VIP-IPA: Lane Detection Team

Final Report

Spring 2023

William Stevens, Mykhailo (Misha) Tsysin,

and Xingyan (Ian) Li

CONTENTS

1. Abstract

With the prevalence of autonomous vehicles, the computer vision algorithms utilized for autonomous driving must be robust and accurate to assess road features through images captured in real time. In our previous work, we designed a multi-task neural network model according to the YOLO architecture and trained on the BDD100k dataset, to give detections and classifications of objects, as well as segmentations and classifications of lane lines, in an image. While last semester's model was successful, there was room for improvement. So, we wanted to explore potential optimizations to the network to achieve better results. This process included the research, development, and testing of alternative strategies for the many components of our multi-task model. This will be achieved through modifying last semester's model according to the fourth edition of the YOLO architecture. The modifications to the model will include implementing an augmented feature extractor and an improved feature aggregation network through the works done in Cross-Stage Partial Networks and Path Aggregation Network. The feature extractor is designed to greatly reduce the number of duplicate gradient computations, and the improved feature aggregator prevents the significant loss of spatial information that occurred in our previous implementation, while still allowing the model to effectively merge shallow and deep information to achieve better predictions. These developments will allow our multi-task model to perform the three vision tasks for autonomous driving at a lower computation cost while also achieving higher accuracy, thus creating an improved version of the 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. We aimed to build on the works of the Lane Detection teams from Fall 2021, Spring 2022, and Fall 2022, which utilized the You Only Look Once (YOLO) framework, the U-Net architecture, and a combination of YOLOv3 and U-Net respectively. The main objective of the team this semester was to develop various augmentations from YOLOv4 to the Fall 2022 model and improve the model's computation cost and detection accuracy. Our plan was to follow in the footsteps of the original developers of YOLO by working through the next version of the framework. Thus, the ultimate goal for this semester was to research all of YOLOv4's modifications, and incorporate them into the project to better 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 [1] 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]

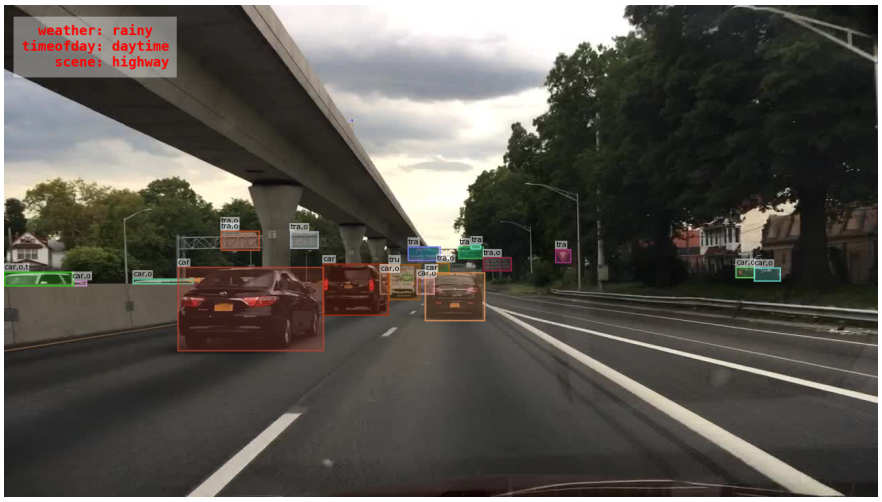

Fig. 1: Object detection visualization during the daytime. [1]

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

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

4. NEURAL NETWORKS AND YOLO

In this section, we'll provide a quick guide into the theory of deep learning and it's applications for computer vision. We'll start by giving an overview of a neural network stricture and discuss the common architectural blocks for convolutional neural networks (CNNs). We'll introduce activation functions, loss and methods for neural network training, such as gradient descent and it's practical application - gradient descent.

A. Introduction

Within the machine learning field, the demand for describing complicated mathematical relations to a computed has been steadily increasing. The general purpose of any machine learning algorithm is to determine an unknown quantity from available data. This is relevant to our work, as our dataset contains the positions of objects and other features on the road, in the form of matrix representations. For extracting feature information from the data, the general problem is as follows:

1) Select a certain family of functions $f_\theta : X \to Y$, where θ - is a parameter list that determines the function behavior, X - input space, Y - output space.
2) Find a set of suitable parameters $\hat{\theta}$ such that they are optimal for making the predicted values $f_{\hat{\theta}}(x_i)$ approximate the true values y_i. Let $\mathbf{x} = \{x^1, x^2, ..., x^N\}$ denote a set of input data points (images) and let $\mathbf{y} = \{y^1, y^2, ..., y^N\}$ be the corresponding labels (coordinates of object positions). N is the total number of training examples.

The simplest example of such a function can be the standard linear relation: $f_\theta(x) = ax + b$. In this case, formally, $\theta = \{a, b\}$. Nearly all other machine learning methods can be described with respect to this relation. However, the problem is that many of relationships present in our multidimensional world can not be represented with simple linear functions, life is not linear, polynomial, or exponential. To account for this, artificial neural networks have been designed and constructed to learn such complex relationships by using the linear relation in a flexible manner. This feat is accomplished by following the basic structure shown in Figure 4. The network shown is a simple neural network with one hidden layer. Note that it has input values x_1, x_2, x_3, which are passed through a series of computations to arrive at the output values y_1

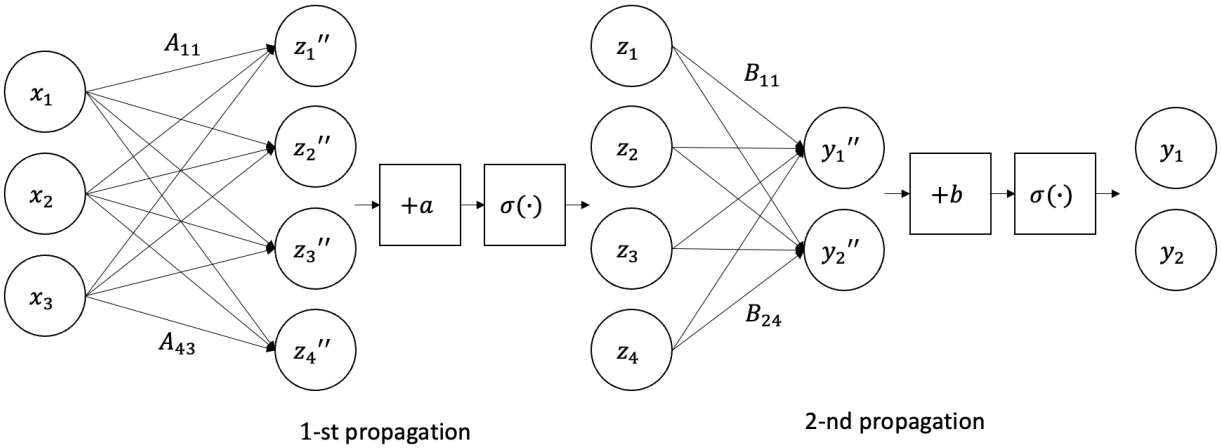

Fig. 4: Neural Network example: one hidden layer.

and y_2. Since the input vector X to the network consists of three numbers, we denote $k_0 = 3$ as input size of the network. Similarly, the output vector Y has two values so we refer to it as l_2 (2 being the index of the last layer in the network). Each of the values A_{ij} represents a certain weight parameter, which is multiplied by an input value x_j and added to z_i'' – an element of subsequent layer in the network. This operation can be viewed in mathematical terms as:

$$z'' = Ax$$

After that, to allow for the most general case, we add a bias term a. What follows next is what gives neural networks their power. The function $\sigma(\cdot)$ is called an *activation* function. This function introduces non-linearity to the model, which allows for the most complete set of possible relationships. The final result for the second layer (indexed as 1) is:

$$z = \sigma(Ax + a)$$

We repeat this process once more with different weights and biases (B, b respectively) and obtain the final output vector y.

B. The Activation Function

The activation function is a very important component of neural networks. The performance of a network can be severely affected by the choice of activation function. Here we describe several widely used functions:

Sigmoid and Tanh:

$$\sigma(x) = \frac{1}{1 + \exp(-x)}, \quad \tanh(x) = \frac{\exp(x) - \exp(-x)}{\exp(x) + \exp(-x)}$$

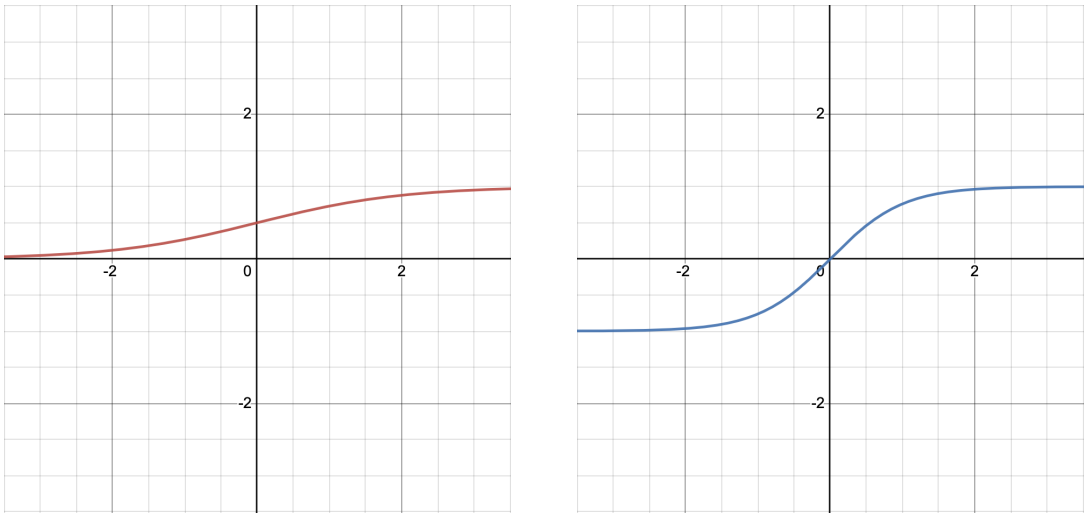

Fig. 5: Sigmoid function (left) and tanh fucntion (right)

Sigmoid was one of the first functions used for deep learning neural networks. Sigmoid essentially clips the values between 0 and 1, preserving a suitable range for layer values. One of the drawbacks of this function is the fact that its gradients far from 0 are prone to approach zero, which can have adverse effects on model performance. Hyperbolic Tanget, or Tanh, has similar behavior to sigmoid, but with output range from -1 to 1.

