Lane Detection with Deep Learning

Project report submitted in partial fulfillment of the requirement for the degree of Bachelor of Technology

in

Computer Science and Engineering

by

SURAJ GUPTA 171282

under the supervision of

Dr. HARI SINGH

JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY WAKNAGHAT, SOLAN-173234

CERTIFICATE

We hereby declare that the work presented in this report entitled ***"Lane Detection with Deep Learning"*** in partial fulfillment of the requirements for the award of the degree of Bachelor of Technology in Computer Science and Engineering submitted in the department of Computer Science& Engineering and Information Technology, Jaypee University of Information Technology, Waknaghat an authentic record of our work carried out under the supervision of Dr. Hari Singh.

The matter embodied in the report has not been submitted for the award of any degree or diploma.

Suraj

Suraj Gupta 171282

This is to certify that the above statement made by the candidate is true to the best of our knowledge.

[illegible]

Dr. Hari Singh

Assistant

Professor

Computer Science

Department Dated : 16

May 2020

ACKNOWLEDGEMENT

ACKNOWLEDGEMENT

We are highly indebted to all the members of the Computer Science Department, Jaypee University of Information Technology for their guidance and constant supervision as well as providing necessary information regarding the project and also for their support in completing the project.

We would like to express our gratitude towards Dr. HARI SINGH, Assistant Professor for his kind cooperation and encouragement which helped us in completion of this project and for giving us such attention and time.

TABLE OF CONTENT

TABLE OF CONTENT

TABLE OF FIGURES AND TABLES

ABSTRACT

Lane identification is to distinguish lanes out and about and give the precise area and state of every lane. It fills in as one of the critical procedures to empower current helped and autonomous driving frameworks. Nonetheless, a few extraordinary properties of lanes challenge the location strategies. The absence of particular highlights makes lane location calculations will in general be befuddled by different items with comparative neighborhood appearance. Additionally, the conflicting number of lanes on a street just as assorted lane line designs, for example strong, broken, single, twofold, blending, and parting lines further hamperthe performance. In this paper, we propose a profound neural organization based strategy, named Lane Net, to separate the lane detection into two phases: lane edge proposition and lane line localization. Stage one uses a lane edge proposition network for pixel-wise lane edge arrangement, and the lane line nearby ization network in stage two at that point recognizes lane lines dependent on lane edge recommendations. If it's not too much trouble note that the objective of our LaneNet is worked to identify lane lines just, which presents more challenges in smothering the bogus discoveries on the comparable lane marks out and about like bolts and characters. In spite of the relative multitude of challenges, our lane discovery is demonstrated to be powerful to both parkway and metropolitan street situations without depending on any suppositions on the lane number or the lane line designs. The high running rate and low computational expense invest our Lane Net the ability of being sent on vehicle-based frameworks. Tests approve that our Lane Net reliably conveys extraordinary performances on true traffic situations.

Chapter-1 INTRODUCTION

1.1 Introduction

Recognizing lanes out and about is a typical errand performed by all human drivers to guarantee their vehicles are inside lane imperatives when driving, to ensure traffic is smooth and limit odds of crashes with different vehicles in close by lanes. Essentially, it is a basic errand for a self-sufficient vehicle to perform. Incidentally, perceiving lane markings on streets is conceivable utilizing notable PC vision methods. We will cover how to utilize different methods to distinguish and draw within a lane, figure lane arch, and even gauge the vehicle's position comparative with the focal point of the lane.

Optical picture based path identification technique is a key component of present day driving help frameworks. In any case, the recognition of paths stays trying for a few reasons. To begin with, the presence of a path is normally amazingly straightforward, which gives no muddled or unmistakable highlights for path location, and significantly expands the danger of bogus positive identifications.Besides, various path designs, for example, strong, broken, parting, and blending paths make independent scratch path demonstrating troublesome. Calculations dependent available made highlights can just understand the path location in restricted situations. What's more, most existing techniques additionally require severe presumptions on paths, for example paths are equal [1, 8, 14, 3], pathsare straight or near straight [11, 15], which are not generally legitimate particularly in metropolitan circumstances. As of late, a few strategies [20, 6] have been proposed to address path identification under couple of suspicions to the paths, yet there is as yet a huge space for additional advancement on the strength to assorted genuine situations. Strategies dependent on profound neural organizations [7, 10, 4], particularly convolutional neural networks animate a promising exploration bearing and furthermore motivate the possibility of our LaneNet. Besides, considering that the path location runs on vehicle-based frameworks, where calculation assets are seriously restricted, the computational cost of a path recognition strategy should likewise be considered as a vital pointer of the general presentation.

In this task, to make progress toward a summed up, low computation cost, and constant vehicle-based arrangement, we propose a path discovery strategy called LaneNet. The supportive of presented LaneNet separates the path discovery task into two phases, for example path edge proposition and path line restriction, respectively; and each includes a free profound neural organization. In the pathedge proposition stage, the proposition net- work runs parallel grouping on each pixel of an information picture for producing path edge recommendations, which are filled in as the contribution to the path line restriction network in the subsequent stage.

The neural networks in the two phases are intended for high exactness, low computational expense, and high running velocity. In particular, a light-weight encoder- decoder design is distinguishable convolution and 1 convolution layers are utilized for pathedge proposition, where stacked depth wise for quick component encoding, and non- parametric deciphering layers for quick element goal recuperation. The acquired supportive of postal map is then changed to path edge arranges and took care of to the subsequent stage, where, a fast path line neighborhood itemization organization, comprising of a point include encoder and a LSTM decoder, restrict the path lines heartily under various situations. Such two-stage plan of LaneNet brings extra desirableproperties. Initially, the path edge map created by the proposition network fills in as interpret able transitional highlights, which somewhat eases the effect of the discovery property of neural organization based technique, and makes the recognition disappointments more identifiable. The two- stage measure permits the boundaries of path line restriction net-work to be refined in a pitifully regulated way which alleviate the solid interest for all around clarified preparing samples. Furthermore, a proficient dimensionality decrease is performed while changing the path edge proposition guide to the path edge organizes between the two phases, which further lessens the multifaceted nature and the organization size of the path line restriction organization, and accelerate the identification with no trade off to the precision. Last however not the least, the capacity of the path edge proposition organization can be integrated into the semantic division organization and further lessens the general computational expense of the driving collaborator frameworks.

