecture computes the IOU, defined by Equation **??**, between each

proposed bounding box and the ground truth selecting the bounding box with the highest IOU to determine which is the best candidate for that grid cell [2].

$$
\begin{aligned}
Loss = \lambda_{coord} \sum_{i=1}^{S^2} \sum_{j=1}^{B} I_{ij}^{obj} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right] \\
+ \lambda_{coord} \sum_{i=1}^{S^2} \sum_{j=1}^{B} I_{ij}^{obj} \left[(\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2 \right] \\
+ \sum_{i=1}^{S^2} \sum_{j=1}^{B} I_{ij}^{obj} \left(C_i - \hat{C}_i \right)^2 + \lambda_{noobj} \sum_{i=1}^{S^2} \sum_{j=1}^{B} I_{ij}^{noobj} \left(C_i - \hat{C}_i \right)^2 \\
+ \sum_{i=1}^{S^2} I_i^{obj} \sum_{c_{obj} \in classes} \left(\mathbb{P}_i(c_{obj}) - \hat{\mathbb{P}}_i(c_{obj}) \right)
\end{aligned}
\tag{1}
$$

Where λ_{coord} is a scaling factor for bounding box loss, λ_{noobj} is a scaling factor for confidence loss, S is the number of grids to split the image into, B is the number of bounding boxes to predict for each grid cell, x_i and y_i are the coordinates of the center of the groundtruth bounding box, $\hat{x}_i$ and $\hat{y}_i$ are the coordinates of the center of the detected bounding box, w_i and h_i are the width and height of the groundtruth bounding box, $\hat{w}_i$ and $\hat{h}_i$ are the width and height of the detected bounding box, I_{ij}^{obj} is the value at i, j of a binary matrix defining whether the grid cell is responsible for predicting the bounding box, I_{ij}^{noobj} is $1 - I_{ij}^{obj}$, C_i is the groundtruth value representing whether or not an object exists in the grid cell, $\hat{C}_i$ is the confidence score of whether or not an object exists in the grid cell, $\mathbb{P}_i(c_{obj})$ is the value representing the object's groundtruth class, and $\hat{\mathbb{P}}_i(c_{obj})$ is the class probability for which object class the model predicts.

Equation 2 shows the first component of the total loss equation: the penalty term addressing the loss with respect to the bounding box coordinates and dimensions. This term is computed from the sum-squared errors of four data points from each bounding box detection. These four data points are the x-coordinate and y-coordinate of the bounding box's center, and the width and height of the bounding box. For the sum-squared errors for width and height, the square root of their values is used rather than their raw values. This solves the problem of inconsistency when computing bounding box loss for bounding boxes of different dimensions. For example, a difference of 10 pixels in a detected bounding box width, where the groundtruth box is only 20

pixels wide, should result in very high loss due to the dimensions being a factor of two different from each other. However, for a difference of 10 pixels in detected bounding box width, where the groundtruth box is 200 pixels wide, the resultant loss should be quite low since the difference is only a small fraction of the groundtruth's width. Using the square root operation on these widths and heights addresses this inconsistency from the bounding box loss equation.

$$\lambda_{coord}\sum_{i=1}^{S^2}\sum_{j=1}^{B} I_{ij}^{obj}\left[(x_i-\hat{x}_i)^2+(y_i-\hat{y}_i)^2\right]+\lambda_{coord}\sum_{i=1}^{S^2}\sum_{j=1}^{B} I_{ij}^{obj}\left[(\sqrt{w_i}-\sqrt{\hat{w}_i})^2+(\sqrt{h_i}-\sqrt{\hat{h}_i})^2\right] \quad (2)$$

Equation 3 shows the second component of the total loss equation: the penalty term addressing the loss with respect to the model's confidence regarding whether or not the bounding box contains an object. This term is derived from the sum-squared error between the predicted objectness confidence and the groundtruth.

$$\sum_{i=1}^{S^2}\sum_{j=1}^{B} I_{ij}^{obj}\left(C_i-\hat{C}_i\right)^2+\lambda_{noobj}\sum_{i=1}^{S^2}\sum_{j=1}^{B} I_{ij}^{noobj}\left(C_i-\hat{C}_i\right)^2 \quad (3)$$

Equation 4 shows the third and final component of the total loss equation: the penalty term addressing the loss with respect to the model's class confidences regarding which object class the bounding box contains. This term is derived from the sum-squared error between the predicted class confidences and the groundtruth class.

$$\sum_{i=1}^{S^2} I_i^{obj}\sum_{c_{obj}\in classes}\left(\mathbb{P}_i(c_{obj})-\hat{\mathbb{P}}_i(c_{obj})\right) \quad (4)$$

In the bounding box and objectness confidence penalty terms, two scaling factors are used: λ_{coord} and λ_{noobj}. These factors are used in order to emphasize or minimize the impact that certain components of the loss function have on the total loss. For example, the paper suggests a λ_{coord} of 5 and a λ_{noobj} of 0.5 to "increase the loss from bounding box coordinate predictions and decrease the loss from confidence predictions for boxes that don't contain objects" [2]. The combination of these three components results in the total loss function for YOLOv1. This outputted scalar value for a specific set of weights can be used to quantify how much the model

should be penalized for its predictions, and help guide the model towards more optimal weights through backpropagation and gradient descent.

D. Evaluation

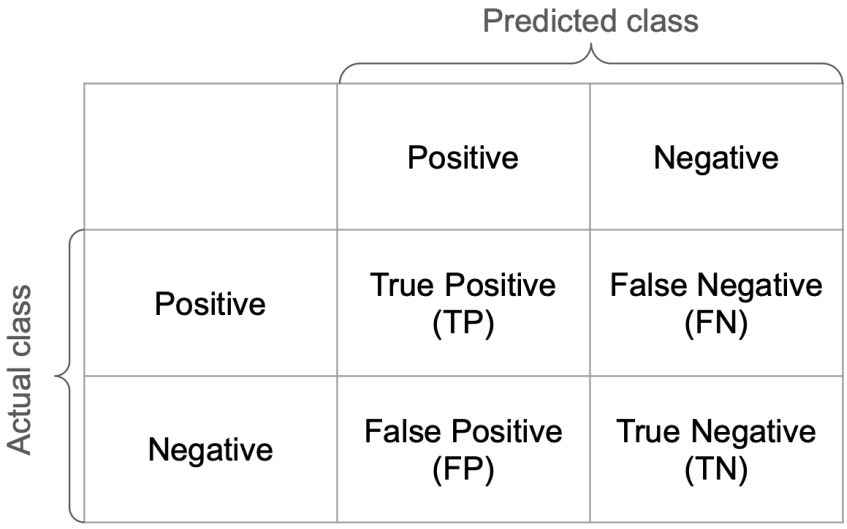


Fig. 6: Confusion Matrix

1) Mean Average Precision (mAP): Mean average precision (mAP) is a metric used to evaluate object detection models. mAP is calculated using average precision, which is the area under the curve of precision-recall graph defined in Equation 8.