ReLU and Leaky ReLU:

$$ReLU(x) = \begin{cases} x: & x > 0 \\ 0: & x \leq 0 \end{cases} \qquad LeakyReLU(x) = \begin{cases} x: & x > 0 \\ \alpha x: & x \leq 0 \end{cases}$$

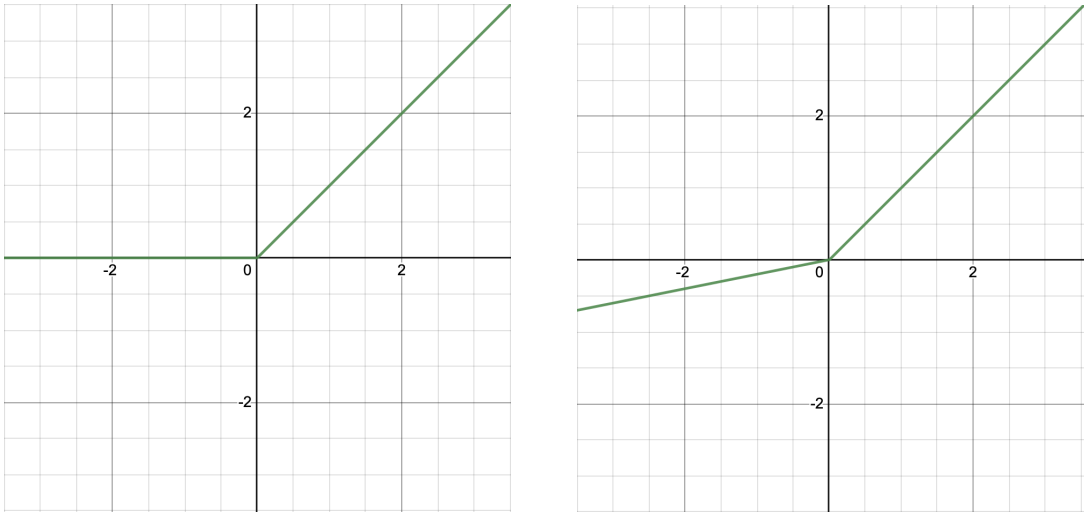

Fig. 6: ReLU function (left) and Leaky ReLU fucntion (right)

Where $\alpha < 1$ is a certain small coefficient. The Rectified Linear Unit (ReLU) and its variation Leaky ReLU have gained a lot of popularity, particularly for computer vision tasks, because of the ease of their implementation and surprising effectiveness. However, there is a problem with both of these functions. Their gradient computations require computationally expensive approximations, as they are not continuously differentiable at the origin. Additionally, the ReLU function always has a gradient of zero for negative inputs, an issue which was solved with the creation of the Leaky ReLU variation.

C. Loss

In order to effectively train a neural network to artificially "learn" from its mistakes, we must have some measurable criteria of how well the predicted labels represent the true values. Although the same results can be derived from estimation theory, using, for instance, MAP or MSE estimators, here we directly introduce the loss function. In simple terms a loss function aggregates all penalties from incorrect predictions that the network produced. A simple example of a loss function is the mean-squared-error (MSE) loss:

$$L_{\mathbf{x},\,\mathbf{y}}(\theta) = \frac{1}{N}\sum_{n=0}^{N} ||y^n - f_\theta(x^n)||$$

To improve model accuracy, it is necessary to minimize the loss function, as this will correspond to the best results. For this, the optimal value of θ should satisfy:

$$\hat{\theta} = \arg\min_{\theta} L_{\mathbf{x},\,\mathbf{y}}(\theta)$$

MSE loss is one of the more basic loss functions, and generally the main problem that arises with its use is the existence of ill-conditioned gradients. When the output of the model is far away from the true value, the model learns quickly, but when it comes to fine-tuning the weights, the flat area of the quadratic term makes it harder to further improve. Moreover, with this type of loss it's hard to prioritize different learning objectives. All such details should be taken into account when designing a loss function for a particular application.

An example of another common loss function is cross-entropy (CE) loss. This loss is a solution to the same estimation problem, but for when the output of the system is a certain discrete class rather than a continuous parameter. For instance, to determine whether a certain object is a pedestrian or a vehicle. The CE loss is defined as:

$$L^{CE}_{\mathbf{x},\,\mathbf{y}}(\theta) = \frac{1}{N}\sum_{n=0}^{N}\sum_{c=0}^{C} t_c^n \log(f(y^n)_c)$$

Where $f(y)_c = \frac{e^{y_c}}{\sum_{c'=0}^{C} e^{y_{c'}}}$ is a softmax activation, the main purpose of which is to convert the general values of the output layer y into class probabilities (which sum up to 1).

D. Gradient Descent

To minimize the loss function described in the previous section, we employ a common optimization method - gradient descent. The gradient descent algorithm can be summarized as "moving downhill until reaching a local minimum." See Figure 7 for a visual representation.

What is shown in Figure 7 is the graph of the loss function with respect to the learnable parameters θ (assume we only have one parameter for this 2D example). Suppose the current value of θ corresponds to the black point on the graph. The goal is to reach the minimum point, which is the red point. When at the initial point, there is not much information about the global structure of the function. However, we can compute the local parameters, such as the gradient, to try and reach the minimum point. This can be achieved because the gradient can be used to

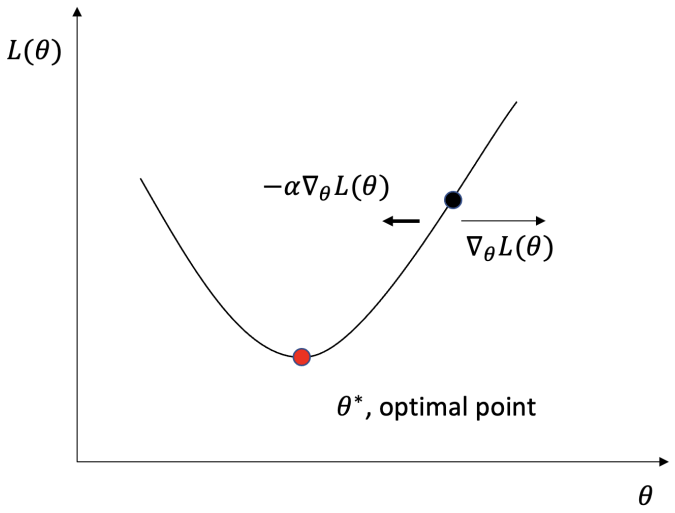

Fig. 7: Gradient Descent Visualized.

find the direction of steepest descent, which will eventually lead towards the minimum point. Making small steps towards the minimum of the curve will eventually reach the optimum point. Formally, the gradient descent algorithm can be summarized as:

1) Find gradient $\nabla_\theta L(\theta)$
2) Assign $\theta := \theta - \gamma\nabla_\theta L(\theta)$
3) Repeat until convergence

This two-dimensional gradient descent concept can be generalized to multiple dimensions, which will be necessary if there are many learnable parameters θ. Using this algorithm in this simple form often results in failure. But with the addition of various tricks such as stochastic gradient descend (SGD), or adaptive momentum (ADAM) it can achieve great results.

E. Backpropagation

Gradient descent is intuitive visually, but complicated mathematically. The main difficulty comes from computing the gradient itself. This is achieved by using the concept of backpropagation.

The backpropagation algorithm is in reality nothing more than a simple chain rule applied to all of the computations in our network. Fortunately, modern software packages often record

the changes of all variables when performing any mathematical operations. This results in the creation of a computational graph. For example, the example model shown in Figure 4 will result in the computational graph shown in Figure 8.

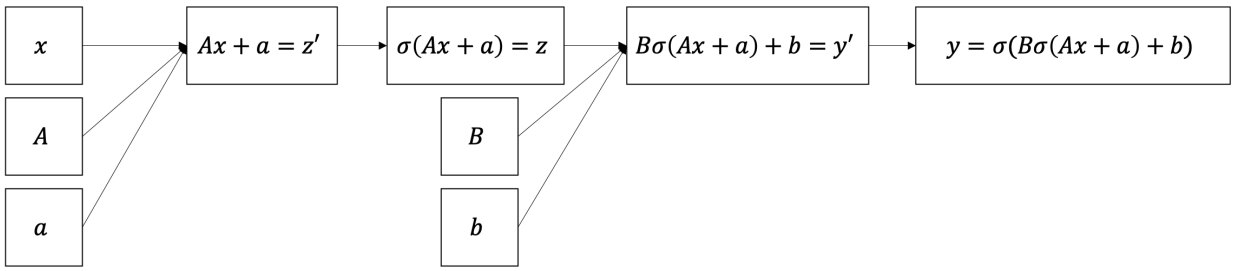

Fig. 8: Backpropagation Computational Graph.

