Due: May 3rd, 01:25 PM	CMPEN 497: Applied Computer Vision & Machine Learning for IoT Applications - Spring 2018	Course Project Documentation
	Project: Object Detection and Recognition for Autonomous Cars	
Project Report [System Design]		

Team Members Submission Statement:

We confirm that we followed the course policies by contributing equal amount of work to this assignment. Below is the approximate ratios of our contributions.

Name	PSU email	Contribution %	Signature
Brandon Bench	btb5124@psu.edu	50%	Brandon Bench
Nick Demarco	njd5223@psu.edu	50%	Nick DeMarco

Document Outline & Grading Rubric

Component	Grade
Project Description ● Abstract ● Introduction	/ 10
Systems analysis, requirements definition	/20
Systems design ● High Level Architecture ---------- /10 ● Level (1) Design ---------- /25 ● Level (2) Design ---------- /25	/ 60
Report Presentation (you don't need to create PPT slides, you will just stand before the class to present and discuss your report)	10
TOTAL	/ 100

Table of Contents

I. Project Description

A. Abstract

There are many methods to implement object detection for self driving vehicles. These methods allow the vehicle to determine if the object is hazardous or not. The overall purpose of our project is to record video to be later analyzed to detect and recognize various objects that are necessary for self driving vehicles. For example pedestrians, street signs, other vehicles, etc. We aim to use computer vision and machine learning to combine already designed algorithms and methods to create an object detection program.

B. Introduction

Autonomous driving is at the forefront of current technology, many startups, and lots of investment dollars. Huge companies such as Uber, Waymo, Apple, Intel, Tesla, are all vying for the prestigious spot to be the first company to have a completely unmanned self driving vehicle. Autonomous vehicles have the potential to revolutionize the way society uses transportation, ships goods, among many other use cases. Our project aims use computer vision and machine learning while focusing on just a tiny piece of the autonomous puzzle: object recognition. Our system will analyze recorded video to detect and recognize important objects such as pedestrians, street signs, other cars, and any obstructive objects that a vehicle must recognize to drive safely. The video will also show where these recognized objects are, by plotting markers around each. In order to detect objects, we are going to create a Convolution Neural Network. This network will use layers and filters to create a model that we will use to detect and classify objects.

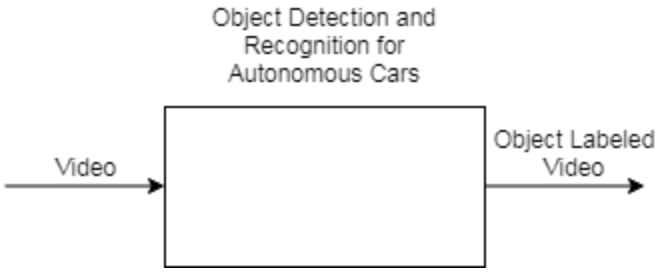
II. System Requirements

- Read in video
- Seperate frames
- Detect objects (street signs, cars, etc.)
- Classify and label object in frame
- Create finished video