$$Precision = \frac{TP}{TP + FP} \tag{5}$$

$$Recall = \frac{TP}{TP + FN} \quad Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{6}$$

$$AP = \int_0^1 Precision(Recall) d(Recall) \tag{7}$$

$$mAP = \frac{1}{classes} \Sigma_{i=1}^{classes} AP_i \tag{8}$$

Where AP_i is the average precision of class i, and $classes$ is the number of classes.

2) Intersection over Union (IOU): For object detection, Intersection over Union (IOU) was chosen to evaluate the accuracy of the predicted bounding boxes. IOU measures the amount of overlap between the groundtruth data and the predictions.

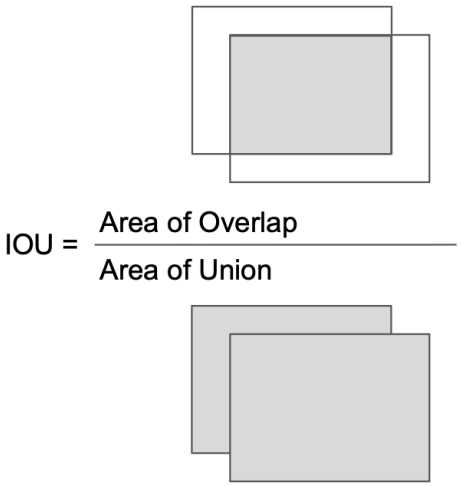


Fig. 7: Visual Representation of IOU

$$IOU = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|} \tag{9}$$

Suppose A is the predicted bounding box and B is the ground truth box. Using the equation in Equation 9, IOU is computed to measure the overall accuracy of the bounding boxes to the ground truth bounding boxes.

E. Results

Our YOLOv1 implementation was trained for 100 epochs on the BDD100k dataset with the Adaptive Momentum (Adam) Optimizer, and the resultant loss curve is shown in Figure 8 with a visualization in Figure 9. Since the YOLO model outputs two bounding boxes for each cell, it often creates bounding boxes that overlap on the same object. To address this problem we utilized non-maximum suppression to eliminate the extra bounding boxes that have a high IOU between each other by keeping the bounding box with the highest objectness confidence and removing the rest. We used mean average precision at an IOU threshold of 50 for measuring the accuracy

of our model. On 20,000 validation images, the mean average precision for our YOLOv1 was 28.64%. However, when diving deeper to see where the model really struggles, it was found that many of the smaller sized objects would often go undetected and that the dimensions of the bounding boxes for larger objects were either too large or too small. We believe the cause of this is due to the fact that our model only detects from the last convolutional layer's feature map, which has been downsampled to $\frac{1}{16}^{th}$ of the input image resolution. This results in a feature map where only certain types of object are visible. In this feature map, only medium sized objects are visible, as this size is the most prevalent size in the dataset. However, even with this issues we found the YOLOv1 model to be a good building block to be built upon for our next goal of creating a single neural network to perform both object detection and segmentation.

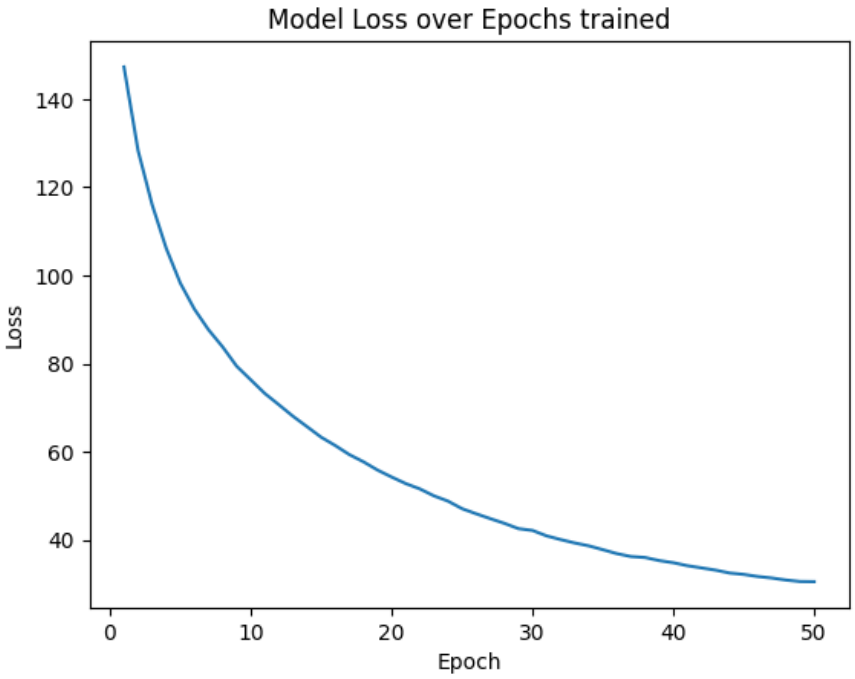


Fig. 8: Training loss curve for YOLOv1 for 100 Epochs

Fig. 9: Results of YOLOv1

5. Multi-task Net

A. *Introduction*

In our work last semester, we were able to build a proof-of-concept model for performing lane segmentation through the use of the U-Net framework [3]. Following last semester's success, we wanted to branch out towards other computer vision tasks needed for autonomous vehicles. Through this we chose to explore object detection with the implementation of YOLOv1 trained on the BDD100k dataset. However, the two models we had implemented had a multitude of problems - the main issue being the inefficiency caused by performing separate forward passes for each task. So, for the second half of this semester we wanted to build a model that unifies these two tasks into one network to allow for a single forward pass, and thus increase efficiency. Additionally, we wanted to improve our model's accuracy through the introduction of a deeper backbone in Darknet-53 and increase the input resolution. There are two main reasons for

changing the input resolution from the 416×416 that we used in YOLOv1 to the 384×640. First, increasing the resolution allows for the model to make higher quality detections by forcing such detections to be made at a higher resolution for both object detection and segmentation. Second, the original BDD100k image's resolution is 720×1280, and we wanted to keep the same aspect ratio rather than compressing the image into a square like we did in YOLOv1.