The aggregate operation here is $y = \sigma(B\sigma(Ax + a) + b)$ and the learnable parameters are $\theta = \{A, a, B, b\}$. Every operation is represented as an arrow which results in some new value. The objective is to find the gradients with respect to θ:

$$\nabla_\theta L(\theta) = \{\frac{\partial y}{\partial A}, \frac{\partial y}{\partial a}, \frac{\partial y}{\partial B}, \frac{\partial y}{\partial b}\}$$

Assume that all the variables are scalars and apply the chain rule. The result is:

$$\begin{aligned}
&\frac{\partial y}{\partial b} = \frac{\partial y}{\partial y'}\frac{\partial y'}{\partial b} = \sigma'(y') * 1;\\
&\frac{\partial y}{\partial B} = \frac{\partial y}{\partial y'}\frac{\partial y'}{\partial B} = \sigma'(y') * z;\\
&\frac{\partial y}{\partial a} = \frac{\partial y}{\partial y'}\frac{\partial y'}{\partial a} = \frac{\partial y}{\partial y'}\frac{\partial y'}{\partial z}\frac{\partial z}{\partial z'}\frac{\partial z'}{\partial a} = \sigma'(y') * B\sigma'(z') * 1;\\
&\frac{\partial y}{\partial A} = \frac{\partial y}{\partial y'}\frac{\partial y'}{\partial A} = \frac{\partial y}{\partial y'}\frac{\partial y'}{\partial z}\frac{\partial z}{\partial z'}\frac{\partial z'}{\partial A} = \sigma'(y') * B\sigma'(z') * x.
\end{aligned}$$

These equations allow us to compute the required gradients. However, generally it cannot be assumed that the parameter values are scalars. For the purpose of deep learning, it is convenient imagine that the gradient is as a "transformer of small changes." It's easy to see that in the scalar case, for small changes δx and δy the following is true:

$$\delta y = \frac{dy}{dx}\delta x$$

The same is true for vectors and matrices:

$$\delta Y = \nabla_X Y * \delta X,$$

Where X and Y are some objects and $*$ is an appropriate product operator. We can also introduce Einstein notation: whenever we have a pair of identical indices, we implicitly assume a summation over it (unless specified otherwise). In other words:

$$\sum_{i=0}^{N} x_i y_i$$

can be written simply as:

$$x_i y_i$$

With this in mind, we can compute the gradients:

$\frac{\partial y}{\partial b}$*:* This is the gradient of a vector with respect to another vector. To satisfy the previous property, we want the Jacobian matrix:

$$\left(\frac{\partial y}{\partial b}\right)_{ij} = \frac{\partial y_i}{\partial b_j}$$

which translates into:

$$\delta y_i = \frac{\partial y_i}{\partial b_j} \delta b_j$$

Using our chain rule, we can write:

$$\left(\frac{\partial y}{\partial b}\right)_{ij} = \frac{\partial y_i}{\partial b_j} = \frac{\partial y_i}{\partial y'_k} \frac{\partial y'_k}{\partial b_j}$$

The previous equation shows that since y_i depends (gnerally) on all y'_k, to find the differential we need to sum the contributions from all partial derivatives with respect to the elements of vector $\{y'_k\}$. This is a known rule for finding the partial derivative of a function of multiple parameters. Continuing,

$$\frac{\partial y_i}{\partial y'_k} = \sigma'(y'_i) * \delta_{ik}$$

(Here, δ_{ik} refers to the delta-function, and no summation is assumed over i). In this case, the derivative is non-zero only if $i = k$:

$$\frac{\partial y'_k}{\partial b_j} = \delta_{jk} \Rightarrow \left(\frac{\partial y}{\partial b}\right)_{ij} = \sigma'(y'_i) * \delta_{ik}\delta_{jk} = \sigma'(y'_i)\delta_{jij}$$

(No summation over i)

$\frac{\partial y}{\partial B}$: This is a gradient of a vector with respect to a matrix, to satisfy the property of gradients, we need to get the following multidimensional matrix:

$$\left(\frac{\partial y}{\partial B}\right)^k_{ij} = \frac{\partial y_k}{\partial B_{ij}}$$

In Einstein notation the relationship between incremental changes can be written as:

$$\delta y_k = \frac{\partial y_k}{\partial B_{ij}}\delta B_{ij}$$

Again, using the same procedure from the previous case, we obtain:

$$\left(\frac{\partial y}{\partial B}\right)^k_{ij} = \frac{\partial y_k}{\partial B_{ij}} = \frac{\partial y_k}{\partial y'_p}\frac{\partial y'_p}{\partial B_{ij}}$$

The interpretation is that, y_k depends (generally) on all of y'_p, thus, to find the differential, we need to sum all partial differentials over the vector y'_p (rule for finding derivative of a function of multiple parameters). Again, for the first multiplier:

$$\frac{\partial y_k}{\partial y'_p} = \sigma'(y'_k)\delta_{kp} = diag(\sigma'(y'))_{kp}$$

Also, the second partial derivative in the product is non-zero only if $p = i$

$$\frac{\partial y'_p}{\partial B_{ij}} = z_j\delta_{pi}$$

And the final result is:

$$\left(\frac{\partial y}{\partial B}\right)^k_{ij} = diag(\sigma'(y'))_{kp}z_j\delta_{pi}$$

Note that all the derivations above are purely theoretical. Using multidimensional matrix notation is useful for understanding how the gradients are structured. However, in practice there is no

need to multiply often very sparse matrices.

F. Convolutional Neural Networks

Convolutional neural networks (CNNs) are a specific type of neural network that are often used for image processing tasks. They are built upon the idea of convolution, which is a mathematical operation that involves sliding a small filter over the image and performing a dot product at each location. The result of this operation is a feature map that highlights certain characteristics of the image. The architecture of a typical CNN is shown in Figure 9.

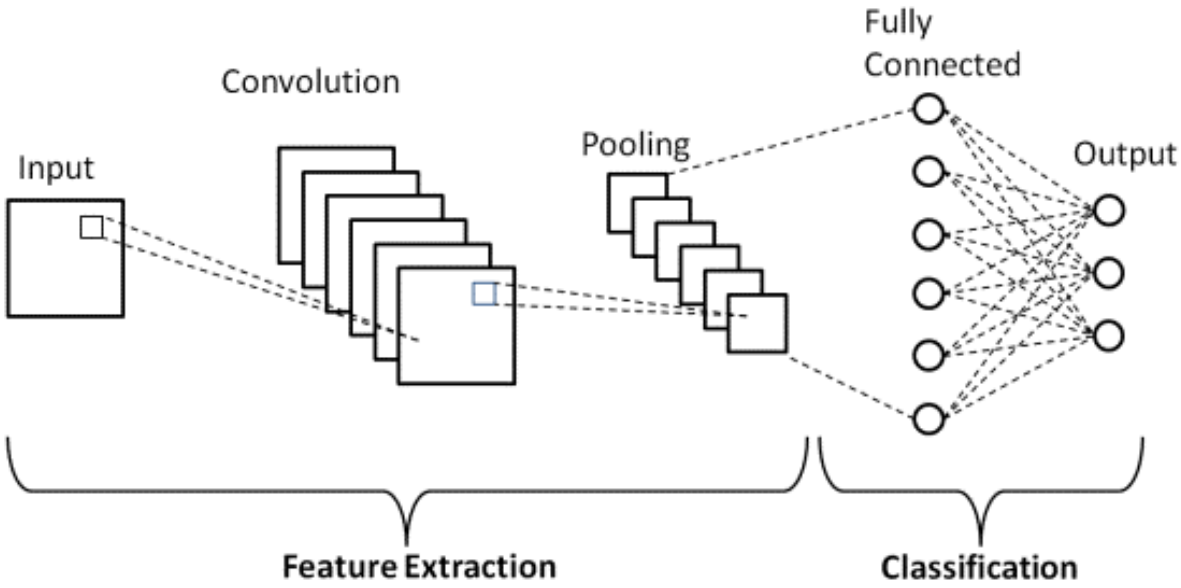

Fig. 9: Convolutional Neural Network architecture.

The basic building blocks of a CNN are convolutional layers, pooling layers, and fully connected layers. Convolutional layers perform the convolution operation on the input image using a set of learnable filters. The output of a convolutional layer is a set of feature maps, where each map corresponds to one filter. Pooling layers reduce the dimensionality of the feature maps by summarizing groups of neighboring values. This is typically done by taking the maximum or average value of a local region. Finally, fully connected layers perform a linear transformation on the flattened output of the previous layer, followed by a non-linear activation function.

One of the key advantages of CNNs is their ability to learn hierarchical representations of an image. Lower-level filters in the network learn to detect simple features such as edges and

corners, while higher-level filters learn to detect more complex features such as shapes and patterns. This allows the network to effectively capture the structure of the image and classify it correctly.

In our project, we use a variant of CNNs called YOLO (You Only Look Once) that was specifically designed for object detection in real-time applications. YOLO uses a single convolutional network to predict bounding boxes and class probabilities directly from the input image, without the need for region proposals or multiple stages of processing. This makes it much faster than traditional object detection methods, while still maintaining high accuracy.

5. YOLOv3-MULTI

A. Introduction

In Spring of 2022, the lane detection team built a proof-of-concept model for performing lane segmentation through the use of the U-Net framework [2]. Following this success, the Fall 2022 team branched out towards other computer vision tasks needed for autonomous vehicles. The members explored object detection with the implementation of YOLOv1 trained on the BDD100k dataset. However, the two models implemented had a multitude of problems - the main issue being the inefficiency caused by performing separate forward passes for each task. So, the Fall 2022 goal evolved into building a model that unified the two detection and segmentation tasks into one network to allow for a single forward pass, and thus increase efficiency. The team improved the model's accuracy through the introduction of a deeper backbone in Darknet-53 and an increase of the input resolution from 416×416 to 384×640. 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. Additionally, the resolution of the images in the BDD100k dataset is 720×1280, so it was desirable to maintain a similar aspect ratio rather than compressing the image into a square and losing some valuable information.

Figure 10 shows a high-level diagram outlining the full structure of the Fall 2022 YOLOv3-Multi model. In the diagram, layers are denoted as 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 indicates where in the model the size of the image is changed and allows for an intuitive understanding of the dimensions at each step.

