

SCHOOL *of* BUSINESS AND TECHNOLOGY

Department of Engineering and Aviation Sciences

**Design of Autonomous Food Delivery Vehicle**

**Montraz Oliver**

Advisor: Dr. Zhang

June. 4, 2020

Design of Autonomous Food Delivery Vehicle

By

Montraz Oliver

Submitted to the Department of Engineering and Aviation Sciences in partial fulfillment of the requirements for the degree of Bachelor of Science in Engineering at the

UNIVERSITY OF MARYLAND EASTERN SHORE

Date

The author hereby grants the University of Maryland Eastern Shore permission to reproduce and distribute publicly, paper and or electronic copies of this document in whole or in parts.

Authors Montraz Oliver

Signature

Date

Department of Engineering and Aviation Sciences

Certified by

List of Contents

[List of Contents 3](#_Toc42169834)

[List of Figures 2](#_Toc42169835)

[List of Tables 3](#_Toc42169836)

[Chapter 1. Introduction 4](#_Toc42169837)

[1.1 Background/Motivation 4](#_Toc42169838)

[1.2 Objective 5](#_Toc42169839)

[Chapter 2. Implementation Plan 6](#_Toc42169840)

[2.1 Tasks 6](#_Toc42169841)

[Chapter 3. Implementation 7](#_Toc42169842)

[3.1 Implementation of Task 4. 7](#_Toc42169843)

[3.1.1. Implementation of Subtask 4.1: Creating dataset 7](#_Toc42169844)

[3.1.2. Implementation of Subtask 4.2: Modeling 10](#_Toc42169845)

[3.1.3. Implementation of Subtask 4.3: Training 12](#_Toc42169846)

[3.1.4. Implementation of Subtask 4.4: Testing 13](#_Toc42169847)

[Appendix 15](#_Toc42169848)

[A. Source Code. 15](#_Toc42169849)

[1. Source Code of Pascal VOC Format Python Coversion 15](#_Toc42169850)

[2. Source Code of Preparing Dataset 15](#_Toc42169851)

[3. Source Code Data Block API and added Test Set 16](#_Toc42169852)

[4. Source Code of Creating Dynamic Unet 17](#_Toc42169853)

[5. Source Code of fitting learning rate and training 17](#_Toc42169855)

[6. Source Code of Predictions on Test Set 17](#_Toc42169856)

[REFERENCES 18](#_Toc42169857)

List of Figures

[Figure. 1. Starship Robot designed by Starship Technologies 6](#_Toc42163864)

[Figure. 2. Original photos captured via iPhone 8](#_Toc42163865)

[Figure. 3. Annotation of image using LabelMe 9](#_Toc42163866)

[Figure. 4. Folder of 200 JPEG images 10](#_Toc42163867)

[Figure. 5. Folder of 200 label images 10](#_Toc42163868)

[Figure. 6. Input data created by data block API 12](#_Toc42163869)

[Figure. 7. Training metrics of Nerual Network 13](#_Toc42163870)

[Figure. 8. Prediction on test set image 15](#_Toc42163871)

List of Tables

**No table of figures entries found.**

1. Introduction

Autonomous self-driving vehicles are growing exponentially in popularity within new-age technology. There are only a few level 5 self-driving vehicles that require no human activation to drive. These self-driving cars are being used to transport goods and people around the world. Furthermore, the electric car is preferred more often because it offers economic, global, and environmental benefits. The car will be planed and designed to deliver food to people autonomously.

## Background/Motivation

On the campus of the University of Maryland Eastern Shore, there are many students that worry about their next meal and how they are going to get hold of it in a timely fashion. A college student has numerous responsibilities on their table, which limits their ability to obtain food at an ideal time. Throughout an average college student's day, they might have at least 3 to 6 classes per day, extracurricular activities if they are an athlete or in some type of group, and they must study to stay on top of their grades. Furthermore, most colleges have freshmen who are not allowed to have cars. And if any student who drives a car must pay a lot of money for parking and fees. This limits the number of college students with cars and increases the number of college students walking around campus. “Among the 214 National Universities that reported these data to U.S. News in an annual survey, the average percentage of students who brought cars to campus in the 2016-2017 academic year was 46.8 percent.”

When college students are in their dorm after a long day of classes and stressful lecturing. They still must do homework and study for their classes. At that point, they are tired of walking and going to get some food that is at least .4 miles away. The combination of the gruesome schedule of college students makes and the tireless walking around campus makes it hard for college students to eat food when it is offered.

What if it was possible to send Campus Universities dining services to students around their dorm room or any location they might possibly be located. This would allow campus living students to have the option to not worry about receiving food without stopping their homework and walking half a mile to their cafeteria or dining services. Autonomous Food Delivery System can be created to deliver food around the campus to any recipient, preferably college students living on campus.

At the University of Maryland Eastern Shore students are offered food at the locations including Students Service Center, Engineering & Aviation Sciences Complex, Hawk’s Nest, Waters Hall. In the Students Service Center, they offer students plateau dining and oasis staff dining. The UMES campus offers food, but they have no form of delivery options for students to receive food on campus. Furthermore, college students often argue and complain about the food they receive from the cafeteria. This leads them to eat the food that the university provides or any other dining services other than the cafeteria options. There should also be an option for students to choose the food they want to receive via delivery options. Instead of dreading the taste of the food they hate, they can designate their specified food to come to their location.

The idea of creating an autonomous food delivery car was sparked by the Starship Delivery Robot. As shown in Figure1, The Starship Robot is a six-wheeled ground robot that can navigate streets and sidewalks, where they offer on-demand package delivery for consumers and businesses. The Starship Robot was created and operated by Starship Technologies in July 2014.



1. Starship Robot designed by Starship Technologies

## Objective

Design an Autonomous Electric Car that can deliver dining services around Campus Universities.