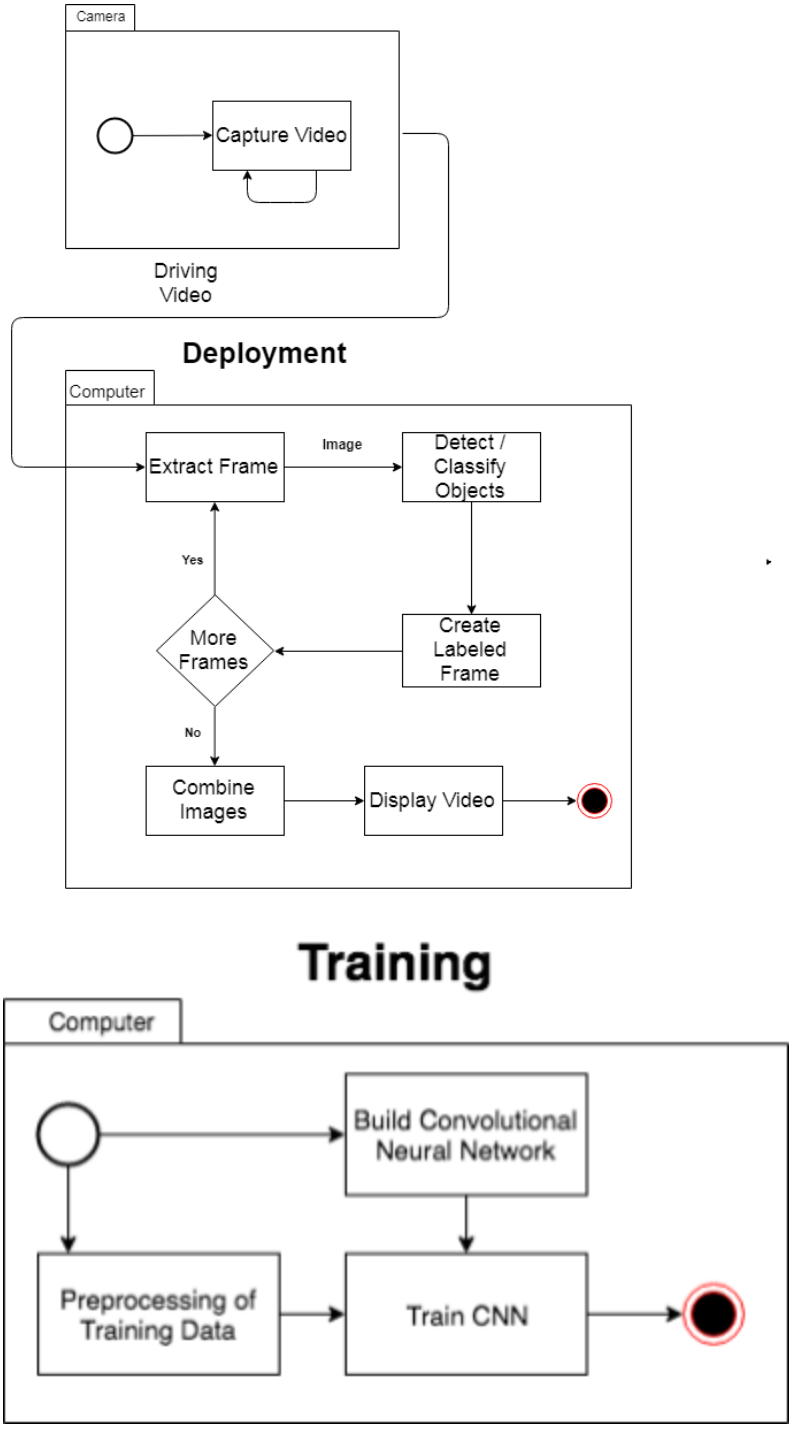
III. System Design

A. Architecture

B. Top Level Design