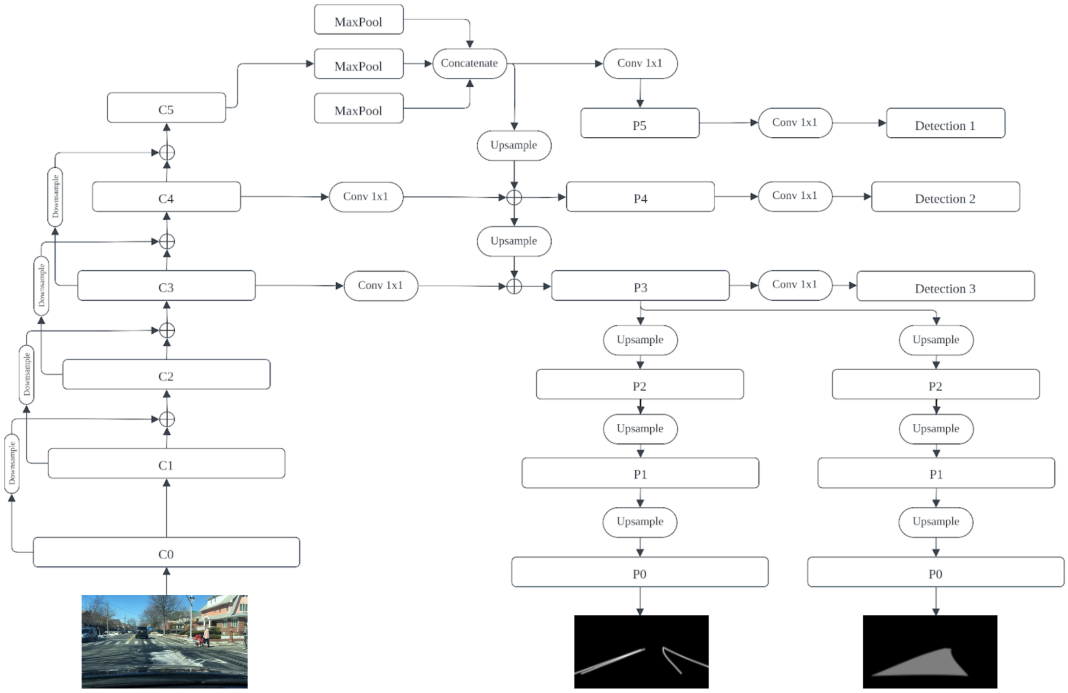

Fig. 10: YOLOv3-Multi Model Structure for Multi-task Learning.

B. YOLOv3-Multi Architecture

1) ResNet:

The core hypothesis behind the ResNet paper was that a deep neural network should perform better, otherwise the same, when compared to a shallower neural network [3]. We define a deep neural network to have more convolutional layers, and the converse 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 [3]. 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 [3]. 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.

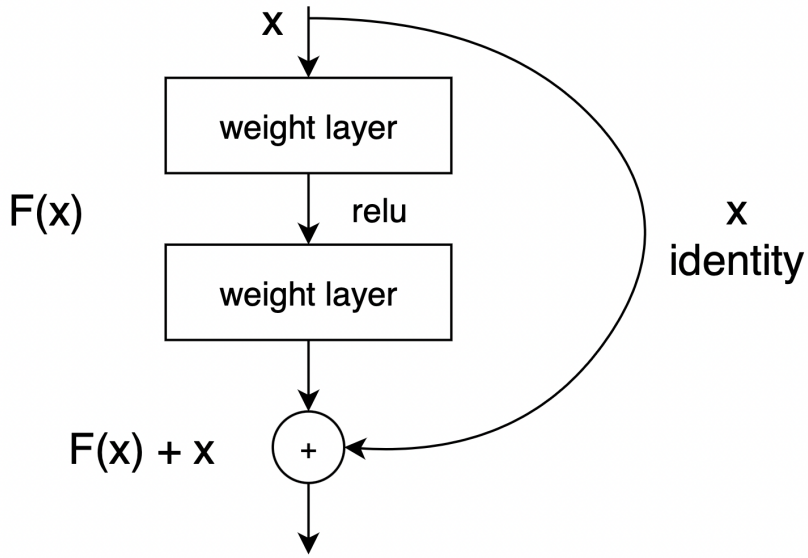

Fig. 11: Residual Block. [3]

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 [3]. 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 the YOLOv3-Multi 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 [4]. Table III shows slight modifications to what was introduced in the YOLOv3 paper. The input image size

was set 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.

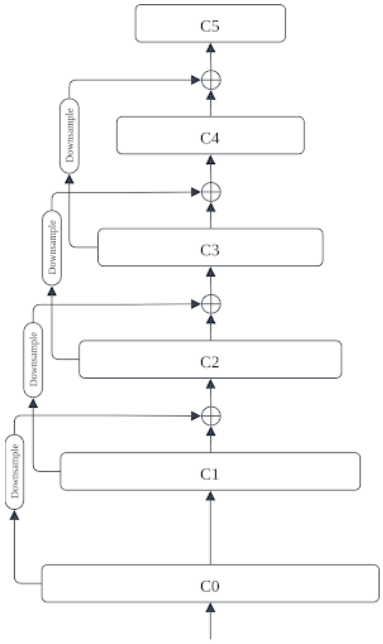

Fig. 12: Darknet-53 Architecture.

	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. [4]

3) 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 allowed the YOLOv3-Multi model to extract the multi-scale local region features which are not captured from the Feature Pyramid Network [5]. 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.

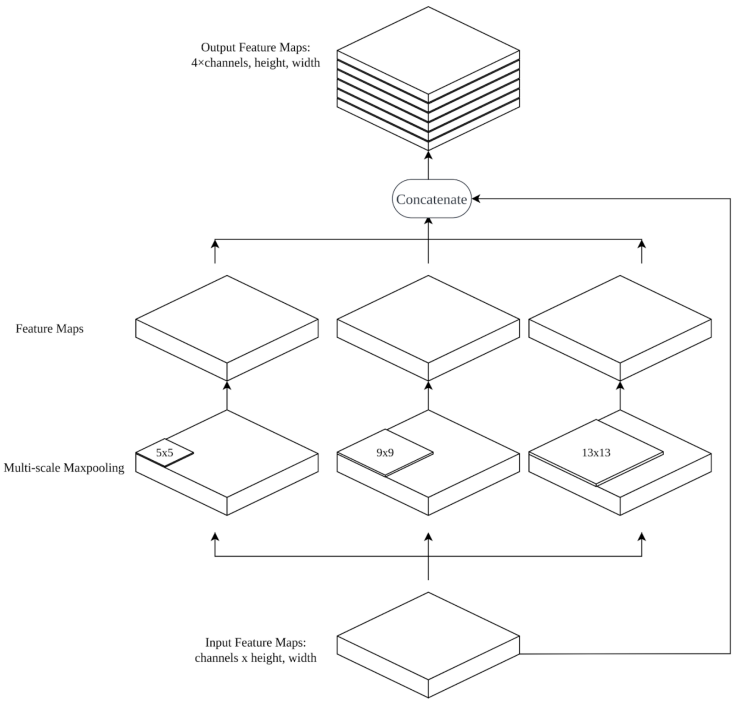

Fig. 13: Spatial Pyramid Pooling. [5]

4) 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, the YOLOv3-Multi model implements a Feature Pyramid Network (FPN), as shown in Figure 14, 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 [6]. 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.

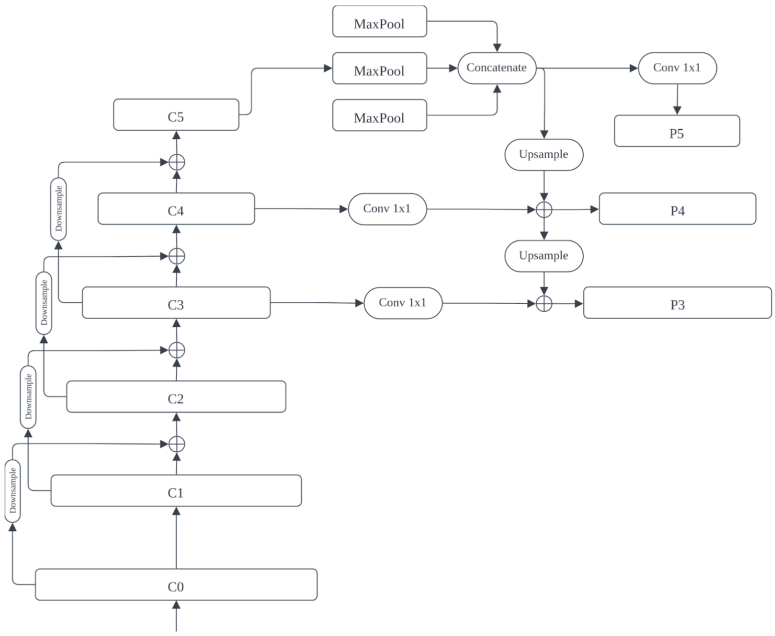

Fig. 14: FPN Architecture.

C. Loss Functions

1) Detection:

To quantify the model's object detection accuracy each after iteration, the YOLOv3-Multi architectures used two loss functions: Focal Loss and Complete-IOU loss [4].

Focal loss [7] in Equation 1 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 [7]. 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.

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

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 [8] defined by Equation 2 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 [8].

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

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 the segmentation tasks, Focal Loss [7] was once again used. 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.

D. Evaluation Metrics

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 6.

$$Precision = \frac{TP}{TP + FP} \tag{3}$$

$$Recall = \frac{TP}{TP + FN} \; Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{4}$$

$$AP = \int_0^1 Precision(Recall) d(Recall) \tag{5}$$

$$mAP = \frac{1}{classes} \Sigma_{i=1}^{classes} AP_i \tag{6}$$

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.

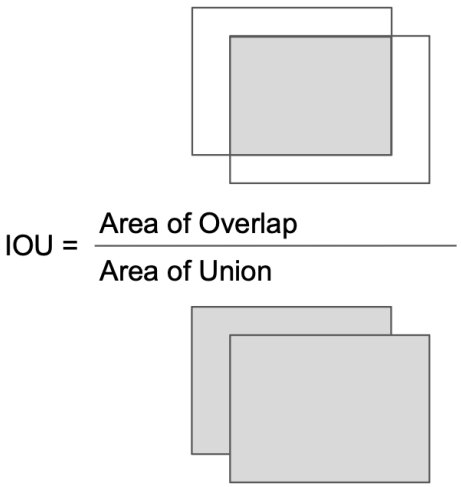

Fig. 15: Visual Representation of IOU.

$$IOU = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|} \tag{7}$$

Suppose A is the predicted bounding box and B is the ground truth box. Using Equation 7, IOU is computed to measure the overall accuracy of the bounding boxes to the ground truth bounding boxes.