Figure 10 shows a high-level diagram outlining the full structure of our model. In the diagram, we use a notation C0, C1, ..., C5 and a similar sequence for P. This is a shorthand notation to indicate the resolution of the output from a given set of operations. The distinction between C and P is to separate the layers in Darknet-53 and the feature pyramid network. C5 indicates that the output resolution for a given block is $\frac{1}{2^5}$ of the input image resolution. This shorthand allows us to indicate where in the model the size of the image is changed and allows us to have an intuitive understanding of the dimensions at each step.

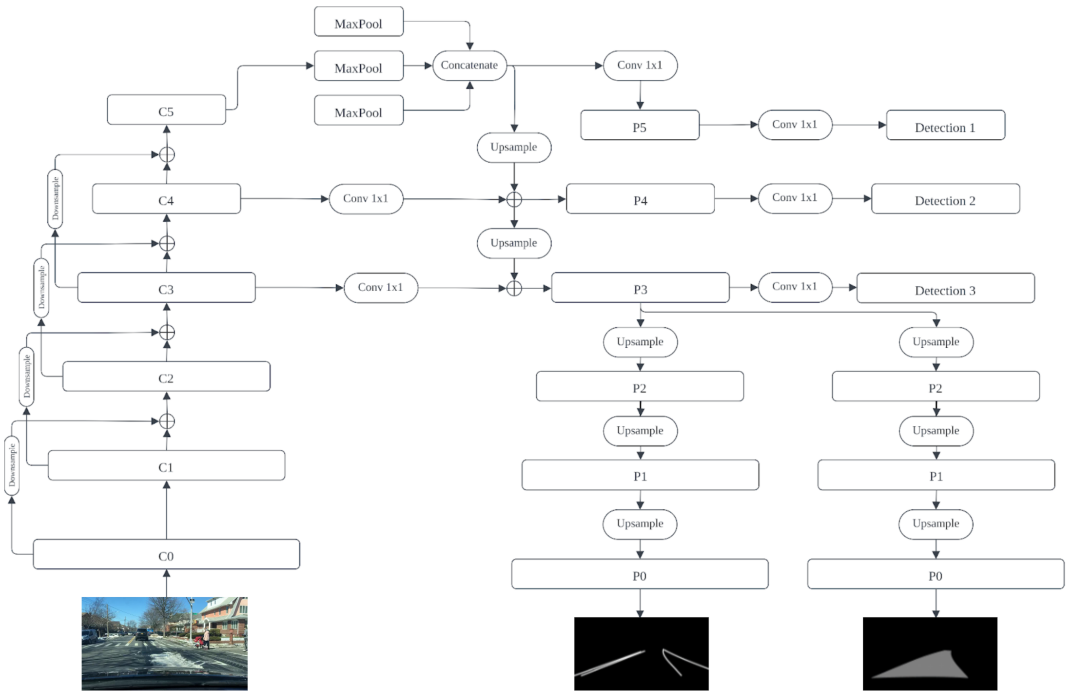


Fig. 10: Model Structure for Multi-task Learning

B. Backbone

1) ResNet: The core hypothesis behind the ResNet paper was that if we have a deeper neural network, it should perform better if not the same compared to a shallower neural network [4]. We

define a deeper neural network to have more convolutional layers and the opposite for shallower networks. The idea behind the hypothesis was that the later layers of the network should ideally learn the identity function if it does not decrease the loss function or will learn weights that will result in a decrease in the loss function. However, this was not the case in practice where we would often see a degradation in the accuracy of the network when more layers are introduced.

The solution to this problem is the introduction of the residual, or skip connections outlined in the Deep Residual Learning for Image Recognition (ResNet) paper [4]. The core rationale behind residual connections is that we have a residual block, which is a set of convolutional layers. Before stepping into the block, we save the input tensor and perform element-wise addition with it and the output tensor from the block. This results in an explicit command to the model: learn the identity function if this set of layers do not result in a reduction in the loss function [4]. A conceptual residual block is outlined in Figure 11, where the input x is "skipped forward" to the output of the block in order to achieve the identity mapping.

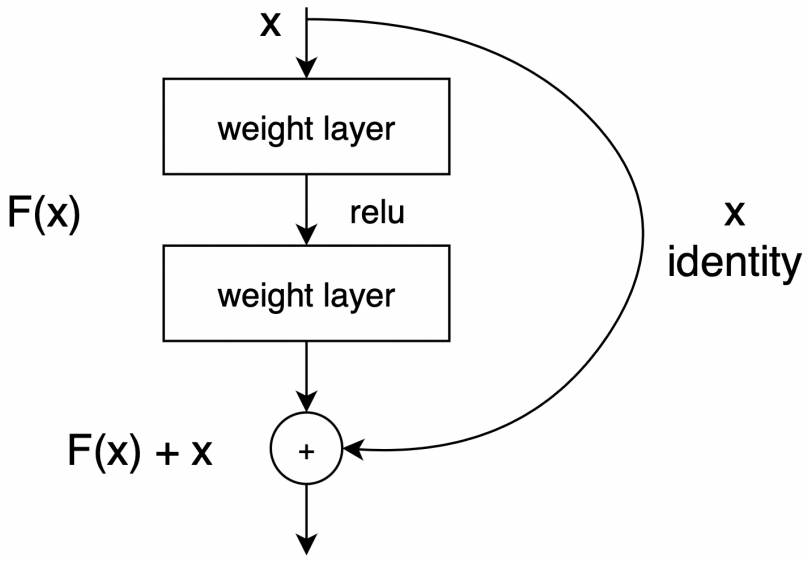


Fig. 11: Residual Block

The use of skip connections within a deep neural network maintains the accuracy increase achieved from the large layer depth, while avoiding the high training error caused by accuracy

degradation [4]. This achievement is what gives ResNet the ability to train on deeper networks without accuracy degradation. In the YOLOv3 model we developed, the concept of residual connections is utilized to combat the accuracy degradation issue that came from the increase in network depth.