1. Implementation Plan

## Tasks

Task1: Vehicle Structural Design

Subtask 1.1: CAD of Structure

Subtask 1.2: Implement Structure

Subtask 1.3: Design Lock System

Task 2: Design of Electrical Power System

Subtask 2.1: Identify electrical components

Subtask 2.2: Connect Components

Subtask 2.3: ESC configuration with Arduino

Task 3: APP Design

Subtask 3.1: Create Ordering System

Subtask 3.2: Design Payment System

Subtask 3.3: Design Food Tracking System

Subtask 3.4: Design SMS

Task 4: Design of the Artificial Intelligence System

Subtask 4.1: Design Semantic Segmentation for System

4.1.1: Gather Dataset

4.1.2: Build Semantic Segmentation Model

4.1.3: Train Model

4.1.4: Test and Evaluate Model

Subtask 4.2: Detection

4.2.1: Deploy Pre-Trained Model to Raspberry Pi

4.2.2: Segmenting mask of input image

4.2.3: Post-Processing

4.2.4: Design Real-Time Segmentation and Drive Control

4.3.5: Design Serial Communication Protocol

Task 5: Design LIDAR System

Subtask 5.1: Design Program for LIDAR sensor

Subtask 5.2: Design connection for LIDAR sensor

Subtask 5.3: Send control and steering commands

Task 6: Navigation

Subtask 6.1: Extract Geographic Coordinate Units

Subtask 6.2: Configure GPS on Arduino

Subtask 6.3: Configure Compass on Arduino

Task 7: System Testing, evaluation, and enhancement

Subtask 7.1: Test Design Requirements

Subtask 7.2: Evaluate and apply changes if necessary

Subtask 7.3: Repeat process

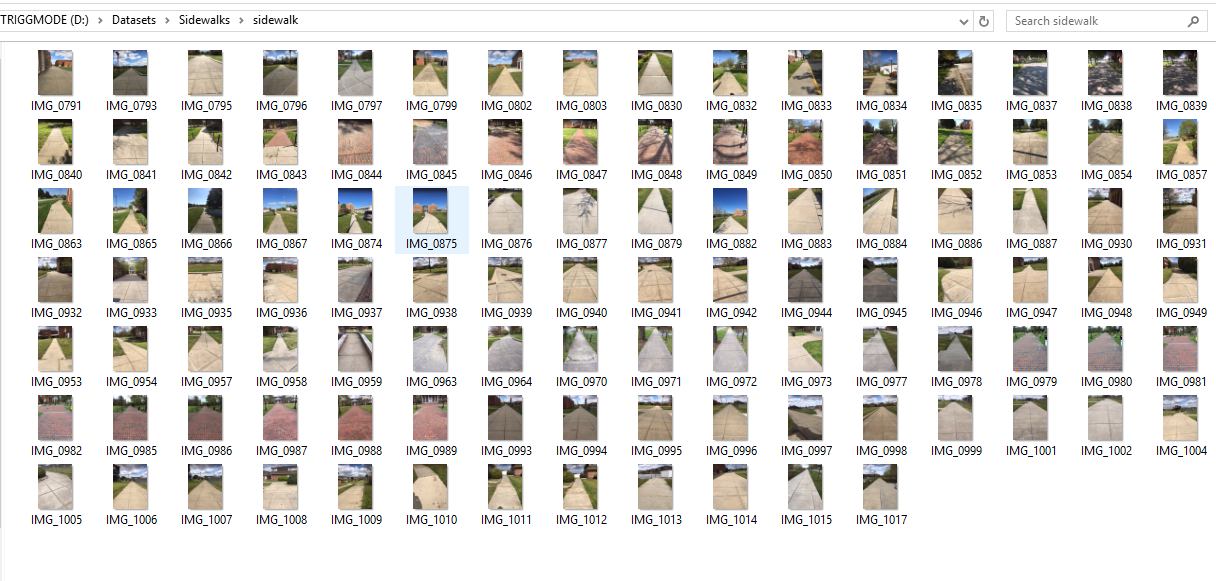
1. Implementation

## Implementation of Task 4.

### Implementation of Subtask 4.1.1: Creating dataset

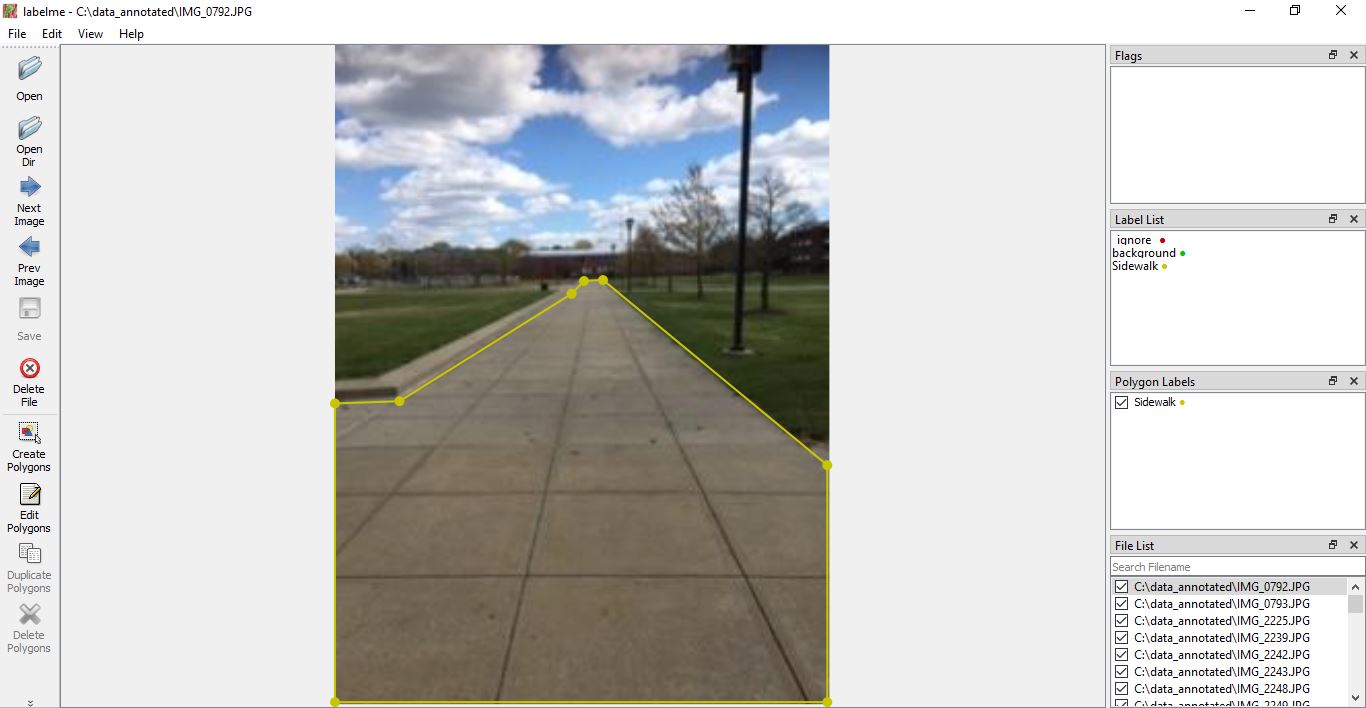
One of the most necessary goals in this project is to design a Neural Network that can differentiate and detect the sidewalk in each frame. A platform must be created to allow the robot to identify the correct path. The problem in this project is to have a Convolutional Neural Network that can detect the Sidewalk in an image. To perform this task, Semantic Segmentation must be completed. Semantic Segmentation allows the robot to detect and extract the path from the rest of the seen. The initial steps of performing this daunting task is creating a dataset which includes capturing images, annotating, and preparing the dataset to eventually feed the Neural Network.