3) Mean IOU:

For evaluating the segmentation results, intersection over union is used to quantify the accuracy of the segmentation compared to the groundtruth. A mean IOU for each class is computed by calculating a vector of ten IOU values for each image, representing the ten non-background segmentation classes present in the dataset. If any of the class types are not present 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.

E. Results

The YOLOv3-Multi model developed in Fall 2022 was trained for 50 epochs with evaluation metric results shown in Table IV. 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. For individual classes, 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 common than any of the other classes. Most of these accuracy results are promising but not satisfactory. The results of the model can be further contextualized through a visualization of the model outputs shown in Figure 16 - 19.

	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: YOLOv3-Multi Evaluation Metrics.

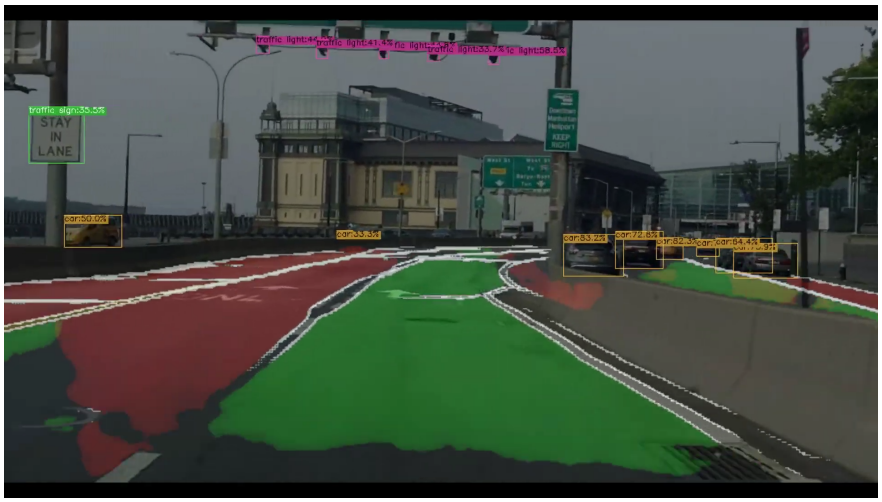

Fig. 16: YOLOv3-Multi Results (1).

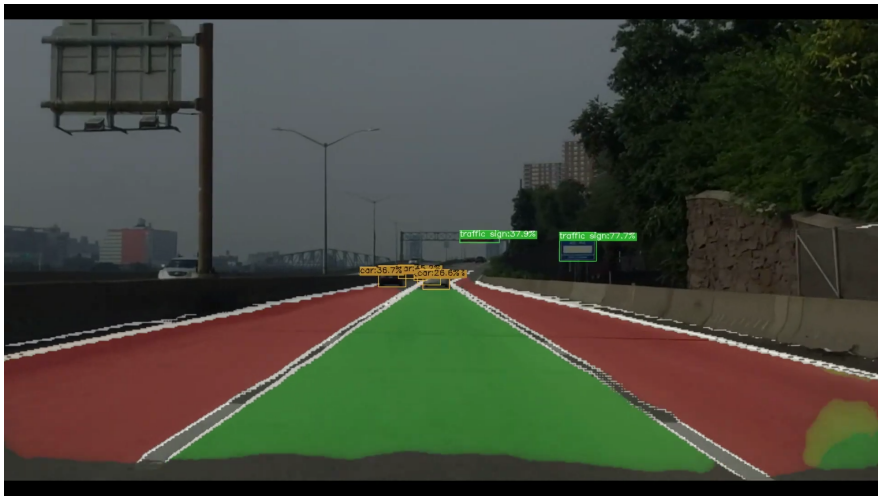

Fig. 17: YOLOv3-Multi Results (2).

Fig. 18: YOLOv3-Multi Results (3).

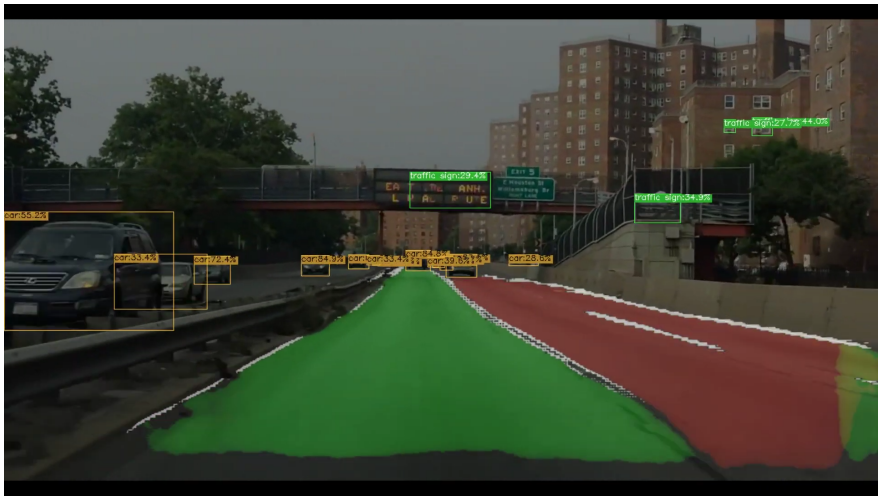

Fig. 19: YOLOv3-Multi Results (4).

6. YOLOv4-Multi

A. Introduction

The main goal of this semester was to adapt last semester's YOLOv3-Multi model to incorporate various optimizations introduced in the official YOLOv4 paper [9] with the purpose of improving on the accuracy results obtained by the Fall 2022 implementation. Our team aimed to achieve this by improving the computation capabilities of the model through a more efficient feature extractor and activation function, and by increasing the accuracy of our model by incorporating an augmented feature aggregator to allow for more effective fusion of high-level semantic features and low-level spatial features. So, this semester we researched a range of concepts, including the Cross-Stage Partial Network, the Path Aggregation Network, the Mish activation function, CutMix data augmentation, and more. Many of these concepts were developed and added to our model to create the complete YOLOv4-Multi architecture, as shown in Figure 20. Additionally, we discovered and fixed various bugs and inconsistencies present in the source code from last semester, and added other minor additions to the project such as the option to train on multiple GPUs, a complete visual postprocessing algorithm, and improved methodology for obtaining evaluation metrics.

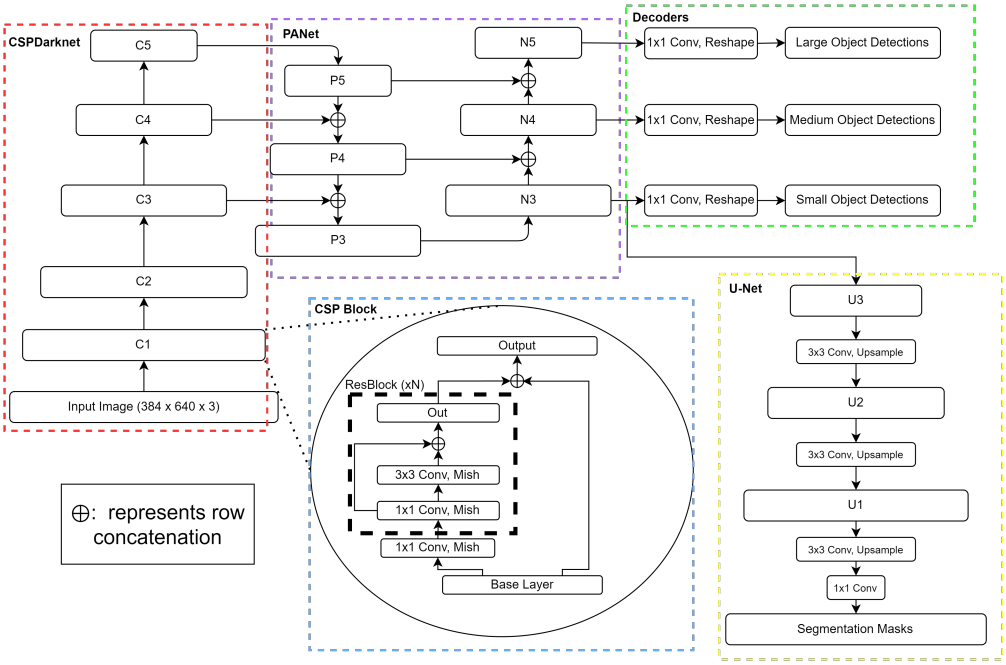

Fig. 20: Complete Model Diagram.

B. YOLOv4-Multi Architecture

1) Mish Activation:

For the activation function of our YOLOv4-Multi model, we utilized the Mish function. The Mish activation function is an improved version of the Rectified Linear Unit (ReLU) function, previously defined in Section 4.2. We again define $ReLU(x)$ in Equation 8, which was used in last semester's Fall 2022 implementation. The problem with ReLU is that it is not continuously differentiable due to its behavior when $x = 0$.

$$ReLU(x) = \begin{cases} x: & x > 0 \\ 0: & x \leq 0 \end{cases} \tag{8}$$

There is a corner at the origin of ReLU, which means that the derivative must be approximated when performing backpropagation. The Mish activation function, as described in the official paper by Misra [10], was derived by a Neural Architecture Search (NAS) which searches the non-linear function space to determine optimal activation functions. Mish builds upon the Swish function, which is a self-gated version of the classic sigmoid function $\sigma(x) = \frac{1}{1+e^{-x}}$. Swish $S(x)$ in Equation 9 achieves self-gating by multiplying the sigmoid by the input x, allowing the function to be unbounded above. This feature, which is shared by ReLU, prevents gradient saturation and improves gradient flow.

$$S(x) = \frac{x}{1 + e^{-x}} \tag{9}$$

Swish, unlike ReLU however, is continuously differentiable and non-monotonic (not strictly increasing/decreasing). These two features, shown in Figure 21, are beneficial as continuous differentiability eases derivative computation and non-monotonicity allows for the preservation of small negative weights, thus preventing model overfitting.