2) Darknet-53: The Darknet-53 architecture shown in Figure 12 is a convolutional neural network that acts as a backbone for our model. As an improvement to Darknet-19, Darknet-53 was first introduced in the YOLOv3 paper for its feature extractor with the use of residual connections and its increase of convolutional layers to 53 to increase the accuracy of the model [5]. Table III shows slight modifications to what was introduced in the YOLOv3 paper. We decided to set our input image size to 384×640 to maintain the aspect ratio and allow the resolution to be divisible by 32. The network successively executes a series of outlined "Residual Blocks" , which contain 3×3 and 1×1 convolutional layers and a residual connection between the block's input and output. This process is repeated by the number of repeats specified for each sequence. The outputs of the third, fourth, and fifth sequences – c3, c4, c5 – are sent to the Feature Pyramid Network (FPN) to combat the loss of spatial information.

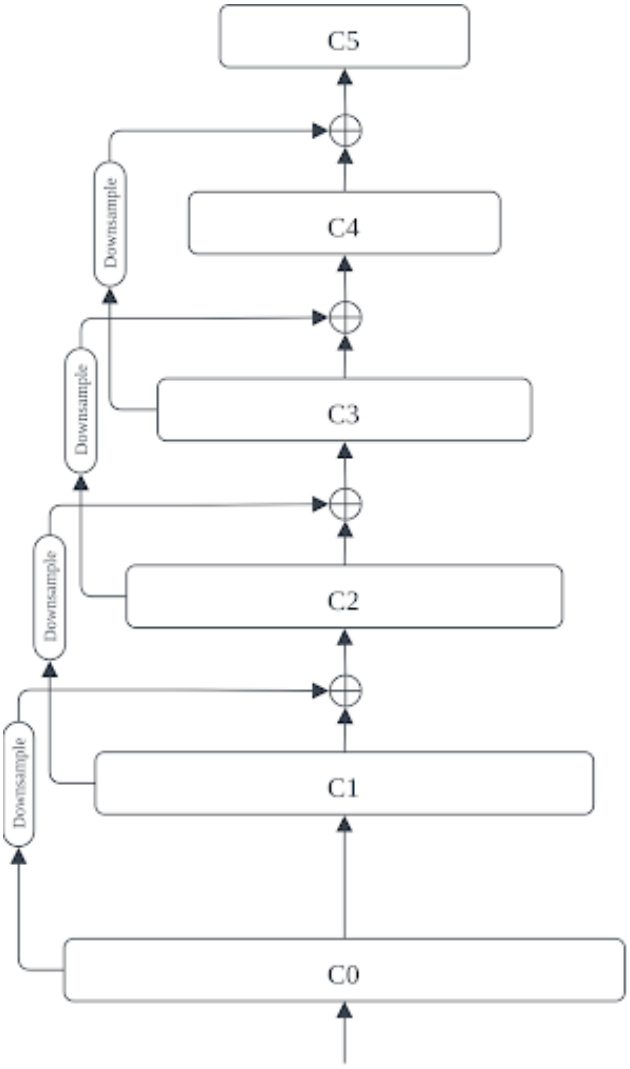


Fig. 12: Darknet-53 Architecture

	Type	Filters	Size	Output
	Convolutional	32	3 x 3	384 x 640
	Convolutional	64	3 x 3 / 2	192 x 320
1x	Convolutional	32	1 x 1	
	Convolutional	64	3 x 3	
	Residual			192 x 320
	Convolutional	128	3 x 3 / 2	96 x 160
2x	Convolutional	64	1 x 1	
	Convolutional	128	3 x 3	
	Residual			96 x 160
	Convolutional	256	3 x 3 / 2	48 x 80
8x	Convolutional	128	1 x 1	
	Convolutional	256	3 x 3	
	Residual			48 x 80
	Convolutional	512	3 x 3 / 2	24 x 40
8x	Convolutional	256	1 x 1	
	Convolutional	512	3 x 3	
	Residual			24 x 40
	Convolutional	1024	3 x 3 / 2	12 x 20
4x	Convolutional	512	1 x 1	
	Convolutional	1024	3 x 3	
	Residual			12 x 20

TABLE III: Darknet-53

C. Spatial Pyramid Pooling

The Spatial Pyramid Pooling (SPP) block was introduced to increase the receptive field of the network. The SPP block illustrated in Figure 13 utilizes three max-pooling layers at different filter sizes while retaining the input resolution and then concatenating the three different outputs together. This allows the model to extract the multi-scale local region features which are not captured from the Feature Pyramid Network [6]. Furthermore, each of the large filter sizes allow the filter to capture a larger portion of the image which is already low in resolution. The concatenation in the SPP block results in four times the amount of features for the model to use without introducing too many parameters, which would lower the latency of the model.