The data is the most important information that the Neural Network must receive. The data captured is specifically prepared to identify and differentiate the sidewalk from the rest of the environment surroundings. Capturing Images is a subtask for creating a dataset for this project. In this project, over 1000 pictures was captured but only 200 images were used as the inputs. The first trial of capturing images, over 300 images, the image files were too large. The image file was 1782 by 1120, which is way too large to feed into a model that will eventually be deployed to a Raspberry Pi. The second trial of capturing images, videos was taking too speed up the process of gathering multiple frames at once. At least 600 images were extracted from these videos. After further evaluation, this method would have deterred the images resolution as they were extracted from videos. The final trial of gathering images, 200 images were manually captured via iPhone and was download to the computer as small file sizes. The size of the images was 320 by 240, which were the smallest possible size before eventually augmenting the size to a smaller size in a data bunch. Refer to figure 2, as it shows the files of the original images captured.

**

1. Original photos captured via iPhone

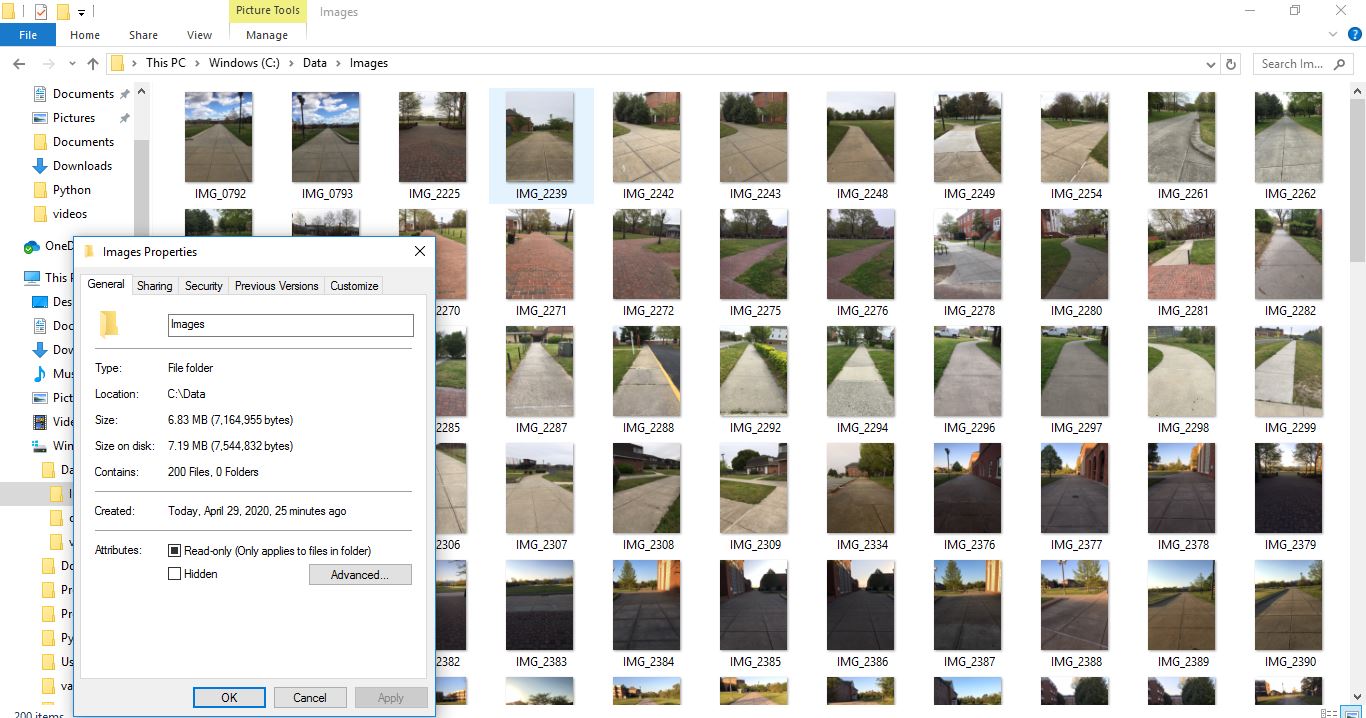
After the necessary images are captured, the next step is to annotate these images. The annotation method allows the user to manually label the images. Annotation is what differentiate between a classification Neural Network and an Image Segmentation Neural Network. As it gives every pixel in an image a label, as in this project a “Background’’ and or “Sidewalk”. This must be done with a Pixel-Wise Annotation Tool. LabelMe is an Image Polygonal Annotation application in Python that is used in the implementation process, whereas to annotate 200 of captured photos. Annotating included drawing a polygon around every sidewalk in an image and labeling it “Sidewalk”. In the same image, drawing polygons around everything but a sidewalk and labeling it “Background”. Figure 3 shows an example of a polygon drawn on a sidewalk in an image with its corresponding label.