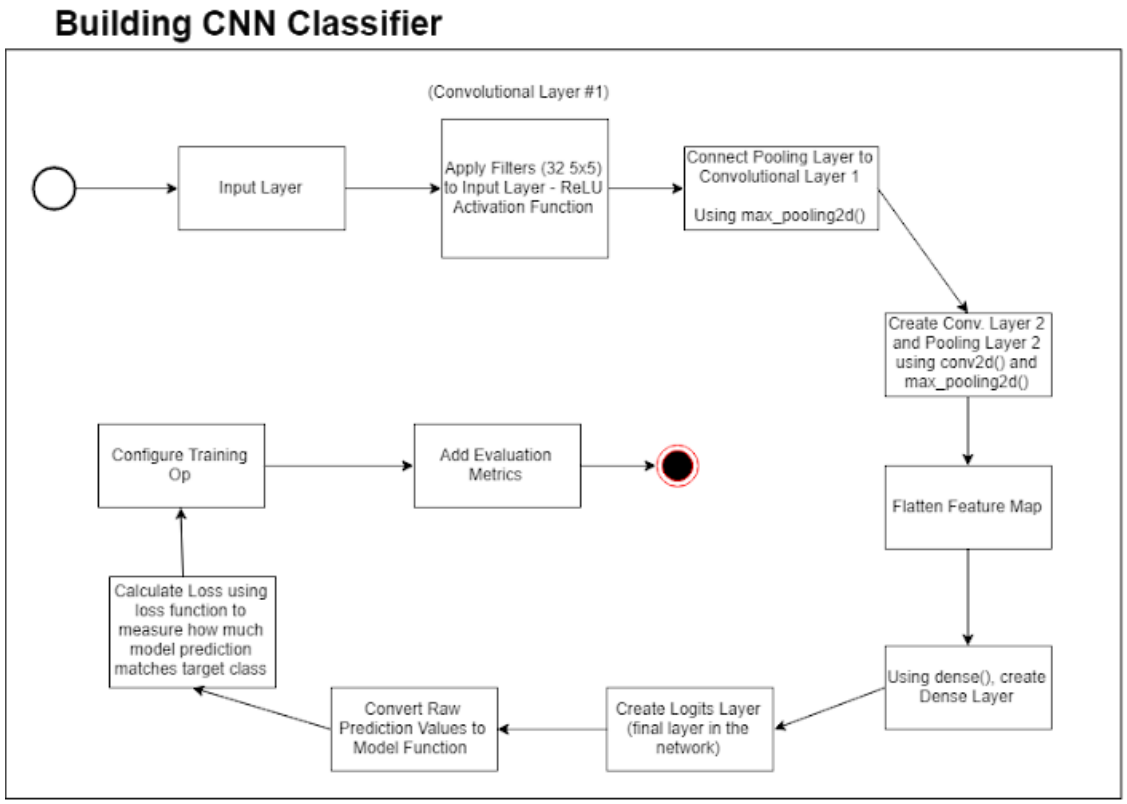


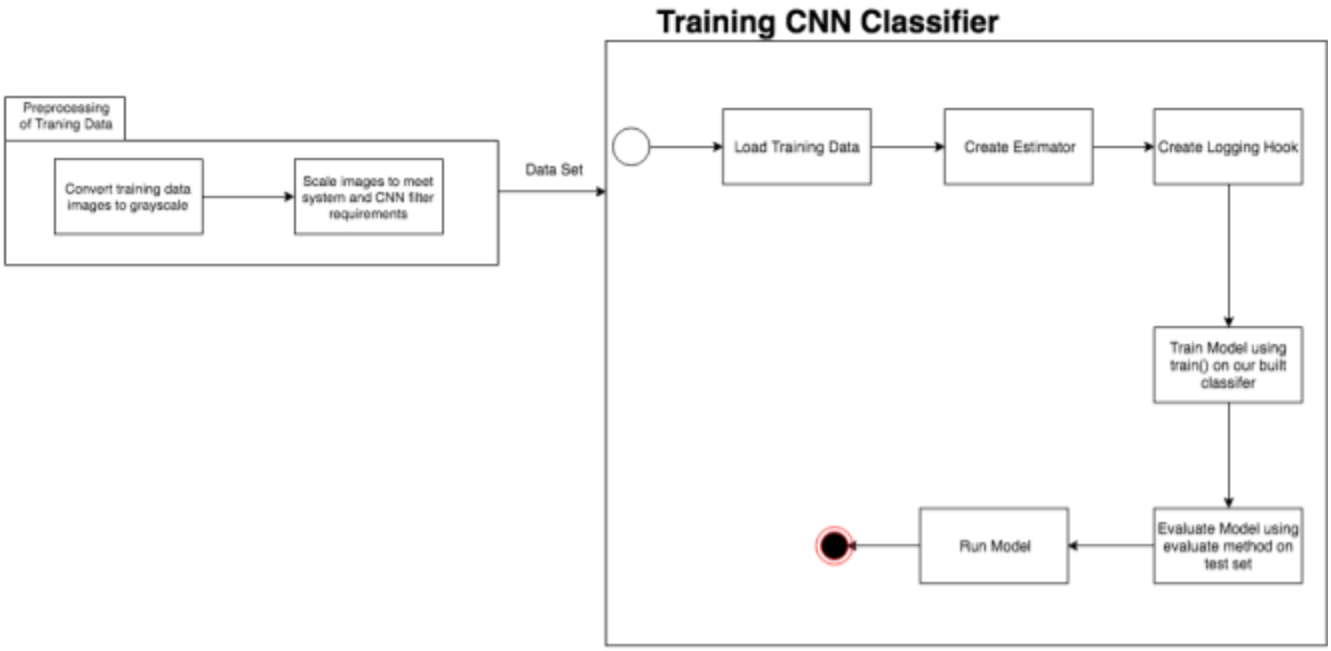