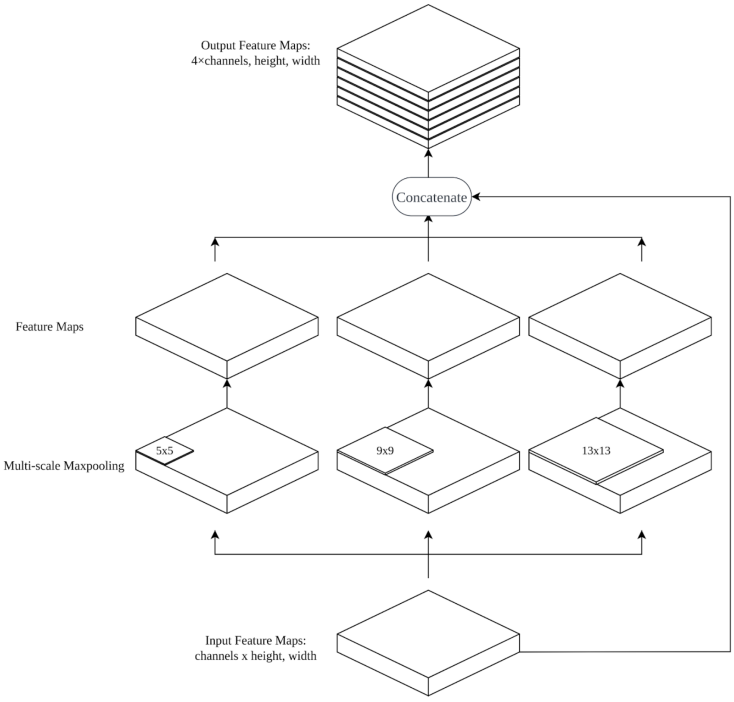


Fig. 13: Spatial Pyramid Pooling

D. Feature Pyramid Network

A major issue with YOLOv1 was that its object detection struggled to detect smaller scale objects in the image. To combat this issue, our multi-task model implements a Feature Pyramid Network (FPN) to allow for object detection at different scales by reintroducing the spatial information lost from the backbone's downsampling. The objective of the FPN is to take a low resolution feature map from the backbone like the C5 layer and merge it with a higher resolution like C4 [7]. This merging operation is done through an upsampling of the C5 layer, then concatenating it with the C4 layer, and then applying convolution to merge the features from both layers. This operation results in a feature map containing the high-resolution spatial information retained prior to the downsampling operation as well as the high-density semantic information obtained from the downsampling. This process is repeated three times at different scales to generate the three necessary feature maps for the scale detections.

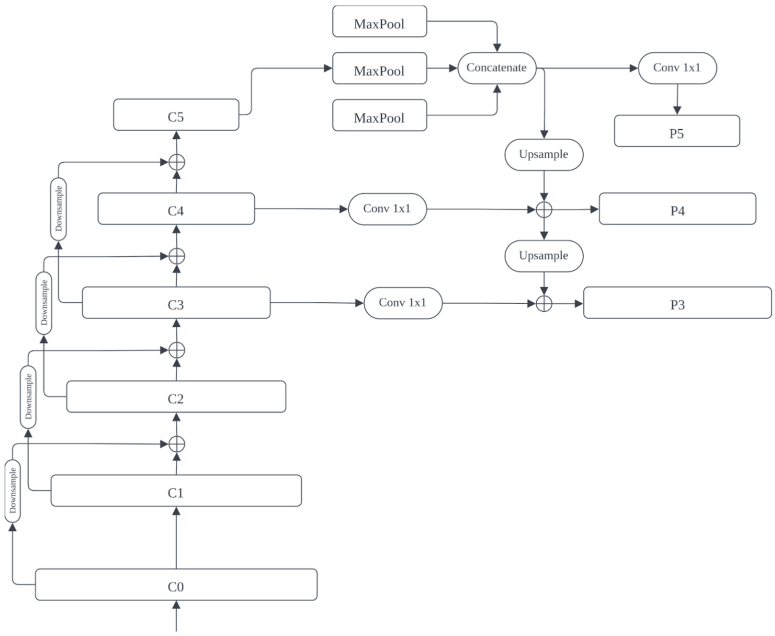


Fig. 14: FPN Architecture

E. Decoders

Our model utilizes two sets of decoders: an object detection decoder and two segmentation decoders. The object detection decoder is based off the anchor-based YOLOv3 decoder, where we are detecting objects at three different scales. The two segmentation decoders are based off last semester's work on U-Net where we quickly upsample the last layer from the Feature Pyramid Network until we restore the input resolution.

The shift from YOLOv1 to YOLOv3's decoder introduced anchor boxes and detections at different scales [5]. Detecting at three scales addresses the problem we had in YOLOv1 where the model struggles to detect smaller objects in the dataset. As discussed in the YOLOv1 results section, we believe this is due to the model performing detection on only the last feature map, which is heavily downsampled due to it being $\frac{1}{16}^{th}$ of the input resolution. In the YOLOv3 decoder, we are performing the detection at $\frac{1}{32}^{nd}$, $\frac{1}{16}^{th}$, and $\frac{1}{8}^{th}$ of the input resolution. This allows the input feature maps inputted to the decoders to capture each of the three different object sizes contained in our dataset.

The idea behind anchors is to generate prior anchor boxes, which attempt to encapsulate the most common object sizes within the dataset [8]. The addition of anchor boxes solves two

fundamental problems from YOLOv1 [9]. First, in YOLOv1 only one object can occupy one grid cell because we assigned each grid cell to be responsible to one specific object if that object is within the grid cell. With the introduction of anchors, an object is assigned to a grid cell and anchors, allowing up to three objects to be within the same grid cell because we have three anchors, one at each scale. Second, this simplifies the training process because the model learns the offset it needs to adjust the anchor boxes rather than learning the exact location of the objects.

F. Loss Functions

1) Object Detection: To quantify the model's object detection accuracy each after iteration, the YOLOv3 architectures uses two loss functions: Focal Loss and Complete-IOU loss [5].

Focal loss in Equation 10 accounts for accuracy with respect to the classification results. It is an augmentation of standard cross-entropy loss that includes a modulating factor to account for class imbalances in training. For example, the modulating factor will de-emphasize loss on common and easy object classification tasks, and increase emphasis for tasks that appear less frequently, and are thus more difficult [10]. It will effectively balance the loss function based on the frequency of certain classification tasks to ensure that the model dedicates equal learning power to each class.

$$FocalLoss(p_t) = -\alpha_t(1-p_t)^{\gamma}log(p_t) \tag{10}$$

Where a_t is the class balancing factor, γ is the focusing parameter, and p_t is the class probability p if the class matches the ground truth, and $1-p$ if the class does not match the ground truth.

Complete-IOU loss defined by Equation 11 accounts for accuracy with respect to bounding box detection. It is an augmentation of standard IOU loss which calculates the intersection-over-union between the model's detected bounding boxes and the ground truth data. In addition to computing IOU, Complete-IOU introduces two more factors: the distance between the centers of the detected bounding box and the ground truth, and the consistency of the bounding boxes' aspect ratios [11].

$$C_{IOU} = 1 - IOU + \frac{\rho^2}{diag^2} + \frac{v^2}{1 - IOU + v} \tag{11}$$

Where IOU is the intersection over union of the detection and the ground truth, ρ is the distance between bounding box centers, $diag$ is the diagonal length of the convex of bounding boxes, v is the aspect ratio consistency, w_{gt}, h_{gt} is the width and height of ground truth, w_d, h_d is the width and height of the detection, and $v = \frac{4}{\pi^2}(tan^{-1}(\frac{w_{gt}}{h_{gt}}) - tan^{-1}(\frac{w_d}{h_d}))^2$.