1.2 Problem Statement

Human beings, when fully attentive, do quite well at identifying lane line markings under most driving conditions. Computers are not inherently good at doing thesame. However, humans have a disadvantage of not always being attentive whether it be because of changing the radio, talking to another passenger, being tired, under the influence, etc. while a computer is not subject to this downfall. As such,if we can train a computer to get as good as a human at detecting lane lines, since it is already significantly better at paying attention full-time, the computer can take over this job from the human driver.

Using deep learning, We will train a model that is more robust, and faster,than the original computer vision-based model. The model will be based on a neural network architecture called a "convolutional" neural network, or "CNN" for short, which is known to perform well on image data. This is a great architecture candidate since We would feed the model frames from drivingvideos in order to train it. CNNs work well with images as they look first for patterns at the pixel level groups of pixels around each other, progressing to larger and larger patterns in more expanded areas of the image.

1.3 Objectives

1. Getting knowledge with the neural networks working

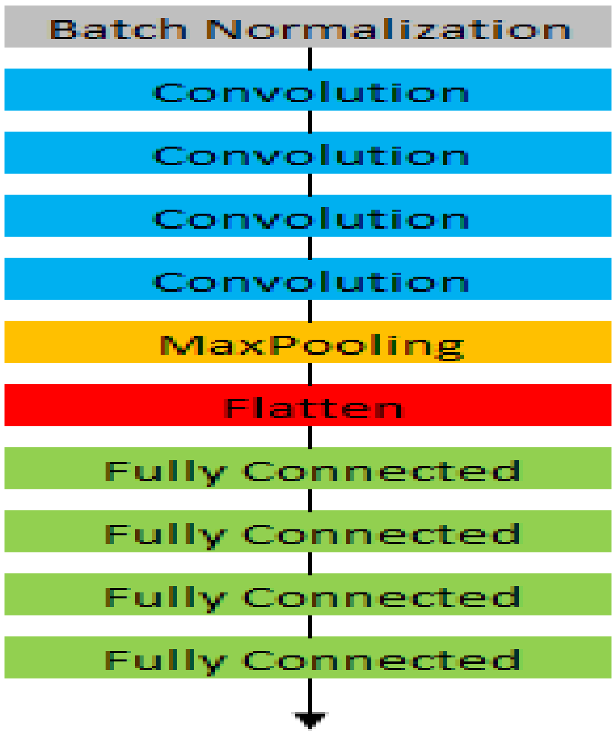
2. Developing a system for lane detection

3. Evaluating the implemented system on real data.

4. Implementing ml/dl algos.

5. Finding best ml/dl algorithm for the lane detection.

1.4 Procedure



1.5 Methodology

The entire procedure of developing the model lane detection usingdeep CNN is described further in detail. The complete process is divided into several necessary stages in subsections below, starting with gathering videos for localization process using deep neural networks

1.5.1 Datasets and inputs

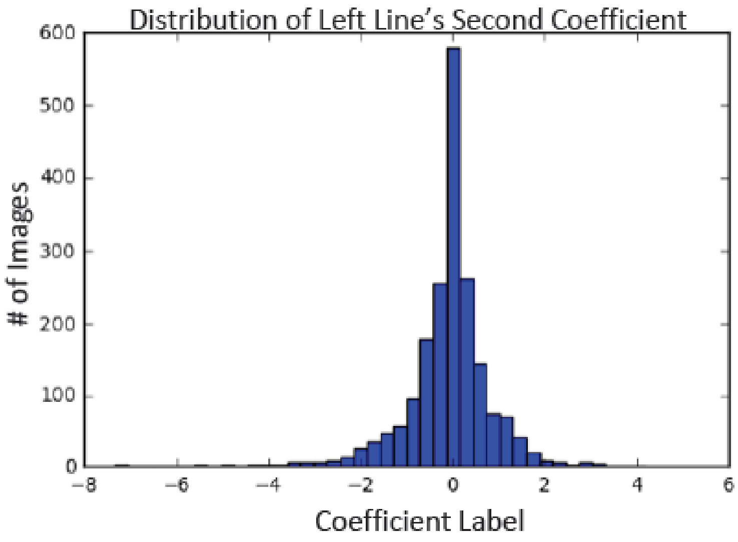
The datasets we used for the project are image frames from driving video we took from our smartphone. The videos were filmed in 720p in horizontal/landscape mode, with 720 pixels on the y-axis and 1280 pixels on the x-axis, at 30 fps. In order to cut down on training time, the training images were scaled down to 80 by 160 pixels (a slightly different aspectratio than the beginning, primarily as it made it easier for appropriate calculations when going deeper in the final CNN architecture). In order to calculate the original labels, which were six coefficients (three for each lane line, with each of the three being a coefficient for a polynomial-fit laneline), we also had to do a few basic computer vision techniques first. We had to perform image calibration with OpenCV to correct for our camera'sinherent distortion, and then use perspective transformation to put the road lines on a flat plane.

Initially, we wanted to make the model more robust to our original model by drawing over the lane lines in the image, which can be blurry or fade awayinto the rest of the image the further to the back of the image it is. We drewover 1,420 perspective transformed road images in red, and ran a binary color threshold for red whereby the output image would show white wherever therehad been sufficient red values and no activation (black) where the red values were too low. With this, we re-ran our original model, modified to output onlythe six coefficients instead of the lane drawing, so that we could train the network based on those coefficients as labels.

However, we soon found that the amount of data we had was not sufficient, especially because most of the images were of fairly straight lanes. So, wewent back and ran the original CV model over all our videos from roads that were very curvy. We also added in a limited amount of images from the regular project video (we wanted to save the challenge video for a true testafter finalizing our model) from Udacity's SDC Nanodegree Advanced Lane Lines project we previously completed, so that the model could learn some ofthe distortion from a different camera. However, this introduced a complicationto our dataset – Udacity's video needed a different perspective transformation, which has a massive effect on the re-drawn lane.