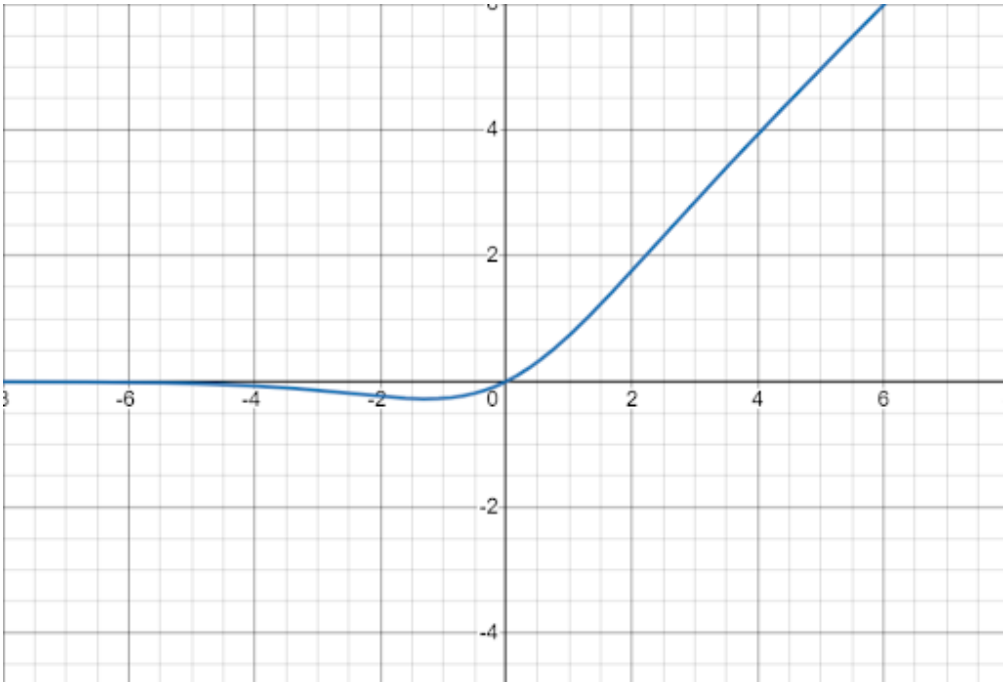

Fig. 21: The Swish function.

While Swish has proven to be a more effective activation function than ReLU, the NAS conducted by Misra determined that the Mish activation function was more optimal. Mish, similar to Swish, is based on the integral of the sigmoid function, also called the SoftPlus function: $\int \sigma(x) = ln(1 + e^x)$. Mish takes the hyperbolic tanget of the SoftPlus function, and multiplies by the original input x, as shown by $M(x)$ in Equation 10.

$$M(x) = x\text{Tanh}(ln(1 + e^x)) \tag{10}$$

These additions to SoftPlus allow Mish to have the same beneficial traits as Swish: continuous differentiability, non-monotonicity, and unboundedness above (See Figure 22); while achieving better accuracy than Swish.

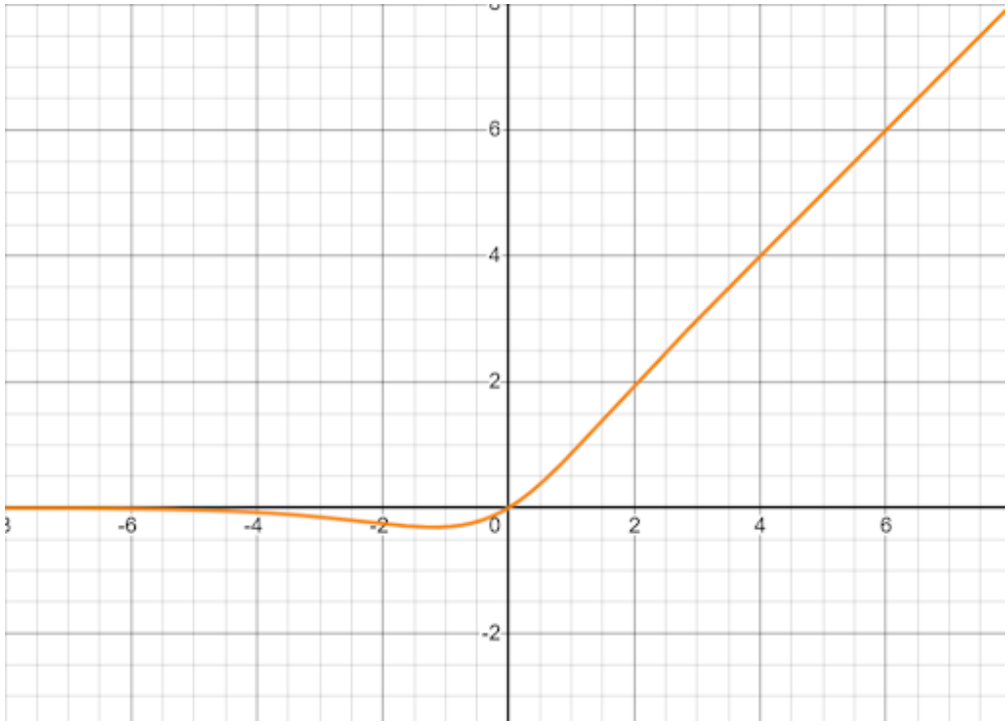

Fig. 22: The Mish function.

2) Cross-Stage Partial Network:

The development of the Cross-Stage Partial Network (CSPNet) as described in the official paper by Wang et al. [11], was motivated by the desire to reduce the computation cost of convolutional neural networks, thus allow for cheaper and less complex processors to run object detection tasks at reasonable speeds. The main intuition behind CSPNet that there are many duplicate gradient calculations performed in the backpropagation of standard CNNs. The authors of the CSPNet paper theorized that many of these duplicate calculations could be removed by integrating early feature maps with later ones, in a manner similar to the concept of residual connections.

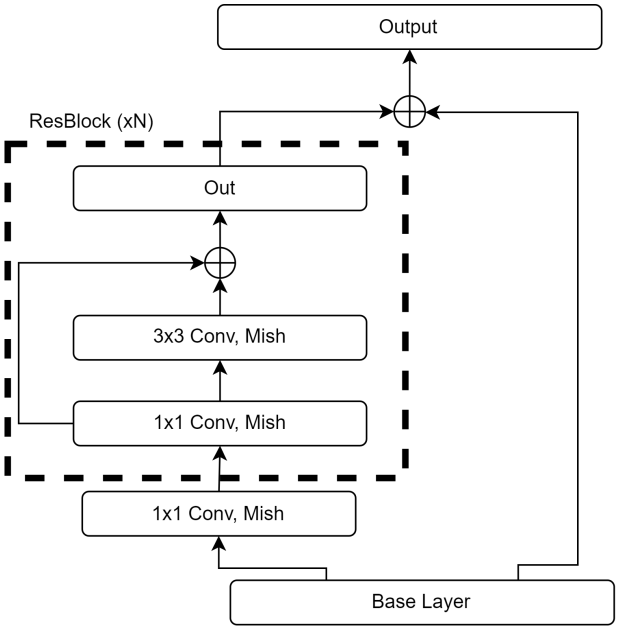

Fig. 23: A Cross-Stage Partial Layer.

The authors designed cross-stage partial connections within the convolutional layers, which partitioned the feature maps of the input into two paths, where one path carried out standard convolutions and activations, whereas the other underwent no operations. These paths, shown in Figure 23, would later be merged across stages before being passed to the next layer. It was found that generally, the introduction of these extra gradient flow connections allowed gradient computation to skip over duplicate calculations, and resulted in a general reduction of network computation costs by around 20% without any loss of model accuracy.

3) Path Aggregation Network:

The feature aggregator we implemented this semester was the Path Aggregation Network (PANet) as described in the official paper by Liu et al. [12]. PANet is an augmentation of the Feature Pyramid Network [6] used in the Fall 2022 YOLOv3-Multi implementation. PANet adds a second "Bottom-Up" path to the end of the standard FPN architecture, and adds additional lateral connections to connect both bottom-up paths with the central top-down path, as shown in Figure 24. These additional lateral connections and extra bottom-up path allow for low-level features, which often contain important spatial information, to take a shortcut and efficiently

aggregate with the rich semantic information extracted from the high-level features.

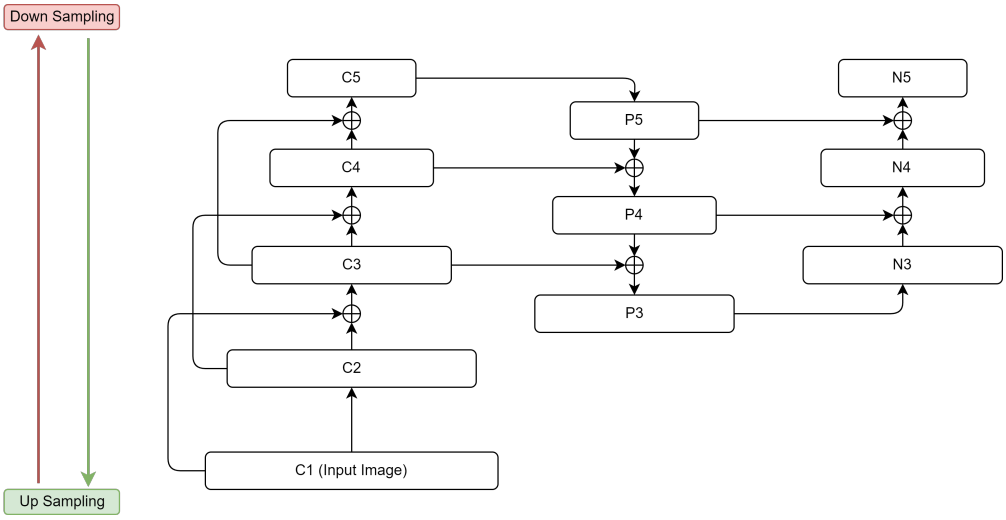

Fig. 24: CSPDarknet with PANet.