**

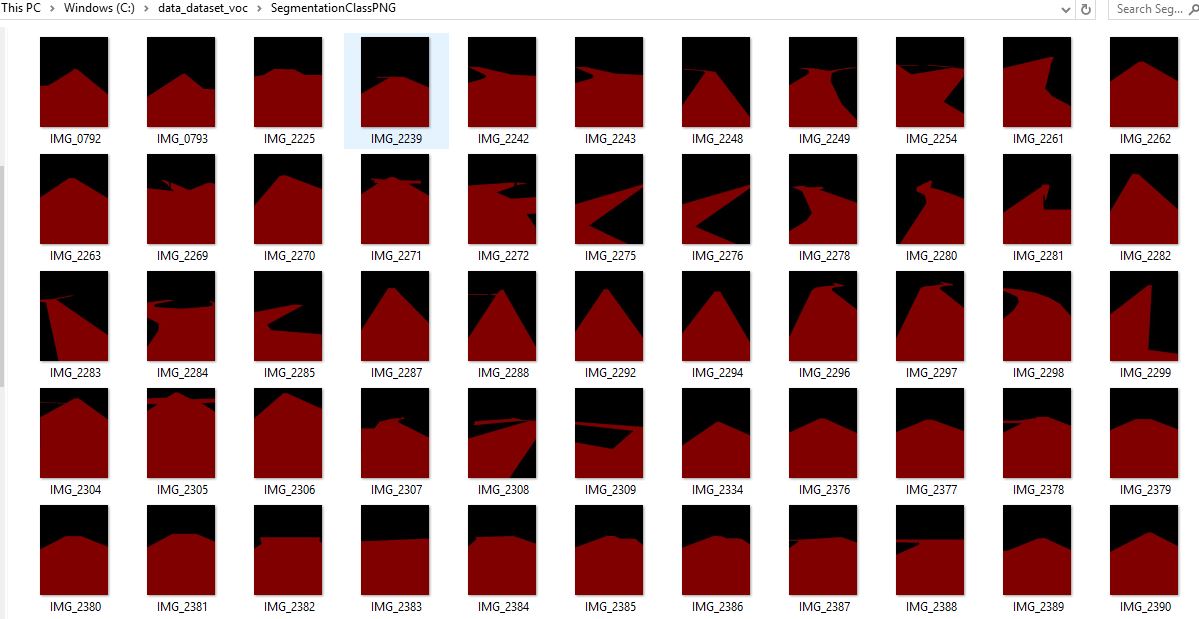
1. Annotation of image using LabelMe

There is more than just drawing the labels on corresponding images. After labeling is done for all 200 images, conversion must be performed to create a dataset that can be used in Python. When labels are created in LabelMe, it outputs a JSON file. Python needs to read the data as an object and not a string. To perform this action the JSON file must be converted to a Pascal VOC Dataset Format. The conversion is done through Python as a script. The script to do this conversion was adapted from a GitHub repository that will be cited in the reference section of the report and portion of the source code is shown in the appendix section, “Source Code of the Pascal VOC Python Format Conversion”. As the source code generally takes the inputs of the following images, JSON files and convert to an output of images, classes, and masked labels.

Furthermore preparing the dataset in the fashion necessary to perform semantic segmentation is the next step in creating a dataset. After the conversion in Python, the Pascal VOC Dataset included several folder sets. The image folder, named “JPEG Images”, included 200 files of JPEG images. Figure 4 shows the JPEG files generated from the Pascal VOC Dataset. The folder, named “SegmentationClassPNG’’, includes PNG files of masked labels. The folder named “SegmentationClassVisualization” included a mixture of masked data and the corresponding image with the labeled colors. Red representing the labeled sidewalk and black representing the labeled background. Figure 5 shows the corresponding masked label that will be used as the labeled input data in the Neural Network. The classes folder, named “codes”, basically is a text file of the corresponding labels as “Background and Sidewalk.