We ended up obtaining a mix of both straight lines and various curved lines, as well as various conditions (night vs. day, shadows, rain vs. sunshine) in order to help with the model's overall generalization. These will help to cover more of the real conditions that drivers see every day. We will discuss more of the image statistics later regarding total training images used, but have provided two charts below regarding the percentage breakouts of road conditions for those obtained from our own driving video collected.

We noted before that one issue with the original videos collected was that too much of the data came from straight lanes, which is not apparent from the above charts – although “Very Curvy” made up 43% of the original dataset, which We initially believed would be sufficient, we soon found out the breakout was terribly centered around straight, as can be seen in the below chart from one of the coefficient labels’ distributions. This is a definite problem to be solved.



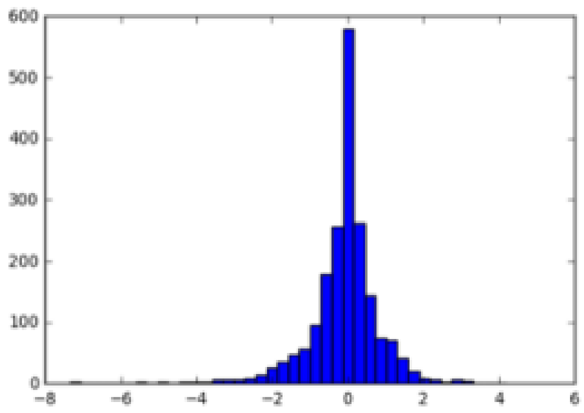
1.5.2 Data Preprocessing

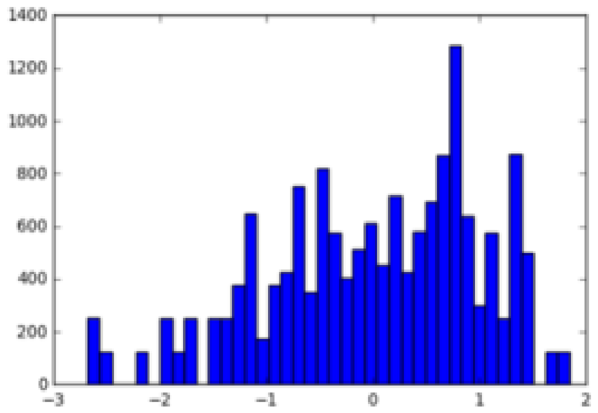
Some of the general techniques we used to preprocess our data to create labelsare discussed above in the "Algorithms and Techniques" section. However, there was a lot more to making sure our model got sufficient quality data for training. First, after loading all the images from each frame of video we took,an immediate problem popped up. Where we had purposefully gathered night video and rainy video, both of these severely cut down on the quality of images. In the night video, where our smartphone camera already was of slightly lesser quality, we was also driving on the highway, meaning a much bumpier video, leading to blurry images. We sorted through each and every single gathered image to check for quality, and ended up removing roughlyone- third of our initial image data from further usage.

From here, we also wondered whether the model might overfit itself by getting a sneak peek at its own validation data if images were too similar to each other – at 30 frames per second, there is not a whole lot of change from one frame to the next. We decided to only use one out of every ten images for training. With these, We drew over the images in red as mentioned above, and then ran our programs for making labels and checking the labeled images.

Here, we found that the process for making the labels was flawed – the original code for our sliding windows failed completely on curves. This was because the initial code, when it hit the side of an image, would keep searching straight up the image – causing the polynomial line to think it should go up the image too. By fixing our code to end when it hit the side of the image, we vastly improved its usefulness on curves. After re-checking our labels and tossing out some bad ones,we moved on to check the distribution of our labels. It was bad – there were still hardly any big curves! Even the videos from curvy roads still had mostly straight lines. So, we re-ran our process over one in every five images, only from the four videos with mostly curved lines.We also decided at this point to add a little bit of the Udacity basic project video from our CV- based project (not from the Challenge video though because we wanted that as the true test of the model's ability to generalize) in order to train the model for different distortion (which we had already obtained previously). However, this caused a new issue – these images needed a different perspective as well. This was not so hard for creating the labels, but we knew it could cause an issue on the tail end, as the labels would require the specific inverse perspective transformation to redraw the lines.

As shown previously in the "Datasets and Inputs" section, after adding in this additional data, our distribution was still fairly unequal. However, we found by using the histograms of the distributions of each label, we could find where the exact values were where only a limited amount of images fell. By iteratingthrough each of the labels, and finding which training images were on the fringes of the data, we could then come back and generate "new" data; thisdata was just rotations of these images outside the main distribution, but ourCNN would likely become much more likely to not overfit to straight lines.





old distribution *new distribution*

Coefficient Labels

The changes in the distribution of image labels for the second coefficients are shown above. We originally normalized the labels with sklearn's "StandardScaler" (causing the differences in values above), which improved training results but also did need to be reversed after training to return the correct label.At this point, the approach depending on the model diverges. Inour initial models, We took in either a perspective transformed image or regular road image, downscaled it from 720x1280x3 to 45x80x3 (scaling down 16X), gray-scaled the image, added back a third dimension (cv2.cvtColor removes the dimension when gray-scaling but Keras wants it to be able to run properly), and then normalized the image (new_image = (new_image / 255) *
.8 - 1) to be closer to a mean of zero and standard deviation of one, whichis key in machine learning (the algorithms tend to converge better).

On the flip side, once we changed to a fully convolutional model, we instead was only down-sizing the road images to 80x160x3 (a slightly different aspect ratio than the original but roughly 8X scaled down), without any further gray- scaling or normalization (we instead on our Batch Normalization layer in Kerasto help there). Additionally, since a fully convolutional model essentially returns another image as output, instead of saving down our lane labels as numbersonly, we also saved down the generated lane drawings prior to merging themwith the road image. These new lane image labels were to be the true labelsof our data. We still used our image rotations to generate additional data, but also rotated the lane image labels along with the training image (based on the distributions of the original label coefficients still). We also added in a horizontal flip of each image and corresponding lane image label to double our dataset for training. For these new lane image labels, We dropped off the 'R' and 'B' color channels as the lane was being drawn in green, hoping to make training more efficient with less of an output to generate.