2) Segmentation: For our segmentation tasks we also used focal loss. Focal loss is needed rather than standard binary cross entropy loss due to the presence of class imbalances between lane types. Since binary cross entropy loss sums the cross entropy between each predicted and ground truth pixel, lane types containing more pixels are more likely to have a higher loss. This biases the network towards detecting these lanes, resulting in poor performance for lanes with a smaller number of pixels. To remedy this issue, focal loss introduces a vector of weights for each class of lane. Higher values of alpha are chosen for those lane classes with a lower number of pixels to enable equal contribution to the loss as lanes with larger number of pixels. In addition to the class imbalance across lane types, each lane type also has a class imbalance between pixels that are and are not lanes. There are many more pixels that are not lanes than are lanes. To remedy this problem, a modulating factor γ is added to de-emphasize loss on the background pixels, and increase emphasis for the lane pixels.

G. Metrics

1) Segmentation: For each image, a vector of eight IOU values is calculated for each type of lane present in the image. If the lane type is not presented in the image, -1 is placed in its corresponding entry. The vectors are transposed and added as rows to a matrix. The mean IOU is calculated by averaging over each column of the matrix. Entries that are -1 are discarded from the averaging.

H. Results

Our multi-task model was only trained on 50 epochs due to the time constraint at the end of the semester, however we were able to get some promising results as shown in Table **??**.

The metrics we used for the three tasks are mean average precision with IOU threshold of 50 for object detection and mean IOU for both the segmentation tasks. The reason for the low lane segmentation IOUs is that the model predicts relatively thicker lane lines compared to the groundtruths which are typically 1 to 2 pixels wide. When we look at the individual classes, we can see that the accuracy is much higher for single white lanes and road curbs compared to any other lane types. This is due to the frequency of the classes within the dataset, as these two classes are much more frequent than any of the other classes. When comaparing our model's performance to other model in the BDD100k model zoo, our model is lacking in the accuracy department and this will need to be address in the future. The results of the model can be further contextualized through a visualization of the model outputs shown in Figure 15 - 18

	mAP50 (%)
Object Detection	42.18
Segmentation Class	**mIoU(%)**
Drivable Area	54.42
Lane Predictions (all)	10.4
Single White Line Lanes	25.3
Road Curbs	18.8
Cross Walks	7.6
Single Yellow Lines	6.4
Double Yellow Lines	2.1
Double White Lines	2.0