**

1. Folder of 200 JPEG images

**

1. Folder of 200 label images

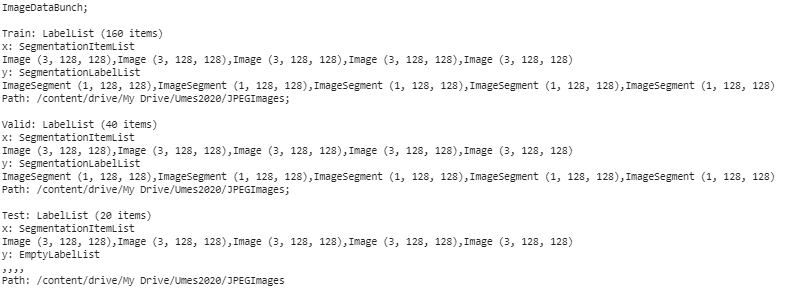
In designing the Neural Network there must be the images, the segmented images, and the classes. Also a test set folder of new images is created to test after training the Neural Network is complete. The neural network is prepared with 200 colored images along with its corresponding labels. Then there are label categories labeled either as “Sidewalk”, “Background” or “Void”. So folders are made for each category and loaded onto Python with a correct source code. This data loaded will eventually be used as input data to build the model for training. The platform used for this task was done on Google Collaboratory. Refer to the appendix, “Source code of Preparing Dataset”. In the source code, the first block demonstrates the dependencies needed for the Python Fastai library. The second block of code demonstrates gathering the dataset folders from the google drive application. The dataset that was on the computer was downloaded to the google drive application for the use of Google Collaboratory. In the third block of code, the script gathered each individual file of the input images, labeled images, and the test set images.

Furthermore, when preparing the data for the Neural Network there must be a process of checking the data of the input images. Neural Networks understand numbers and all data has numbers in the form of tensors. Therefore the masked data input must correspond with labels of “Sidewalk”, “Background” or “Void”. This is done so the output will match the corresponding masked image with a label. In the mask data in this project the tensor matrix ranged between 0 to 38. The top matrix was generally zeros which corresponds to the top portion of the masked image being the background. The bottom matrix was all 38 which corresponds to the bottom portion of the masked image being a sidewalk. A classes folder which is generally a text file with corresponding labels was created to match this need. As zero, the first text name in the text file was named “background”. Being that the data ranges between zero and 38, there must be categories to fill that spot. Thirty- eight text names were named “Void” following the “background” text name. The 38th or the 39th file names are named “Sidewalk” to match the 38-input tensor as a label named Sidewalk. After the data is prepared correctly a model must be created to train the input dataset.

### Implementation of Subtask 4.1.2: Modeling

Python has become one of the most popular platforms for deep learning, as there are many platforms that allow coding for deep learning tasks simpler than it really is. One of those Python libraries that is used design this Neural Network is called Fastai. Fastai is a Python programming application that was created to make coding for Artificial Intelligence simple. Furthermore this python was used on Google Collaboratory for this project. This is useful because Google Collaboratory is free cloud service that allows python programming and it also has free GPU. When training a Neural Network the GPU memory comes into play because if there is not enough GPU size for training than the Neural Network cannot be complete. The Google Collaboratory is a free Juypter Notebook that uses a Tesla K80 GPU and gives a total of 12 GB or ram. Implementing a GPU manually onto the computer can be very difficult and time consuming, so having a Jupyter notebook with free GPU is helpful. Furthermore 12 GB of ram is more than enough for the training of this Neural Network as there is only training on 200 images and labels. Fastai was purposely used because it is an easy to understand python language and there is Semantic Segmentation example that would help with any Semantic Segmentation problem.

A Neural Network for Semantic Segmentation in Fastai is designed to train on data that includes a training set, validation set, a batch size, image size, and a labeled function. This data is created a using a data block API. The data block API allows the coder to create a data bunch that can generate a training set, validation set, and test set with the corresponding image and batch sizes. In this data bunch there is numerous lines of codes that is generated. In the block API, the first line of code tells where to find the data. It gathers the 200 input images for the variable, “path\_img”. And then the second line of code splits the 200 input images as training set and a validation set. Implementing 160 input images into the training set and 40 input images into the training set. The third line uses the label function created to match each input image to its corresponding labels. The rest of the data bunch include making the size 128 and having a batch size of 4. This include that the images that will be going into the input layer will be 3 by 128 by 128, thus having 49,152 neurons in the input layer. The batch size implies that the training will go through 4 training iterations for every epoch. After the data bunch was created a test set was added on to the data bunch because it is unable to create a dataset in the data bunch because the test set are unlabeled. Thus the test set was added to the data outside of the data block API. Refer to the appendix, “Source Code of Data Block API and Added Test Set”, as it demonstrate the source code to create a data block API and test set that will act as the input data to feed the Neural Network. And refer to figure 6 as it shows the printed data bunch created from the data block API source code.



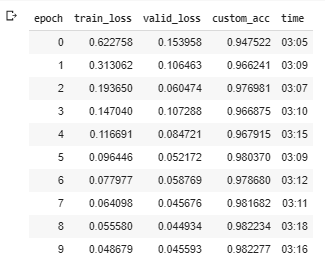
1. Input data created by data block API

Now that the input layer is initialized it can go through a model best for semantic segmentation. A U-NET is a Convolutional Neural Network originally made for biomedical image segmentation but has been proven to perform on any image segmentation tasks. The goal is design U-Net Segmentation model is that can display the sidewalks in an image as labels. To be more precise, the model being used for training is the Dynamic UNET. This may seem like a difficult task but Fastai makes this task simple for coders. A U-net network are highly efficient for semantic segmentation as the Fastai library has built in features that can quickly create a UNET model and replace the encoder with a pre-trained model to take advantage of transfer learning. A UNET architecture contains a contracting path which is also called the encoder and a symmetric expanding path also called the decoder. In the encoder part of the architecture the size of the image gradually reduces while the depth increases. This enables the neural network to determine the “what” information in the image. When this is going on the model loses the “where” information in an image. That is where the decoder part of the architecture which applies transposed convolutions along with regular convolutions come into fruition. This method of decoding the image enables the neural network to determine the “Where” information by apply up-sampling. Dynamic UNET combines transfer learning of applying a pre-trained model as the encoder. Transfer learning is a technique where the coder can use a model trained on a very large dataset and then adapt it to their own dataset. Therefore the pre-trained model used as the encoder in this Neural Network is the Resnet 34. The encoder network Resnet 34 is a model created by ImageNet that has trained a thousand images for classification. Using this for transfer learning allows semantic segmentation of one’s own dataset more accurate and less painful. A computer vision learner in Fastai is a module that is used to apply this transfer learning to the build the model. Fastai allows users to implement this by a simple line of code. This explanation is demonstrated in the appendix section, “Source Code of Creating Dynamic UNET”. This code basically builds the Dynamic UNET model and puts it into a variable called ‘learn’ to be used later for training.