1.5.3 Image Statistics

- Here are some statistics from our data pre-processing:
- 21,054 total images gathered from 12 videos (a mix of different times of day, weather, traffic, and road curvatures – see previous pie chart breakout)
- The roads also contain difficult areas such as construction and intersections
- 14,235 of the total that were usable of those gathered (due to blurriness, hidden lines, etc.)
- 1,420 total images originally extracted from those to account for time series (1 in 10)
- 227 of the 1,420 unusable due to the limits of the CV-based model used to label (down from 446 due to various improvements made to the original model) for a total of 1,193images

- Another 568 images (of 1,636 pulled in) gathered from more curvy lines to assist in gaining a wider distribution of labels(1 in every 5 from the more curved-lane videos; from 8,187 frames)
- In total, 1,761 original images
- We pulled in the easier project video from Udacity's Advanced Lane Lines project (to help the model learn an additional camera's distortion) - of 1,252 frames, we used 1 in 5 for 250 total, 217 of which were usable for training
- A total of 1,978 actual images used between our collections and theone Udacity video
- · After checking histograms for each coefficient of each label for distribution, we created an additional 4,404 images using small rotations of the images outside the very center of the original distribution of images. This was done in three rounds of slowly moving outward from the center of the data (so those further out from the center of the distribution were rotated multiple times). 6,382 images existed at this point.
- · Finally, we added horizontal flips of each and every road image and its corresponding label, which doubled the total images. All in all, there were a total of 12,764 images for training.

Chapter-2 LITERATURE SURVEY

2.1 Summary of Papers

2.1.1

Tittle	**A Study on Real-Time Detection Method of Lane and Vehicle for Lane Change Assistant System Using Vision System on Highway**
Authors	HeungsukKim[a] SeoChangJun[b] KwangsuckBoo[a]
Year of Publication	February 2016
Web Link	https://www.sciencedirect.com/science/article/pii/S2215098617317317

2.1.2

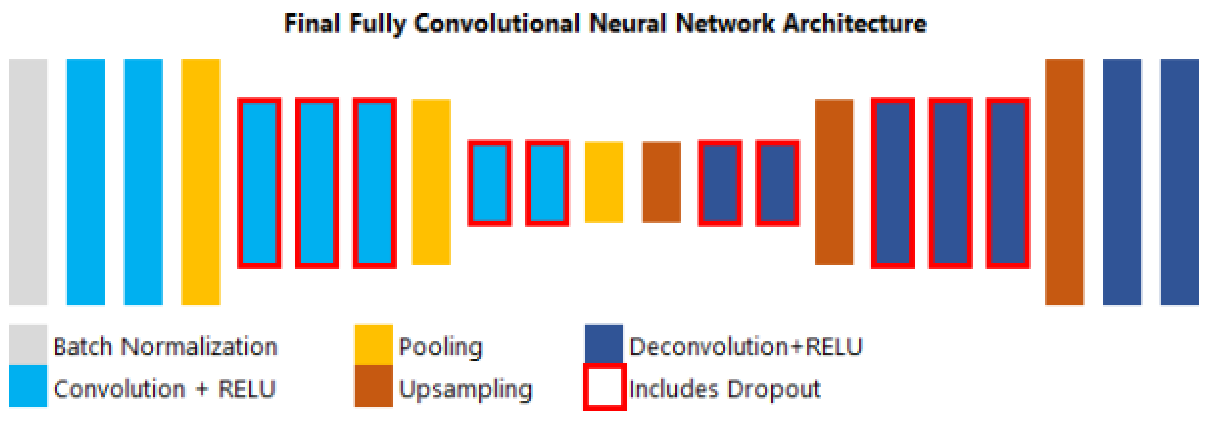
Tittle	**A practical method of road detection for intelligent vehicle**
Authors	G. Dezhi, L. Wei D. Jianmin Z. Banggui
Year of Publication	Aug. 2009
Web Link	https://ieeexplore.ieee.org/abstract/document/5262562

2.1.3

Tittle	Lane detection for driver assistance and intelligent vehicle applications. 2007 international symposium on communications and information technologies. 1291-1296
Authors	Craig D'Cruz, Ju Jia Zou
Year of Publication	March 2016
Web Link	https://ieeexplore.ieee.org/document/4392216

CHAPTER-3 SYSTEM DEVELOPMENT

3.1 Algorithms and techniques



First, we must extract the image frames from videos we obtain. Next, we must process the images for use in making labels, for which we will use the same techniques we previously used in our computer vision-based model. Next, we will calibrate for our camera's distortion by taking pictures of chessboard images with the same camera the video was obtained with, and undistort the image using "cv2.findChessboardCorners" and "cv2.calibrateCamera". With the image now undistorted, we will find good source points (corners of the lane lines at the bottom and near the horizon line) and determine good destination points (where the image gets transformed out to) to perspective transform the image (mostly by making educated guesses at what would work best. Whenwe have these points, we can use "cv2.getPerspectiveTransform" to get a transformation matrix and "cv2.warpPerspective" to warp the image to a bird's eye- like view of the road.

From here, in order to enhance the model's robustness in areas with less thanclear lane lines, we drew red lines over the lane lines in each image used. In these images, we will use a binary thresholding on areas of the image with high red values so thatthe returned image only has values where the red lane line was drawn. Then, histograms will be computed using where the highest amount of pixels fallvertically (since the image has been perspective transformed, straight lane lines will appear essentially perfectly vertical) that split out from the middle of the image so that the program will look for a high point on the left and a separate high point on the right. Sliding windows that search for more binary activation going up the image will then be used to attempt to follow the line.