TABLE IV: Results of multitask model

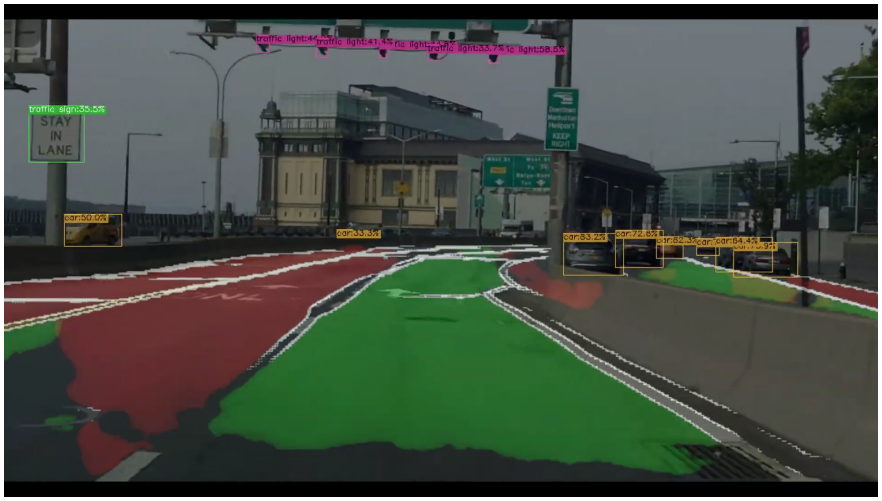


Fig. 15: Results of multitask model

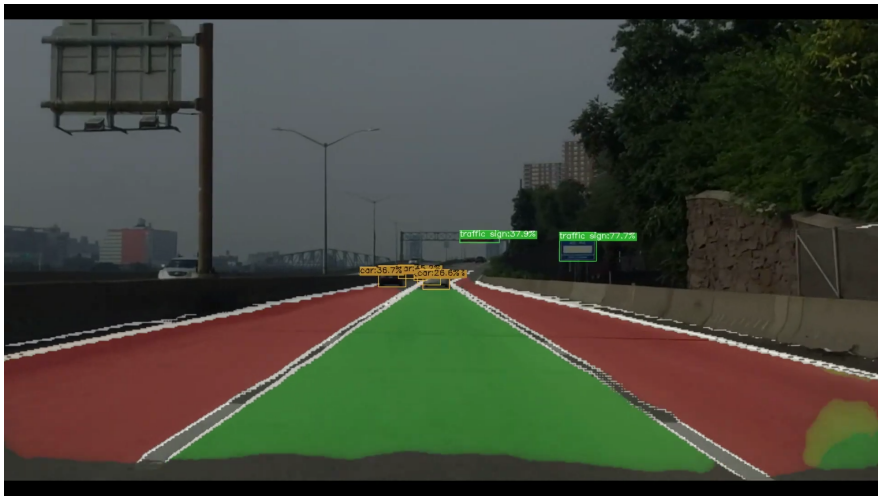


Fig. 16: Results of multitask model

Fig. 17: Results of multitask model

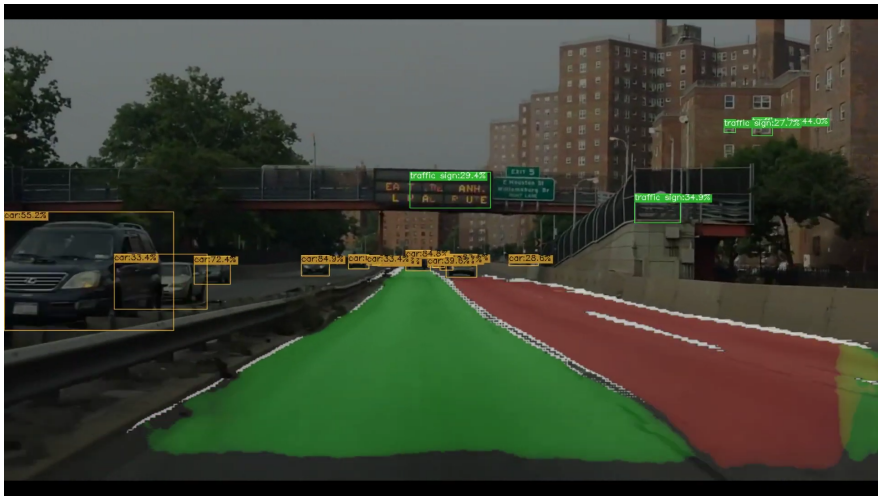


Fig. 18: Results of multitask model

6. Future Works

Despite making great progress throughout this semester, there are still multiple areas that we can improve upon. We discussed in the results section that the performance of our model is lacking compared to the current state of the art models. To address this, there can be multiple changes to the backbone, one in adtopting an entirely differnet backbone or introducing cross stage partial networks seen in YOLOv4 [12]. On the efficiency side looking at works like MobileNet [13] or EfficientNet [14] can be avenues we can explore into. The feature pyramid can also be changed for a Bidirectional Feature Pyramid or adopt attention based feature aggregation. For the decoders, working on adopting an anchor-free object detection decoder is an avenue that can be explored. Moreover, the BDD100k dataset provides multiple temporal data which we believe can have an impact on the model. This could result in utilizing semi-supervised learning model.

7. Conclusion

The results in this paper demonstrated a prototype model that is able to perform object detection, lane detection, and drivable area segmentation. Object detection had a mean average precission of 42.18%, lane detection had a mean IOU of 10.40%, and drivable area segmentation had a mean IOU of 54.42%. The lane IOU is more reasonable considering that the area covered

by a lane line is quite small while the predictions of our model are relatively thicker. These are certainly not state of the art results, however, our model contains fewer parameters than running three separate networks for the three tasks. Additionally, we found that the work done in modifying the YOLOv3 network was a success in allowing the architecture to be able to perform detection alongside segmentation. In summary, the final lane detection model was able to run all the three tasks we set out to achieve in one single forward pass.

REFERENCES

[1] F. Yu, H. Chen, X. Wang, W. Xian, Y. Chen, F. Liu, V. Madhaven, and T. Darrell, "Bdd100k: A diverse driving dataset for heterogeneous multitask learning," 2018.

[2] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," May 2016. [Online]. Available: https://arxiv.org/abs/1506.02640v5

[3] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation." [Online]. Available: https://arxiv.org/abs/1505.04597

[4] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," Dec 2015. [Online]. Available: https://arxiv.org/abs/1512.03385

[5] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," Apr 2018. [Online]. Available: https://arxiv.org/abs/1804.02767

[6] Z. Huang, J. Wang, X. Fu, T. Yu, and Y. Guo, Mar 2019. [Online]. Available: https://arxiv.org/pdf/1903.08589.pdf

[7] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature pyramid networks for object detection," Apr 2017. [Online]. Available: https://arxiv.org/abs/1612.03144

[8] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," Jan 2016. [Online]. Available: https://arxiv.org/abs/1506.01497

[9] J. Redmon and A. Farhadi, "Yolo9000: Better, faster, stronger," Dec 2016. [Online]. Available: https://arxiv.org/abs/1612.08242

[10] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," Feb 2018. [Online]. Available: https://arxiv.org/abs/1708.02002

[11] Z. Zheng, P. Wang, W. Liu, J. Li, R. Ye, and D. Ren, "Distance-iou loss: Faster and better learning for bounding box regression," Nov 2019. [Online]. Available: https://arxiv.org/abs/1911.08287

[12] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, "Yolov4: Optimal speed and accuracy of object detection," Apr 2020. [Online]. Available: https://arxiv.org/abs/2004.10934

[13] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," Mar 2019. [Online]. Available: https://arxiv.org/abs/1801.04381

[14] M. Tan and Q. V. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," Sep 2020. [Online]. Available: https://arxiv.org/abs/1905.11946

[15] D. Wu, M. Liao, W. Zhang, X. Wang, X. Bai, W. Cheng, and W. Liu, "Yolop: You only look once for panoptic driving perception," Mar 2022. [Online]. Available: https://arxiv.org/abs/2108.11250

8. Appendix

A. *Pume Tuchinda*

This semester was my third and last semester on the Lane Detection team. Coming into this semester I had a clear idea of what I wanted to do for the team, as I was continuing the project from last semester. This semester I wanted to look into object detection since our dataset already had all the necessary labels for the task and also expand last semester's work to also segment drivable areas. However, drawing inspiration from my time during Fall 2021, I knew I wanted to create a network that would be able to perform all three tasks with one forward pass rather than having three separate networks. Since this is quite an ambitious goal I set, I dedicated the first part of the semester to catching up the members on the work that has been done from previous semesters and also have them learn the fundaments for Machine Learning. This came in the form of me creating a YOLOv1 template containing the necessary functions and the explanations of each function needed to implement and train YOLOv1. I sent the template to all the members for them to do as an assignment for them to gain an understanding of YOLOv1 and also familiarize them with PyTorch. I also have a fully running YOLOv1 implemented and trained for them to check if there implementation was correct.

Following the onboarding assignment was researching on how to design and implement a neural network that would be capable of doing all three tasks. This came in understanding the structure of object detectors and seeing how I could modify the YOLO model while incoporating last semester's work. I drew alot of inspirations from last semester's paper and also from YOLOP [15] to fully design the network we are using now. Following this I started delegating tasks to the members and continuosly help explain the concepts and help with debugging. I fully wrote out the SPP block and worked on fully integrating everyones code into a fully functioning system. This came in having to rewrite the loss function, writing the training and evaluatation script. Once everything was working the last few weeks was spent on getting all the evaluation metrics and also the postprocessing of the model for displaying the outputs.

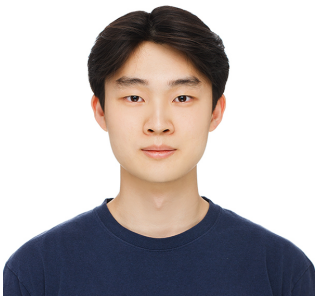
B. William Stevens

This semester was my second semester with VIP-IPA but my first experience on the lane detection team. Last Spring provided me with a strong background in image processing, which was helpful for the development of my understanding of deep learning this Fall. My first month on the Lane Detection team was spent doing various learning, researching, and experimenting to get myself comfortable with the concepts of machine learning so that I could begin to contribute towards the team's goal of writing software to perform object detection and classification on dashboard images. Some examples of the onboarding tasks I completed include watching lectures from online courses, reading papers in the IEEE and Arxiv databases, and implementing simple machine learning tutorials provided by the PyTorch organization. For the remainder of the semester, I spent most of my time reading research papers about YOLO and other related works, and eventually implementing such concepts into our code base to help achieve our goal.