### Implementation of Subtask 4.1.3: Training

Modeling and training go hand in hand in design a Neural Network. The model that is created uses the parameters designated as the batch size, image size, and number of epochs to complete training the Neural Network. When creating the UNET learner that includes the data and the pre-trained model as the encoder, coders can also pass in training metrics into the learn object. Metrics for training Fastai models are simple functions that take input and target tensors and return metrics of interest for training. Coders can customize the metrics to meet the needs they want to display. Configuring metrics in the learn object will output the training loss, validation loss, accuracy, number of metrics, and the time it takes to complete each epoch. The appendix section, “Source Code of Creating Dynamic UNET”, shows the source code for training metrics placed into the learning object.

Now that the model is completely created, there must be a search for a fitting learning rate. After the learning rate is complete then the model begin training. Refer to the appendix section, “Source code of fitting learning rate and training”, as it shows the source code of finding the fitting learning rate. Also the learner that trains the model is ‘learn. Fit’, this prints the metrics after every epoch. The number of epochs used in training is 10 epochs. With only 10 epochs training of the model was able to obtain a prediction accuracy of 98%. In approximately 30 minutes. This result is good as this will demonstrate accurate output predictions on the test set. The printed-out training metrics of the Neural Network is represented in figure 7.



1. Training metrics of Nerual Network

To summarize the training/modeling for the design of the Neural Network. The model takes in an input layer that contains images 3 by 128 by 128. Therefore there is 49,152 neurons in the input layer, that acts as the initial data to be taken from the hidden layers. Between the input layer and every other layer, there is weights that are applied to each neuron in each layer. These weights are used to help the model learn the precise predictions. In the Dynamic UNET model that was created to perform the Neural Network contained three sequential layers. A sequential layer allows the model to be layered by layered in a step by step fashion, creating complex layers. In the first sequential layer there is sixteen convolutional layers followed by 16 Batch Normalizations, one Max Pooling layer, and there is eight Rectified linear units (RELU). The second sequential layer includes 20 convolutional layers followed by 20 batch normalization layers and 9 RELU. The third sequential layer has 18 convolutional layers, 5 batch normalization layers, 22 RELU, 2 merge layers, 5-pixel shuffles, 5 pooling average layers, and 5 replication padding layers. Every neuron in a Neural Network has an activation functions that produces the output of the neuron. The summary of the Dynamic UNET architecture built is a combination of determining the “what” and “where” information in the training. After going through this model of training the neural network can produce an output of predictions on input images. In Fastai, after training is completed the model can be saved in a safe directory and later can be used with an inference learner.

### Implementation of Subtask 4.1.4: Testing

The next step is to test the accuracy of 98% on the unlabeled test set images by producing predictions. The predictions should produce a masked label of the input image. To complete this action the saved model is loaded into python, only if the user wants a separate testing script. This is done by a method in Fastai called inference learner, as it basically learns the model that was created and use the data needed without performing any training. When a saved model is loaded it contains it designated weights and biases in the model. In training a Neural Network, the weights and biases are initially designated at random and then goes through a method called backpropagation where it makes the weights and biases more accurate for predictions. So the model uses these weights and biases to produce the predictions on the images the coder wants to produce predictions on. Furthermore from these weights and biases the trained model obtains parameters. The parameters are the weights and optionally biases in a neural network and the parameters can be obtained for each layer. In each layer of the model it can show if the layers are trainable or non-trainable. Therefore in this Neural Network there is 41,224,868 total parameters. 19,957,220 of those parameters are trainable and 21,267,648 of those parameters are non-trainable.

The testing of the neural network consists of carrying predictions on each individual image in the test set. As discussed earlier the test set includes 20 new images that were not included in the training and the validation sets. This is important as the training and validation set have corresponding labels to them, while the test set has unlabeled data. Therefore the Neural Network will make prediction of the labels on new information. In Fastai, the library is unable to calculate the accuracy on the test set because the test set is meant to be unlabeled. Therefore once the training is done the coder can use the trained model and test the predictions on a single image.

The code use to get predictions on a single image is ‘learn. Predict ()’, as it returns as predicted class, labels, and probabilities for an item. Also code it implemented to show the predicted output of the single image. The source code that demonstrates these actions is shown in the appendix section, “Source Code of Predictions on Test Set”. The predicted output was done for each image in the test set and was able to show accurate prediction outputs for all. Predictions on one of the images in the test set is shown in figures 8. The figure shows the input image in the test set followed by its prediction that was activated by the source code. As displayed in the figure, differentiation is created between the sidewalk and the background. As the output mask of the sidewalk is labeled as yellow. And the output mask of the background is labeled as purple.



1. Prediction on test set image

This implementation of the project focuses on implementing Semantic Segmentation on a pascal VOC dataset using a Dynamic UNET. This was done with a Fastai library on the Google Collaboratory cloud service. The accuracy achieved from training is 98% with an output image size of 128 x 128. This saved model should be able to be deploy to another application such as the Raspberry Pi. As the model can correctly identify the sidewalk. The sidewalk can be designated as the correct path that the Raspberry Pi should visualize with the camera. In the Raspberry Pi operating system, Fastai library can be installed to try to produce the similar results as done on the Google Collaboratory Python Cloud Service.