C. Level (1) Design

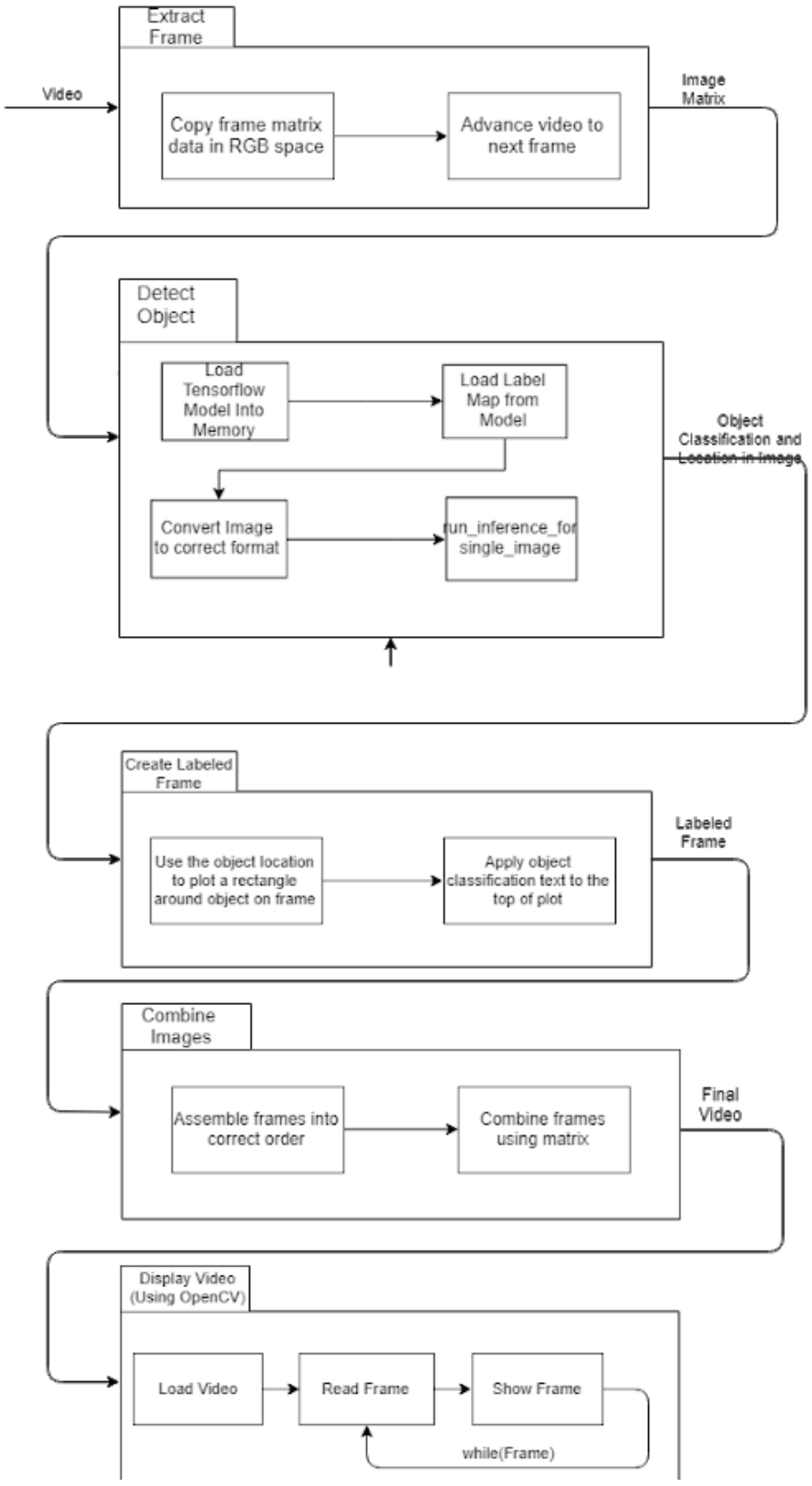


D. Level (2) Design

[1]