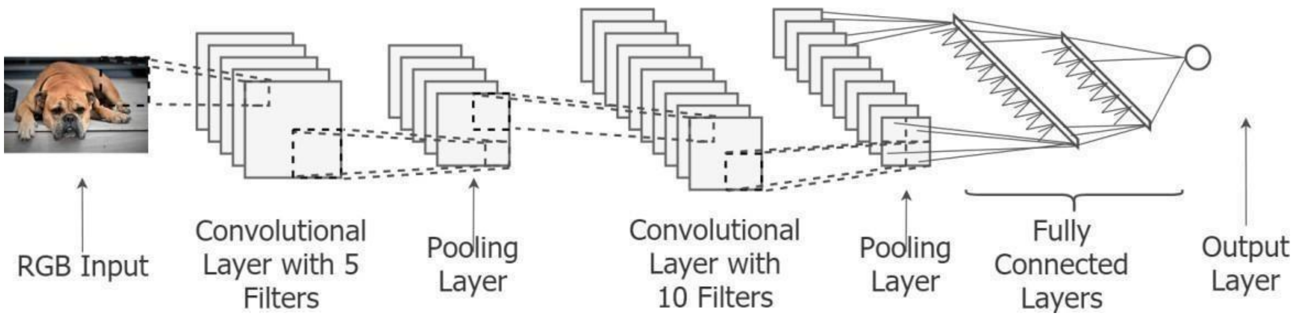
Based on the detected pixels from the sliding windows, "numpy.polyfit" will be used to return polynomial functions that are most closely fit to the laneline as possible (using a polynomial allows for it to track curved lines as well as straight). This function actually returns the three coefficients of the "ax^2+bx+c" equation, where "a", "b" and "c" are the coefficients. We will append the total of six coefficients (three for each of the two lane lines) toa list to use as labels for training.

However, prior to training, we will want to check whether the labels are even accurate at all. Using the labels from above, we can feed an image throughthe original distortion and perspective transformation, create an image "blank" with "numpy.zeros_like", make lane points from the polynomial coefficients by calculating the full polynomial fit equation from above for each line, and thenuse "cv2.fillPoly" with those to create a lane drawing. Using "cv2.warpPerspective" with the inverse of our perspective transformation matrix calculated before, we can revert this lane drawing back to the space of the original image, and then use "cv2.addWeighted" to merge the lane drawing with the original image. This way, we can make sure We feed accurate labelsto the model.

Lastly, our project will use Keras with TensorFlow backend in order to create a convolutional neural network. Using "keras.models.Sequential", we can create the neural network with convolutional layers (keras.layers.Convolution2D) and fully- connected layers (keras.layers.Dense). We will first try a model architecture similar to the one at left, which we used successfully in a previous project for Behavioral Cloning.

3.2 Mathematical Model

Convolution neural networks (CNNs) are a family of deep networks that can exploit the spatial structure of data (e.g. images) to learn about the data, so that the algorithm can output something useful. For example, if We give the the CNN an image of a person, this deep neural network first needs to learn some local features (e.g. eyes, nose, mouth, etc.). These local features are learnt in ***convolution layers***. Then the CNN will look at what local features are present in a given image and then produce specific activation patterns (or an activation vector) which globally represents the existence of those local features maps. These activation patterns are produced by ***fully connected*** layers in the CNN. For example, if the image is a non-person, the activation pattern will be different from what it gives for an image of a person. There are three different components in a typical CNN. They are, convolution layers, pooling layers and fully-connected layers. A convolution layer consists of many ***kernels***. These kernels (sometimes called ***convolution filters***) present in the convolution layer, learn local features present in an image (e.g. how the eye of a person looks like). Such a local feature that a convolution layer learns is called a ***feature map***. Then these features are convolved over the image. This convolution operation will result in a matrix (that is sometimes called an ***activation map***). The activation map produces a high value at a given location,if the feature represented in the convolution filter is present at that location of theinput .The pooling layer make these features learnt by the CNN translation invariant (e.g. no matter the person's eye is at [*x=10, y=10*] or [*x=12,y=11*] positions, the output of the pooling layer will be same).Fully connected layers are responsible for producing different activation patterns based on the set of activated feature maps andthe locations in the image, the feature maps are activated for. This is what CNNlooks like visually


RGB Input
Convolutional Layer with 5 Filters
Pooling Layer
Convolutional Layer with 10 Filters
Pooling Layer
Fully Connected Layers
Output Layer

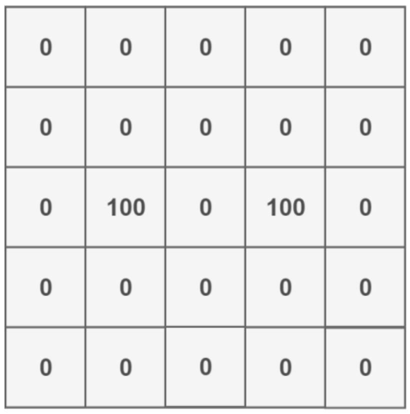
Network diagram

3.2.1 Convolutional Layer

Convolution operation outputs a high value for a given position if the convolution feature is present in that location, else outputs a low value. More concretely, at agiven position of the convolution kernel, we take the element-wise multiplication of each kernel cell value and the corresponding image pixel value that overlaps the kernel cell, and then take the sum of that. The exact value is decided according tothe following formula (*m* — kernel width and height, *h* — convolution output, *x* — input, *w* — convolution kernel).

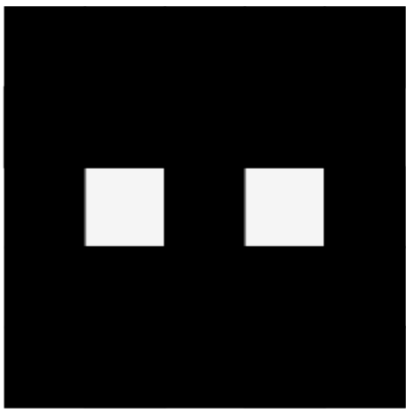
$$\mathrm{h}_{i,j} = \sum_{k=1}^{m} \sum_{l=1}^{m} w_{k,l} x_{i+k-1,j+l-1}$$

It is not enough to know what the convolution operation does, we also need to understand what the convolution output represents. Just imagine colours for the values in the convolution output (0 — black, 100 — white). If visualized ,this image will represent a binary image that lights up at the location the eyes are at.