Once I was familiar with the fundamentals of machine learning, I joined the others in the implementation of YOLOv1. For each of the files in YOLOv1, we collaborated and combined our individual work to create the model file, the dataset loader, and the loss function. The model handled the creation of convolutional layer sequence and specified the parameters to use and operations to execute within each layer, the dataset loader parsed through our dataset and converted the data to be inputted to the model, and the loss function computed how much the model should be penalized for its predictions at each iteration.

After the team was finished with our implementation of the YOLOv1 architecture, we began working to unify multiple object detection and classification tasks into one network. My tasks for this goal included researching on the YOLOv3 architecture, ResNet, anchor boxes, and loss functions, as well as writing the code for the Darknet-53 model, implementing a K-means algorithm to determine anchor box dimensions, and writing the object detection loss functions. In the writing of this report, I was tasked with writing the introduction, YOLOv1 architecture, YOLOv1 loss function, ResNet, and YOLOv3 detection loss function sections.

C. Mingyu Kim



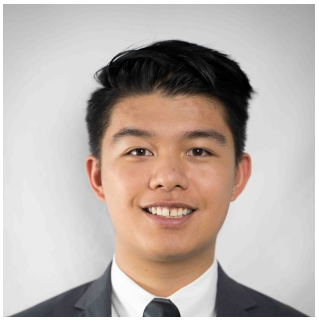
This was my first semester in VIP Lane Detection team. I had no prior experience in image processing and deep learning, but under Pume's leadership I was able to pick things up quickly. In the beginning of the semester, I watched Professor Delp's image processing and analysis tutorial and implemented some of the basic fundamentals of image processing. I implemented a simple function for average and median filters, Sobel edge detection, and RGB to grayscale using three different methods — lightness, average, and luminosity method.

After that, I jumped into implementing YOLOv1. Initially I struggled with understanding the concepts of CNN and applying the information from research papers into code, and I had to fill in the blanks through trial and error. Although I had experience in Python, PyTorch and its implementation such as training and custom dataloader was completely new to me. I worked on implementing the network architecture and loss function of YOLOv1, along with IOU to filter one of the two bounding boxes for prediction.

Towards the end of the semester, the team shifted to implementing multi-task learning. Referencing to the code I wrote for YOLOv1, I built Darknet-53, the backbone of our model, with the addition of residual blocks. I was able to acquire better understanding of kernel size, stride and padding for a desired output, as well as how tensors are shaped and manipulated. After that, I worked on non-max suppression and evaluation metrics to solve precision and recall for mAP.

Overall, I was lucky to be in the Lane Detection team and was able to learn a lot about image processing and CNN. I'm excited to see what's to come next in the following semester and learn more about this field.

D. Justin Chan



This was my first semester in the Image Processing Analysis lane detection team. Coming into the semester, I had taken courses such as ECE 20875 and ECE 49595 which have taught me the foundations of data science. This semester, I had the opportunity to apply my previous course work and continue to learn in detail image processing and computer vision. With the great help of Professor Delp, Professor Zoltowski, and my fellow teammates, I was able to get a better grasp of image processing especially in the domain of autonomous driving vehicles. At the beginning of the semester, I leveraged Professor Delp's videos on the fundamentals of Image Processing to gain a broad understanding of concepts that are widely used in the field. In the lectures, I learned about convolutions, sobel-edge detection, and otsu thresholding. From here, I spent the next two weeks watching videos made by MIT and various industry leaders to learn about deep learning, neural networks, and computer vision which were extremely crucial for this semester's project. Moving from this, I started to implement the YOLOv1 architecture used traditionally for object detection. Through implementing the model and the loss function, I was able to learn about convolutional neural networks as well as the principals behind loss functions.

Through implementing YOLOv1, the team and I started to see the limitations of the framework where it would struggle to detect object at diferent scales. Through this, we pivoted to more updated versions of YOLO which included Feature Pyramid Networks used to detect object at different scales. Concurrently, the team was moving in the direction of using 1 neural network for drivable area segmentation, object detection, and lane detection. For this multi-tasked learning model, I implemented the Feature Pyramid Network as well as IOU Loss to quantify the error of our model. I was able to gain a better understand the tradeoff between semantically rich feature maps and spatial information.

Overall, I had a great experience this semester and learned a tremendous amount about image processing. I look forward to seeing the future works of the Lane Detection team.

E. Daniyaal Rasheed

This was my first semester in the Image Processing & Analysis VIP team. I came in with knowledge of the structure and training of fully connected neural networks, 1D convolution, and multivariate calculus. This semester, I learned about convolutional neural networks, YOLO, and a few loss functions used for segmentation and object detection, like binary cross entropy, focal loss, and dice loss.

Earlier in the semester, we focused on establishing fundamental knowledge of machine learning and digital signal processing. I implemented a 2D convolution function to apply the impulse responses of sobel edge detection, gaussian filtering, and other basic image processing operations. Then we focused on learning fundamental machine learning concepts. Since I was familiar with neural networks, I focused on understanding convolutional neural networks, new activation functions, such as ReLU, and the caveats of each during the training of a neural network by watching Stanford lectures on the subject.

When our team knew our final architecture would include the backbone of YOLOv3 to extract features from the images, I began understanding and implementing YOLOv1 to gain a more solid footing into its more complex successor. I implemented the model and dataloader of YOLOv1 in PyTorch. This required me to read the YOLOv1 research paper and implement tutorials from the PyTorch documentation.

In the later part of the semester, we began building the multi-task network. My contributions to the final model were the Feature Pyramid Network, focal loss for segmentation, and mean IOU segmentation evaluation metric. Implementing and understanding FPN required me to rely on my digital signal processing knowledge from ECE 438 to understand why certain design decisions were made, such as the need for low-pass filtering after upsampling to remove anti-aliasing effects.

I will be joining next semester and continuing with this project. To improve myself I plan on reading and solving the problems out of my digital signal processing and probability textbooks. I also plan on reading research papers in the field of computer vision and machine learning and playing around with our existing model to see how our team can improve it next semester.