4) Detection Decoder:

For transforming the model's output into detections, we utilized a similar version of decoder used last semester. Our YOLOv4-Multi model continues to analyze features at three different scales, obtained from the feature maps in N3, N4, and N5 (See Figure 20 or 24), in order to detect objects of varying sizes.

To arrive at the desired output matrices necessary for visualizing and quantifying the detections, we carry out a series of convolutions and reshapes as shown in Figure 25. The N5 layer will have 1024 12×20 feature maps, the N4 layer will have 512 24×40 feature maps, and the N3 layer will have 256 48×80 feature maps. Each of these layers are sent through a 1×1 convolution to pool all their channels into 54 channels. These three outputs are then reshaped to reintroduce the three RGB color channels into the detections, achieving three RGB grids of varying sizes, where each 18-value grid-cell represents a detection at a given grid-cell size. The 18 values signify the five bounding elements for the object (center x-coordinate, center y-coordinate, box width, box height, and objectness confidence), as well as the 13 object class probabilities.

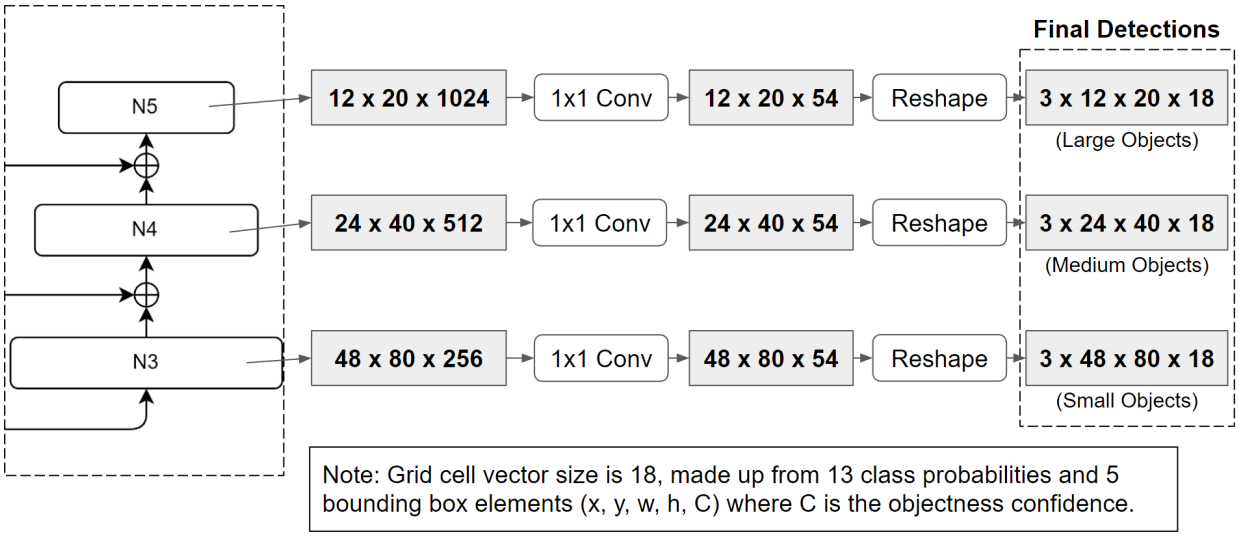

Fig. 25: Detection Decoder.

5) Segmentation Decoder:

As for obtaining segmentations from the model's output, we again utilized the second half of the U-Net architecture developed by the Spring 2022 team. Our YOLOv4-Multi model takes the N3 feature maps (See Figure 20 or 24) and sends a copy to the segmentation decoder to achieve the desired segmentation masks.

The N3 layer's feature maps are sent through the U-Net component as shown in Figure 26. The U-net component carries out a series of 3×3 convolutions and 2D upsamples to eventually return to the original input image resolution of 384×640. However, U-Net ends with 32 channels, so these channels are pooled down to 12 to represent the segmentation masks for the 12 classes in our dataset (8 lanes and background, 2 drivable areas and background).

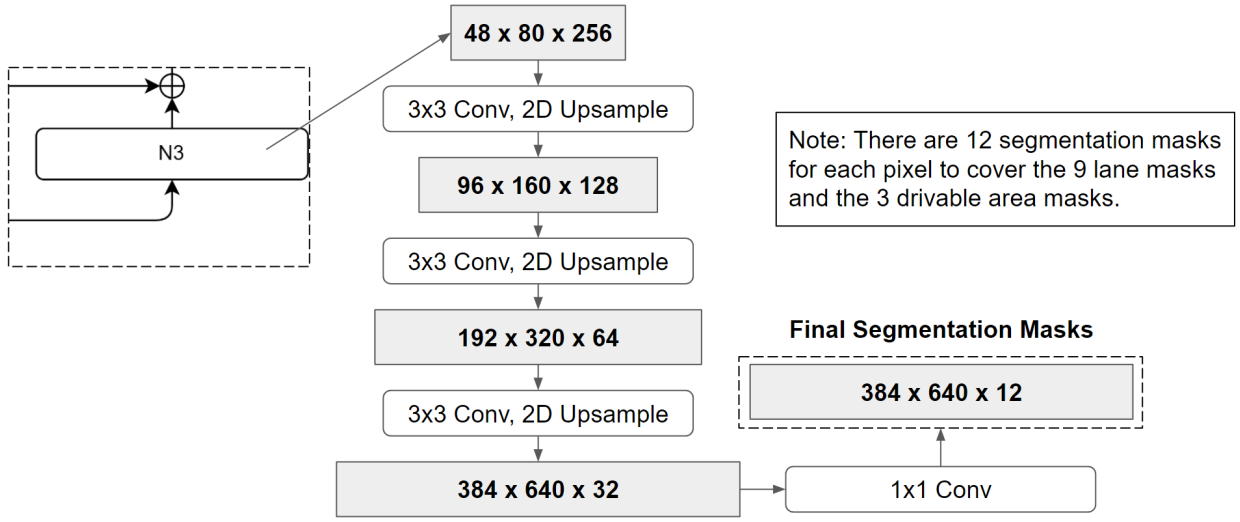

Fig. 26: Segmentation Decoder.

C. Loss Functions

For quantifying loss, we utilized many of the same functions as the Fall 2022 YOLOv3-Multi implementation. For object detection, Focal loss and Complete-IOU loss were once again used (See Section 5.3.1). For segmentation loss, we still used Focal loss for classification accuracy, but added a IOU loss as a segmentation accuracy.

IOU loss measures the overlap accuracy between a groundtruth mask and a predicted segmentation mask for a given class. The IOU loss across all classes is shown in Equation 11. It takes the bitwise-and of the two binary masks, and divides by the bitwise-or of the masks. If the divisor is zero, we simply add the bitwise-and of the masks, which will represent a false positive segmentation of a class that does not exist in the image.

$$\mathcal{L}_{IOU} = 1 - \frac{1}{C}\sum_{c=1}^{C} IOU_c \tag{11}$$

$$IOU_c = \frac{|P_c \cap G_c|}{|P_c \cup G_c|} \tag{12}$$

Where C is the number of segmentation classes, IOU_c is the computed IOU for class c (defined in Equation 7), P_c is the predicted segmentation mask for class c, and G_c is the groundtruth segmentation mask for class c.

D. Evaluation Metrics

For evaluating the accuracy of our model's detections and segmentations, we used the same metrics as the Fall 2022 YOLOv3-Multi implementation (See Section 5.4). Intersection over union, and the connected Mean IOU metric, were used to measure the segmentation accuracy of our model. However, we were not able to get complete object detection results this semester, so mean average-precision was not needed for evaluation.

E. Segmentation Results

The YOLOv4-Multi model developed this semester was trained for 300 epochs with evaluation metric results shown in Table V. The metric we used for the measuring the segmentation task was mean IOU. The drivable area segmentation results saw the greatest accuracy increase, jumping from just about 54% to a much higher accuracy of about 90%. However, a persistent issue with our model is the low lane segmentation IOUs. Just like the Fall 2022 model, the YOLOv4-Multi model predicts thick lane lines compared to the groundtruth lanes, which are only a couple pixels wide. Again, for individual classes the segmentation accuracy is higher for the more frequently appearing classes, such as single white lanes and road curbs. While most of these lane line accuracy results yet not satisfactory, we are pleased that the accuracy increased for nearly all classes, save for the crosswalk class. The results of the YOLOv4-Multi model can be further contextualized through a visualization of the model outputs shown in Figure 27 - 30.

Fall 2022 (YOLOv3)		Spring 2023 (YOLOv4)	
Segmentation Class	mIOU(%)	Segmentation Class	mIOU(%)
Drivable Area	54.42	Drivable Area	89.92
Lane Predictions (all)	10.4	Lane Predictions (all)	16.87
Single White Line Lanes	25.3	Single White Line Lanes	40.54
Road Curbs	18.8	Road Curbs	32.8
Cross Walks	7.6	Cross Walks	3.59
Single Yellow Lines	6.4	Single Yellow Lines	14.8
Double Yellow Lines	2.1	Double Yellow Lines	5.81
Double White Lines	2.0	Double White Lines	3.7

TABLE V: Comparison of Segmentation Evaluation Results.