0	0	0	0	0
0	0	0	0	0
0	100	0	100	0
0	0	0	0	0
0	0	0	0	0

Convolution Output



Convolution Output
(as an Image)

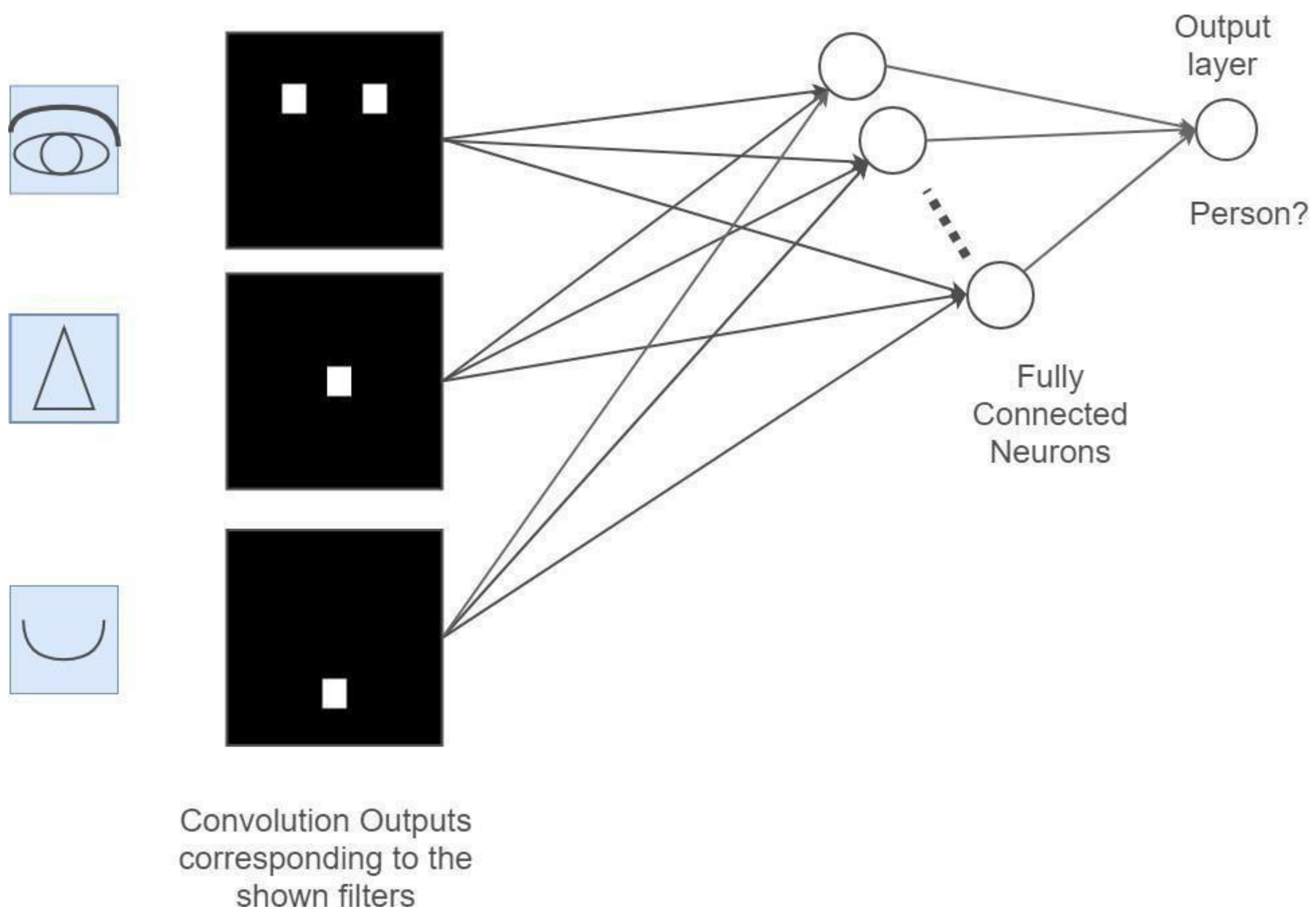
3.2.2 Pooling Layer:

Pooling (or sometimes called subsampling) layer make the CNN a little bit translation invariant in terms of the convolution output. There are two different pooling mechanisms used in practice (max-pooling and average-pooling). More precisely, the pooling operation, at a given position, outputs the maximum value of the input, that falls within the kernel. So mathematically,

$$\mathrm{h}_{i,j} = max\{x_{i+k-1,j+l-1} \forall 1 \leq k \leq m \text{ and } 1 \leq l \leq m\}$$

3.2.3 Fully Connected Layers:

Fully connected layers will combine features learnt by different convolution kernels so that the network can build a global representation about the holistic image. The neurons in the fully connected layer will get activated based on whether various entities represented by convolution features is actually present in the inputs. As the fully connected neurons get activated for this, it will produced different activation patterns based on what features are present in the input images. This provides a compact representation of what exists in the image, to the output layer, that the output layer can easily use to correctly classify the image.