### Implementation of Subtask 4.2.1: Deploy Pretrained model on Raspberry Pi

The goal is to take the pre-trained model that was trained on the Google Collaboratory GPU platform and export the learned model on the Raspberry Pi 3 B + microcomputer. This objective offers many daunting tasks as exporting semantic segmentation logic onto a different machine may pursue numerous issues. Issues including knowledge of Raspberry Pi and Raspbian Operating System, Compatibility, performance and accuracy, etc.

The first initial step in this task, is setting up the Raspberry Pi and the Raspbian Operating system. The basic setup for the raspberry pi is to apply heat sinks to the CPU and the chip that controls the ethernet and USB cable ports because of the possibility of overheating. Then the SD card can be inserted into the SD card socket. The SD card is used to install the Raspbian Linux as the Raspberry Pi operating system. To install Raspbian, a keyboard, mouse and a display monitor needs to be connected to the Pi. Once the Raspberry Pi is powered on, it will load the Noobs tool. The NOOBS tool was already configured in the SD card, so this enables direct installation of Raspbian Linux.

A Camera, Raspberry Pi Camera V2, is one of the most essential components in this project. As it acts as a sensor to take input images of the surrounding and feed its computation to the Raspberry Pi B +. Without input images and the camera, Semantic Segmentation will be unsuccessful. Therefore, a test of seeing if the Raspberry Pi Camera V2 is working correctly with the Raspberry Pi 3 B + is necessary. To set up the Raspberry Pi Camera V2, place the bottom tale into the MIPI camera serial interface and power on the raspberry pi. Then use the Raspberry Pi Configuration tool to enable the Camera interface. Raspberry Pi Camera V2 configuration is displayed in figure 9. The best way to check if the camera is working properly is write up a code in the raspberry pi command prompt. This command can be “ raspistill -o Desktop/image.jpg”, this will enable a still picture to be completed. However, for the most part this project will include python scripts using the PiCamera class library which enables intuitive projects to be completed with the Raspberry Pi Camera.

A circuit board

Description automatically generated

Moving on to the process of deploying the pre-trained model on the Raspberry Pi model B +. There are software dependencies that to be in place to deploy the model on the machine without any errors. The model was pre-trained with a Pytorch version of 1.4.0, Pytorch Vision version as 0.5.0, and a Fastai version as 1.0.61. In the python environment on the raspberry pi, those dependencies must be installed. To install the Pytorch, pre-build Pytorch wheels were downloaded and then installed on the raspberry pi python distributions. Also, the fastai version that is installed must be compatible with the Pytorch version 1.4.0 and 0.5.0. Therefore 1.0.61 fastai version was installed on the raspberry pi instead of the fastai v2. Once the Python, Pytorch, and Fastai version are installed correctly the exported model can deployed by simply downloading the exported pkl file on the raspberry pi 3. The pkl file can be called into the python script to be loaded and make predictions on the input images.

### Implementation of Subtask 4.2.2: Segment mask on input images

To test the model that has been deployed from Google Collab GPU to the Raspberry Pi # B +, inferencing take place. Where input images are past through the model for predictions of the output segmented mask. This is important to test because the model needs to be tested to see if the masks are still accurate after been exported to a different machine. To test the model, a batch of test sets images where inferenced to see the segmented masks of each. The test images acting as the input for inferencing have a size of 240 by 320. Also, these images they were taken manually by a iPhone camera. After inferencing the segmented image should have a tensor shape 1 by 128 by 128 and should display a mask with at least 97 percent accuracy. Twelve different input images were inferenced by the deployed model and it took a total of 67.8 seconds to segment every input image. Approximately 5.6 seconds inference time per input image. The semantic segmentation result of inferencing on one of the input images is displayed in figure #.

A close up of a sign

Description automatically generated

In this project, the model will be making predictions on input images from the Raspberry Pi Camera V2. Capturing images from the raspberry pi camera and feeding it into the model to receive output mask is an even better way to test the Neural network. As this method is applicable to the Autonomous Food Delivery Vehicle. Python allows the user to use the PiCamera library to do amazing projects with the Raspberry Pi Camera V2 and in this project capture input images of the surrounding sidewalks around campus. 50 photos of the pavement around the Engineering and Aviation Science Complex were capture by the Raspberry Pi Camera and python script saved those photos. Then the photos were taken and passed through the learning model to predict the resulted mask. Fifty mask results were accurate and can differentiate between the sidewalk and the background. However instead of saving the captured image and then predicting the image in a later time. The best scenario is to capture the input image and predict the output mask all in real time. As the car will be moving, photos need to be taken and processed in a significant amount of time to give commands to help control the car. Therefore, a python script was made to capture the input image of the sidewalk every 1 second and output the mask of the input image in real time. Timing is essential to this project , so it is needed to capture the time it takes to load the model, capture the input image, and predict on the input image.[Continued….]

### Implementation of Subtask 4.2.3: Post-Processing

Once the captured input image is fed through the Neural Network model in real time, a segmented mask is produced as an result. The segmented mask diferentaite the sidewalk from the rest of the input image. The goal is to drive on the sidewalk, so segmentation of the image is not the end result. Post-processing of the segmented image has to take place in order for the camera and ultimately the car to know what direction to follow on the sidewalk. The approach is to take the input mask and establishing motion planning for the autonoumous food delivery vehicle.The method includes designing lane navigation of the masked sidewalk to determine the heading direction and untimately the steering angle. This post-processing of the segmented image is done by using Python-OpenCV Library.

The first step in post-processing of the segmented image is capturing the segmented frame. Once the input image from the pi camera it is feed through the model to obtain the mask prediction. From there the mask needs to be captured to feed through post-processing. The mask can be saved in the memmory so the post-processing can be completed. The mask can differentiate the sidewalk by doesnot have any understanding of where to go. A method is to obtain the outside lane or skeleton of the sidewalk from the masked image. This is done with edge detection methods through opencv. Obtaining the edges of masked image is the best approach because the only edge detected will be the ones important for control. As if taking a raw input image will detect multiple edges within the sidewalk to be detected. When the edges of the mask sidewalks are obtained it produces two lanes to navigate within.