Extract Frame
Video
Copy frame matrix data in RGB space
Advance video to next frame
Image Matrix
Detect Object
Load Tensorflow Model Into Memory
Load Label Map from Model
Convert Image to correct format
run_inference_for single_image
Object Classification and Location in Image
Create Labeled Frame
Use the object location to plot a rectangle around object on frame
Apply object classification text to the top of plot
Labeled Frame
Combine Images
Assemble frames into correct order
Combine frames using matrix
Final Video
Display Video (Using OpenCV)
Load Video
Read Frame
Show Frame
while(Frame)

IV. Appendix

A. UML Diagram

General UML Diagrams that are helpful for you are:

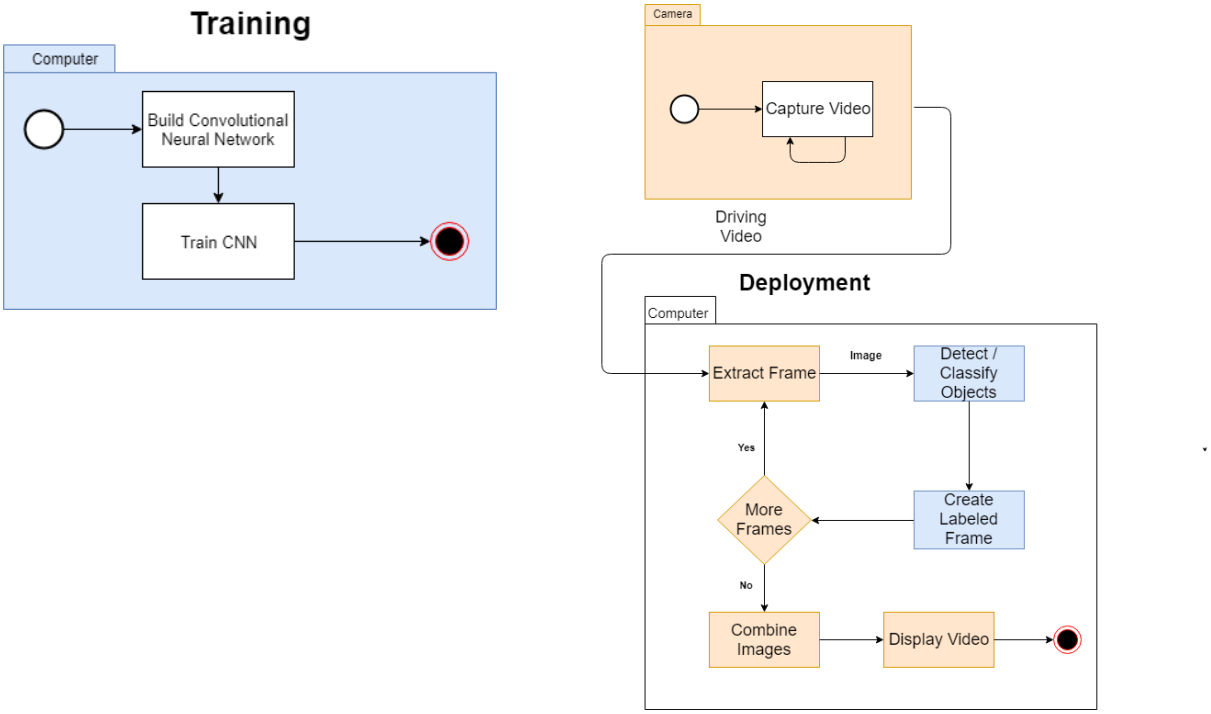
- Structural Diagrams
    - Class diagram
    - Object diagram
    - Component diagram
    - Deployment diagram

- Behavioral Diagrams
    - Sequence diagram
    - Collaboration diagram
    - Statechart diagram
    - Activity diagram
    - Use case diagram

- Here is a link to full tutorial with examples on using the UML diagrams:
    - https://www.tutorialspoint.com/uml/uml_standard_diagrams.htm
- There are several free online tools that you can use to create several system design diagrams including standard UML diagrams. I recommend using draw.io, if you do not already have a favorite one because this dramatically simplifies the learning curve. Here is a link to it:
    - https://www.draw.io/

Milestones

- Milestone 1: Train the CNN on a dataset that contains cars, people, and stop lights. Implement the CNN in python taking a single image as input, and classifying any objects existing as output.
- Milestone 2: Script that will take a video and label all detected objects and create a new output video.