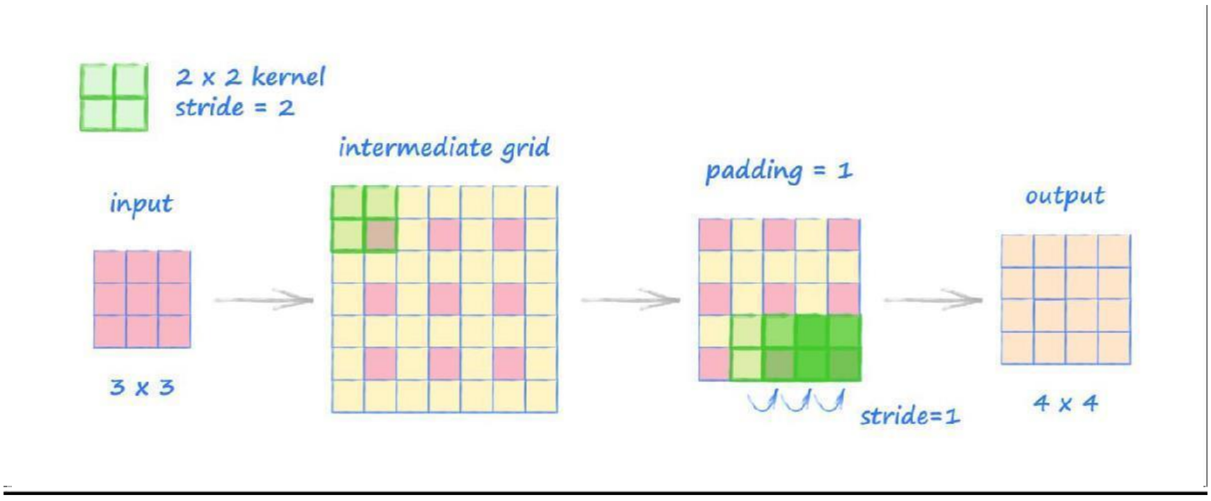
3.2.4 Deconvolutional Layer

Deconvolutional networks are convolutional neural organizations (CNN) that work in a switched cycle. Deconvolutional networks, otherwise called deconvolutional neural organizations, are fundamentally the same as in nature to CNNs run backward however are a particular use of man-made consciousness (AI).

Deconvolutional networks endeavor to discover lost highlights or signals that may have beforehand not been considered essential to a convolutional neural organization's errand. A sign might be lost due to having been tangled with different signs. The deconvolution of signs can be utilized in both picture combination and investigation.

A convolutional neural organization imitates the activities of a natural cerebrum's frontal projection work in picture preparing. A deconvolutional neural organization develops upwards from prepared information. This retrogressive capacity can be viewed as a figuring out of tangled neural organizations, developing layers caught as a feature of the whole picture from the machine vision field of view and isolating what has been tangled.

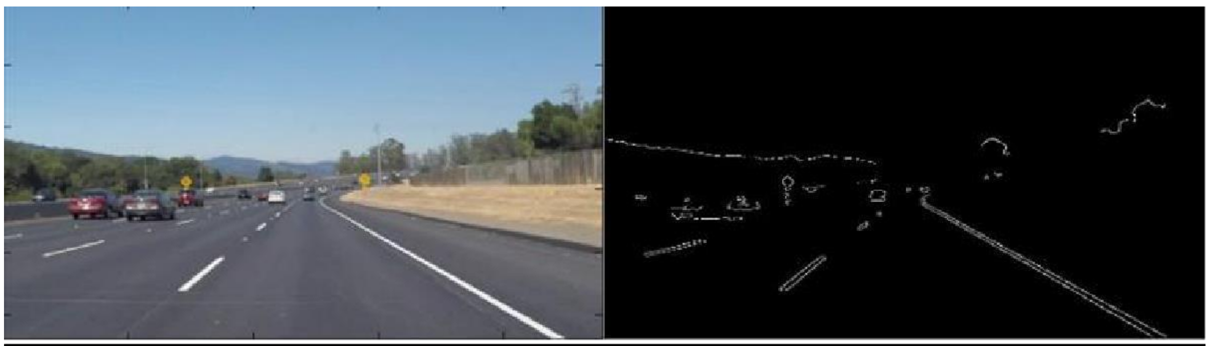
Deconvolutional networks are identified with other profound learning techniques utilized for the extraction of highlights from progressive information, for example, that found in profound conviction organizations and chain of importance meager programmed encoders. Deconvolutional networks are essentially utilized in logical and designing fields of study.



Deconconvolutional Layer

3.2.5 **Weaving Them Together**

Now all we have to do is put all these together, to form an end-to-end model, from raw images, to decisions. And once connected the CNN will look like this. To summarize, the convolution layers will learn various local features in the data (e.g. what an eye looks like), then the pooling layer will make the CNN invariant to translations of these features (e.g. *if the eye appear slightly translated in two images, the CNN will still recognize it as an eye*).Finally we have fully connected layers, that says, “we found two eyes, a nose and a mouth, so this must be a person, and activate the correct output.

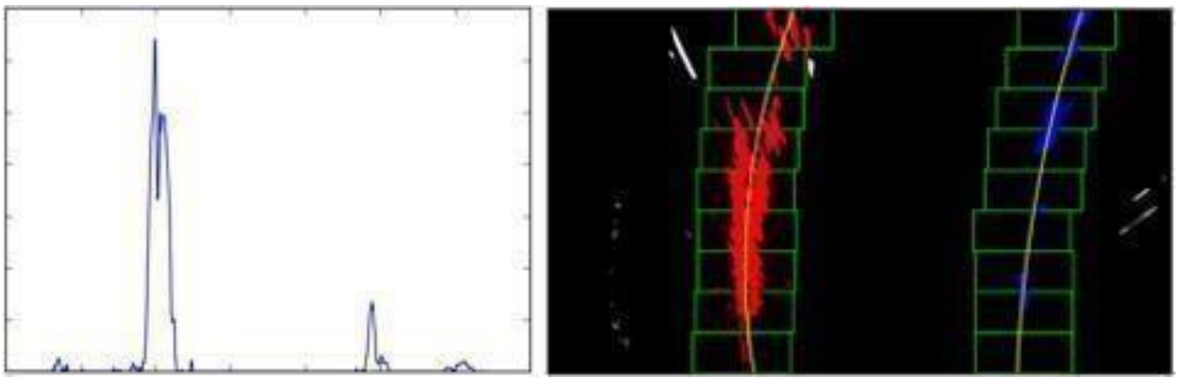


Canny Edge Detection

3.3 Implementation

Our first CNN build used perspective transformed images as input, and can be seen in the “perspect_NN.py” file. It used batch normalization, four convolutional layers with a shrinking number of filters, followed by a pooling layer, flatten layer, and four fully- connected layers, with the final fully-connected layer having six outputs – the six coefficient labels of the lane lines.