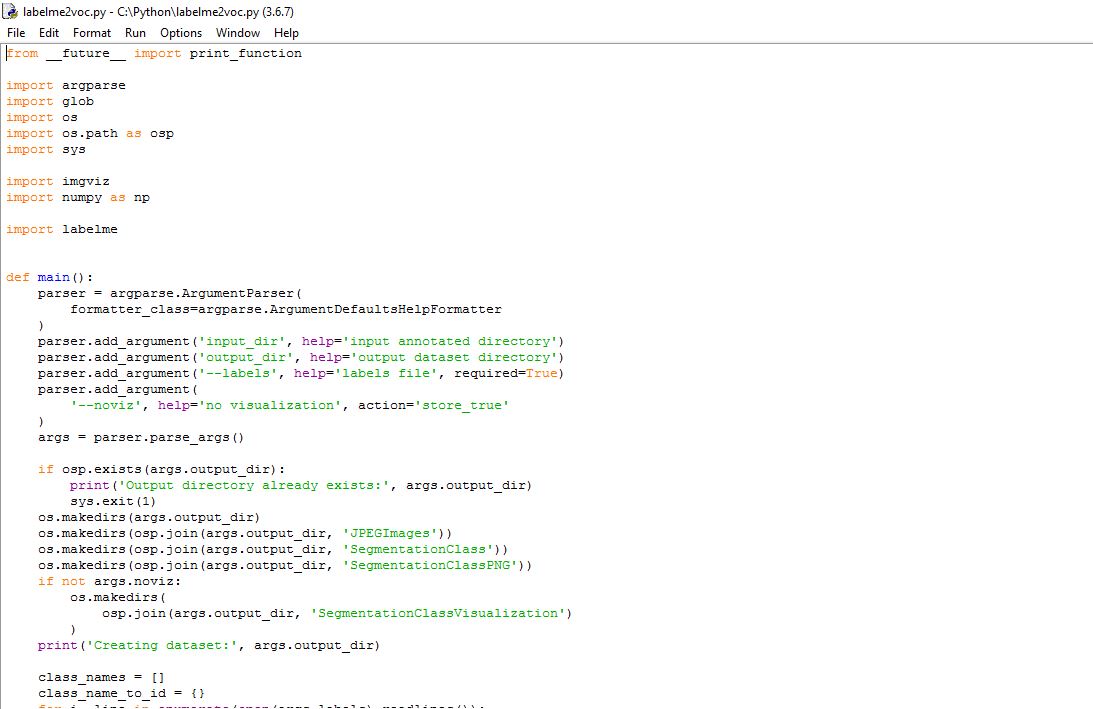
The purpose is to generate steering commands that will allow the car to stay in the middle of the lane. This is simply done by establishing the heading direction from the input. The heading direction is computed by averaging the two far endpoints on the two sidewalk lane lines. The image result of post-processing a segmented image to determine the heading direction of the car is shown in figure #. Ultimately, the heading direction is converted to the steering angle.

A screenshot of a social media post

Description automatically generated

Appendix

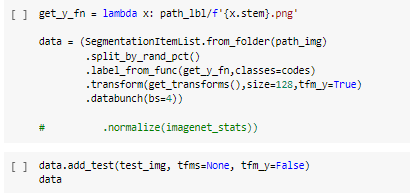
1. Source Code.
2. Source Code of Pascal VOC Format Python Coversion



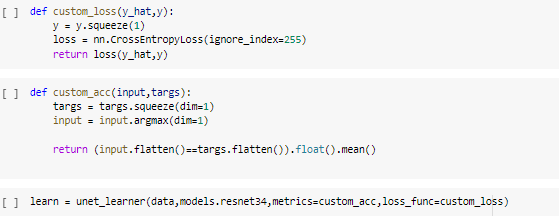
1. Source Code of Preparing Dataset



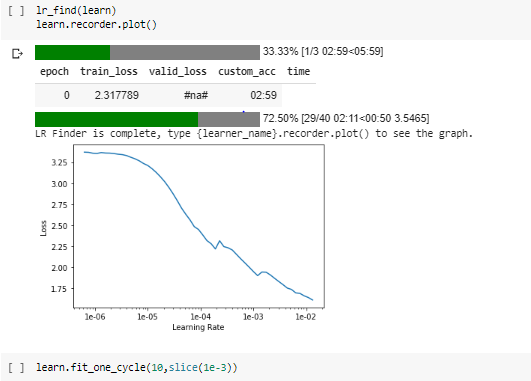
1. Source Code Data Block API and added Test Set



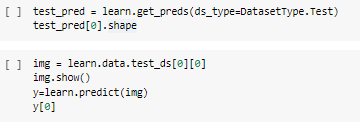
1. Source Code of Creating Dynamic Unet



1. Source Code of fitting learning rate and training



1. Source Code of Predictions on Test Set



REFERENCES

[1] Docs.fast.ai. 2020. | Fastai. [online] Available at: <https://docs.fast.ai/index.html> [Accessed 2020].

[2] Medium. 2020. Image Segmentation With Fastai. [online] Available at: <https://towardsdatascience.com/image-segmentation-with-fastai-9f8883cc5b53> [Accessed 2020].

[3] Wada, K., 2020. Wkentaro/Labelme. [online] GitHub. Available at: <https://github.com/wkentaro/labelme> [Accessed 2016].

[4] GitHub. 2020. Keyurparalkar/Semantic-Segmentation-With-Unets. [online] Available at: <https://github.com/keyurparalkar/Semantic-Segmentation-with-UNETs.git> [Accessed 4 June 2020].