Sprints

- Sprint 1:
    - Time: 2 days
    - Setup development environment in python. Download TensorFlow and OpenCV libraries needed for this project.
    - Start looking for and finalizing the training data sets we will be using. Should consist of cars, pedestrians, and street signs.
- Sprint 2:
    - Time: 3 days
    - Build the CNN
    - Run an example project that detects and classifies objects using a CNN.
- Sprint 3:
    - Time: 4 days
    - Preprocess data set. Convert all data sets into grayscale values, and appropriate sizes that will work with our CNN.
- Sprint 4:
    - Time: 1.5 Weeks
    - Train and Test CNN
- Sprint 5:
    - Time: 3 days
    - Collect video fragments from multiple road tests of different lengths and different objects.
- Sprint 6:
    - Time: 1 Week
    - Take video input and extract each frame separately. Convert each to grayscale and the appropriate size. Pass each frame to the completed CNN to receive output. Label objects in each frame and draw a box around each. Reconstruct video.

Environment Setup

Installation

Make sure that you have downloaded either the 2017 or 2018 version of Matlab. You should also make sure that you have the Computer Vision and Neural Network toolboxes installed in your Matlab toolbox libraries.

When working in Matlab, you will also want to make sure that you are working out of the directory where all of the project matlab code is stored. If you do not change the working directory, you will run into errors at execution. To do this, right click on the name of the file at the top of Matlab's editor and the first option will be to change the current folder. Choose this option and you should be good to go.

renameImages.m

Line 5: Rename project directory to the directory that contains the dataset of 100 images
Line 11: Save files to which ever directory you want to save the renamed images to.

InputPreprocessing.m

Line 16: Change the directory to the directory which contains your renamed images. This should be the same directory as the one you created in renameImages.m. Keep the last part "%d.jpg" because this will get the exact filepath for the images.

CarDetect.m

No changes need to be made to the matlab code.

Execution

To run the code, all you need to do for each matlab file is click *Run* at the top of the program. No further action is necessary for system execution. However, depending on how you want the results to display, you can uncomment the alternative display code at the bottom of CarDetect. The order of execution is as follows.

1. renameImages.m
2. InputPreprocessing.m
3. CarDetect.m

When run in this order, you will get the results displayed below.

System Results

In the images below, you see a comparison between the original image and the downsampled image using wavelet preprocessing. The wavelet that we used to obtain our results is Coif4. We chose this type of wavelet based on our previous knowledge as well as results from our previous course projects. When getting these results, we did not have to make any changes to the input layer on the classifier's neural network.

Original vs. Wavelet Preprocessing

Original vs. Wavelet Preprocessing


Original vs. Wavelet Preprocessing
Car: 55.84%
Car: 74.04%
Car: 77.61%
Car: 56.21%
Car: 88.9%

Grayscale Results

These results were captured after applying the wavelet transformation on a grayscale image. Original image size was larger than wavelet size. Initially they were equal.

Original vs. Wavelet Preprocessing


Original vs. Wavelet Preprocessing
Car: 53.98%
Car: 73.41%
Car: 67.3%
Car: 92.19%
Car: 98.83%

System Evaluation

The car detection and classification system performs well. It is able to recognize some automobiles, however does have a high chance of false positive results. We speculate that the amount of training data is to a large degree the main hindrance to accurate results. The training set consisted of about 295 images. There is an extremely large amount of variation in automobiles (size, body styling, color, lighting, angles), therefore we can conclude that 295 images is not enough to build an accurate network.

When comparing the original image with the wavelet preprocessed image, we noticed that the wavelet image appeared to have a higher accuracy rate. It was better able to identify vehicles more often. We also tested our system taking a grayscale wavelet image compared with the original image. Once again, we found that the grayscale image was also superior to the original image. A conclusion can be drawn from these results. Our network is able to better identify cars when the input is preprocessed as a grayscale wavelet. The lack of our networks accuracy due to training set size we believe, is the reason wavelets appear to be classified better. These images have less detail and are significantly smaller. The grayscale wavelets are over 3 times smaller.

References

[1] [Online]. Available: https://www.tensorflow.org/tutorials/layers#getting_started

Dataset: https://github.com/udacity/self-driving-car/tree/master/annotations