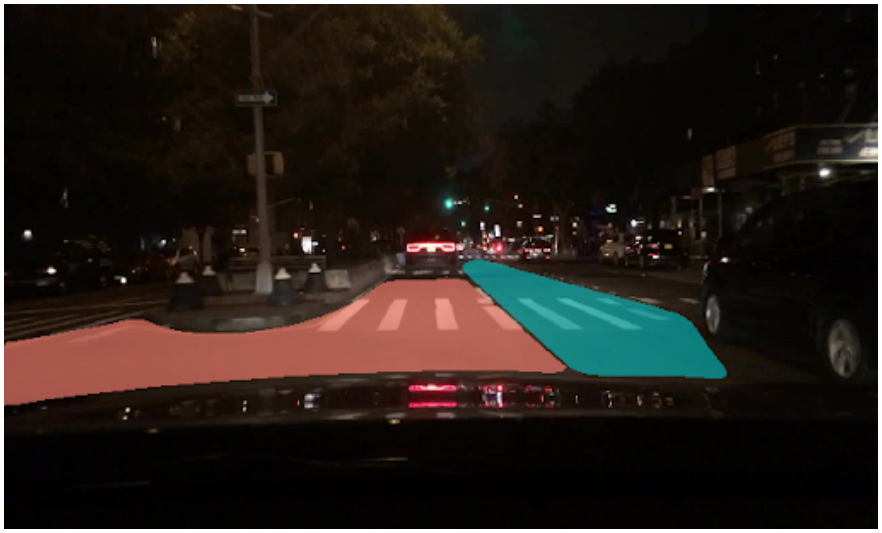

Fig. 27: YOLOv4-Multi Drivable Area Results (1).

Fig. 28: YOLOv4-Multi Lane Line Results (1).

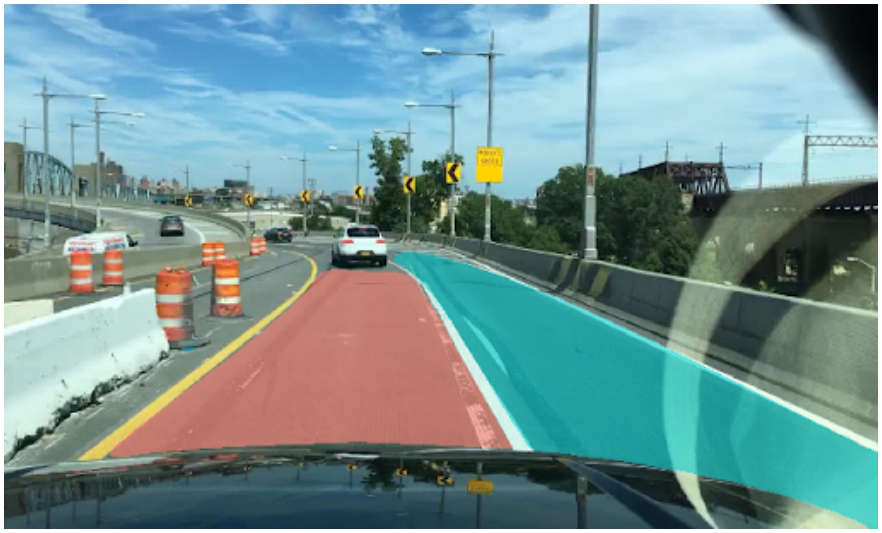

Fig. 29: YOLOv4-Multi Drivable Area Results (2).

Fig. 30: YOLOv4-Multi Lane Line Results (2).

7. Future Work

A. Data Augmentation

Data augmentation can improve the training of deep learning models like ours by introducing image variation in the training dataset. Data augmentation concepts have been proven to achieve better generalization and robustness. In this semester, we researched implemented CutMix data augmentation [13], a combination of image mixing and partial image pasting, as a data augmentation technique. CutMix combines two images and their labels by cutting and pasting a random patch from one image to another. This process results in a new image that contains a mix of the original images and their corresponding labels. This strategy introduces more variation to the groundtruth data, which can force the model to learn less prominent object features, rather than relying heavily on the more common identifying features.

To optimize the performance of CutMix, we have also implemented a Grid Search algorithm in order to determine the best hyperparameters for this technique, such as the mixing ratio and the size of the patch. However, we encountered several bugs during the implementation phase, which limited the effectiveness of CutMix and Grid Search in our current work.

In future works, the team can focus on resolving these bugs and refining our CutMix and Grid Search implementations. This optimization has the potential to further improve the model's ability to recognize objects in diverse and difficult scenarios, thus enhancing the model's performance.

B. Object Detection Results

The increased complexity of this year's model led to many challenges, including the difficulty of training with standard training loops. We encountered this problem when training our model for the object detection task. Since our detection model deals with many classes in very diverse environments, the model is prone to overfitting. Given our limited access to computational resources, future work can involve attempting to incorporate tools that improve the training performance and generalizability of object detection training. Such tools include but are not limited to various weight initialization techniques, weight decay, learning rate schedulers, etc. More work needs to be done to improve numerical stability and ensure that the weights have consistent distributions among the layers.

8. Conclusion

The results obtained this semester are promising, as our optimized YOLOv4-Multi model is able to perform lane detection and drivable area segmentation at improved computation cost and accuracy when compared to the Fall 2022 YOLOv3-Multi implementation. The team also made significant progress towards obtaining improved object detection results, but computation and time constraints caused this task to be pushed towards future work. However, our segmentation results are quite promising, especially for drivable area. Our overall lane lane segmentation had a mean IOU of 40.54%, and drivable area segmentation had a mean IOU of 89.92%. The lower lane line accuracy is still understandable, considering that the groundtruth lane lines are very small. Regardless, these are certainly not optimal results, and future work should focus on eliminating the thick segmentations of our model. In summary, our final lane detection model was able to run the multiple tasks we set out to achieve in a single forward pass, and improved on computation costs and model accuracy when compared to the Fall 2022 implementation.

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] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," May 2015. [Online]. Available: https://arxiv.org/abs/1505.04597

[3] 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

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

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

[6] 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

[7] 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

[8] 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

[9] 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

[10] D. Misra, "Mish: A self regularized non-monotonic activation function," Aug 2020. [Online]. Available: https://arxiv.org/abs/1908.08681

[11] C.-Y. Wang, H.-Y. M. Liao, I.-H. Yeh, Y.-H. Wu, P.-Y. Chen, and J.-W. Hsieh, "Cspnet: A new backbone that can enhance learning capability of cnn," Nov 2019. [Online]. Available: https://arxiv.org/abs/1911.11929

[12] S. Liu, L. Qi, H. Qin, J. Shi, and J. Jia, "Path aggregation network for instance segmentation," Sep 2018. [Online]. Available: https://arxiv.org/abs/1803.01534

[13] S. Yun, D. Han, S. J. Oh, S. Chun, J. Choe, and Y. Yoo, "Cutmix: Regularization strategy to train strong classifiers with localizable features," Aug 2019. [Online]. Available: https://arxiv.org/abs/1905.04899

9. Appendix

A. *William Stevens*

Spring 2023 was my third semester in VIP-IPA, and my second semester on the Lane Detection team. I was the only team member who returned for this semester, so it fell upon me to direct the team and get the new team members up to speed in order for them to help me continue the project from last semester. I compiled much of the work from Fall 2022 into a condensed onboarding program for the new members to engage in during their first month on the team. After finishing the tasks in the program, my team members were knowledgeable of the various machine learning concepts and neural network architectures researched in Fall 2022.

For my own work, I spent the first two and a half months researching and implementing many of the network optimizations introduced in YOLOv4. The optimizations I researched include the Cross-Stage Partial Network feature extractor, the Path Aggregation Network feature aggregator, the Mish Activation function, and CutMix data augmentation. After learning about and presenting on each of these topics, I implemented and tested them in Python and adjusted last semester's model to account for the new changes. I spent the last month of the semester working on the training script and various postprocessing algorithms in order to produce final results, both visual and numerical.

On a more weekly basis, I directed the work of the team each week in our short meetings after Tuesdays and our longer virtual meetings on Sundays. I made plans for and contributed slides to the weekly presentations. In addition, I created our poster submission for participation and presentation in the Research Conference Symposium. Within this report, I was responsible for writing the abstract, introduction, YOLOv4-Multi, and conclusion sections. I also adapted the dataset and YOLOv3-Multi sections from last semester's report to fit into the context of this semester.

B. Mykhailo (Misha) Tsysin

This semester was my first semester in the VIP team and in the Lane Detection team. Coming with a limited background in image processing, I was able to learn significant amount of material. I started by getting familiar with the image processing foundations, such as affine and projective transformations, color modifications, and various classic filters. I continued by catching up with the team knowledge in deep neural networks. Beginning with the implementation if a basic classifier model, I gradually proceeded to more complex topics such as YOLOv1, YOLOv3, and the new additions to our model – CSPNet, PANNet, etc. During the second half of the semester I was mainly responsible for developing the detection part of our model. I started with the last year's code as a baseline and was able to significantly improve it and fix most of the major bugs. Among them were such things as inappropriate use of activation functions, incorrect and numerically unstable losses, floating point arithmetic overflows and more. I proceeded to training the detection model, which was pretty challenging considering the greatly increased complexity of this semester's model. I researched the prior work in the field and incorporated various improvements to the training loop, such as learning rate schedulers, custom initialization functions, weight decay and others. These are the main points that I plan to improve for the next semester – given the overall complexity of our model, it's more reasonable to tune it to its fullest potential. For this report, I worked on the theoretical foundations of neural network design.

C. Xingyan (Ian) Li

This semester, I joined the Lane Detection team for the first time, with no prior experience in deep learning and image processing. Under the guidance of my teammates and with the help of various resources, I quickly learned and adapted to the challenges presented. Initially, I struggled to grasp basic concepts and keep up with my teammates' progress, but my dedication and strong work ethic enabled me to overcome these obstacles and contribute to the project.

I began by focusing on understanding the fundamentals of convolutional neural networks (CNNs) and their applications in image processing. I delved into various research papers and online courses to establish a solid foundation in this area. To improve my understanding of deep learning and image processing, I spent a significant amount of time learning through various tutorials and resources. As a result, I acquired the skills necessary to implement YOLOv1 and YOLOv3 models independently. This newfound knowledge allowed me to contribute effectively to the group project and collaborate with my teammates. My involvement positively impacted several aspects of the project, such as implementing image resizing, CutMix, and Grid Search techniques to enhance the model's performance. Additionally, I prepared weekly slides and presentations, contributed to the final report, and participated in the poster presentation.

Overall, my experience with the VIP Lane Detection team has been highly rewarding. I have acquired valuable knowledge and skills in deep learning, image processing, and engineering design, which will undoubtedly serve me well in my future endeavors. I eagerly anticipate building upon this experience and further exploring the captivating world of computer vision and machine learning.