Each layer used RELU activation, or rectified linear units, as this activationhas been found to be faster and more effective than other activations. Wetried some of the other activations as well just in case, but found RELU tobe the most effective, as expected. Also, in order to help prevent overfitting and increase robustness, we added in strategic dropout to layers with the most connections, and also used Keras’s ImageDataGenerator to add in image augmentation like more rotations, vertical flips, and horizontal shifts. We originally used mean-squared-error for loss, but found that mean-absolute error actually produced a model that could handle more variety in curves. Note that We also made the training data and labels into arrays before feeding the model as it works with Keras. Also, We shuffled the data to make sure thatthe different videos were better represented and the model would not justoverfit on certain videos. Last up was splitting into training and validation sets so we could check how the model was performing.

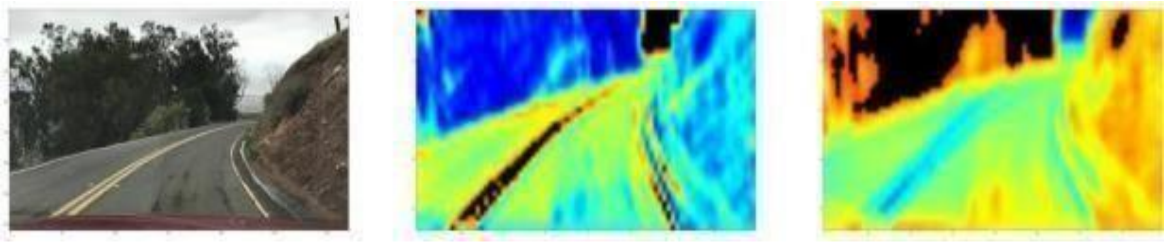

Perspective Image’s Histogram and Sliding Windows

After training this first model and creating a function to actually see the re- drawn lanes, we found this first model to be moderately effective, given that you were using the same perspective transformation from the original model. However, our end goal was to ignore the need to perspective transform an image for the CNN altogether, so after finding that the first model was moderately effective at producing a re-drawn lane, we shifted course. In “road_NN.py”, this second model is included. Other than feeding in a regular road image, the only change we made to this model was adding a Crop layer, whereby the topthird of the image was removed (we played around with one half or one third without much difference). We found quickly that the CNN could, in fact, learn the lane coefficients without perspective transformation, and the resultingmodel was actually a little bit more effective even.There was still one big problem – if our model was predicting the lane label coefficients, this meantthat the lines still need to be drawn in a perspective-transformed space, and reverted to the road space, even if the original image was not perspective- transformed. This caused massive issues in generalizing to new data – we would need the transformation matrix of any new data (even slight changes in camera mounting would cause a need for a new matrix).

3.3.1 Refinement

Our first thought was whether or not we could actually look directly at theactivation of the convolutional layers to see what the layer was looking at. We assumed that if CNN was able to determine the appropriate line coefficients, it was probably activating over the actual lines of lane, or atleast some similar area in the image that would teach it the values to predict.

After some research, we found the keras-vis library to be great for looking at the activation of each layer. This library can actually look atthe class activation maps (in our case the “classes” are actually each of the coefficient labels since this is not a classification problem) in each layer. We thought We had found our solution, until we looked at the activation mapsthemselves.



While the above activation maps of the first few layers look okay, these were actually some of the clearest we could find. Interestingly enough, CNN actually often learned by looking at *only one lane line* – it was calculating the position of the other line based on the one it looked at. But that was only the case for curves – for straight lines, it was not activating on the lane lines at all! It was actually activating directly on the road in front of the car itself, and deactivating over the lane lines. As a result, we realized the model was activating in different ways for different situations, which would make using the activation maps directly almost impossible. Also, notice in the above second image that the non-cropped part of the sky is also being activated (the dark portion) – due to the various rotations and flips, the model was also activatingin areas that were telling it top from bottom. Other activation maps also activated over the car at the bottom of the image for the same purpose.

We also briefly tinkered with trying to improve the activation maps above by using transfer learning. Given that in our Behavioral Cloning project, the car needed to stay on the road, we figured it had potentially learned a similar, but perhaps more effective, activation. Also, we had tens of thousands of images to train on for that project, so the model was already more robust. After using "model.pop" on that model to remove the final fully-connected layer (which had only one output for that project), we added a new fully-connected layer withsix outputs. Then, we trained the already-established model further on our realroad images (the old model was trained on simulated images), and actuallyfound that it did a better job on looking at both lines, but still failed to havea consistent activation we could potentially use to redraw lines more accurately.

At this point, we began to consider what we had read on image segmentation, especially SegNet ,which was specifically designed to separate different components of a road out in an output image by using a fully convolutional neural network. This approach was different from mine in thata *fully* convolutional neural network does not have any fully-connected layers (with many more connections between them), but only uses convolutional layers followed by deconvolutional layers to essentially make a whole new image. We realized we could skip the undoing of the perspective transformation for the lane label entirely but actually train directly to the lane drawing as the output. By doing so, it meant that even if the camera was mounted differently, the lanes were spaced differently, etc., the model would still be able to return an accurate drawing of the predicted lane.

3.4 Experimental Development

3.4.1 TENSORFLOW

TensorFlow is an open source software library for high performance numerical computation. Its flexible architecture allows easy deployment of computation across a variety of platforms (CPUs, GPUs, TPUs), and from desktops to clusters of servers to mobile and edge devices. Originally developed by researchers and engineers from the Google Brain team within Google’s AI organization, it comes with strong support for machine learning and deep learning and the flexible numerical computation core is used across many other scientific domains.

3.4.2 KERAS

Keras is a high-level neural network API, which is written in Python language and capable of running on top of TensorFlow, Theano or CNTK. It was developed with focus on the enabling fast experimentation. Being able to go from idea to resultwith the least possible delay is key to doing good research. Keras was developedand is maintained by Francois Chollet and is part of the Tensorflow core, which makes it Tensorflow preferred high-level API. Use Keras if you need a deep learning library that:

➢ Allows for fast and easy prototyping(through user friendliness, modularity, extensibility).

➢ Recurrent and Convolution networks both are supported .

➢ Can run on both CPU and GPU

3.4.3 PYTHON

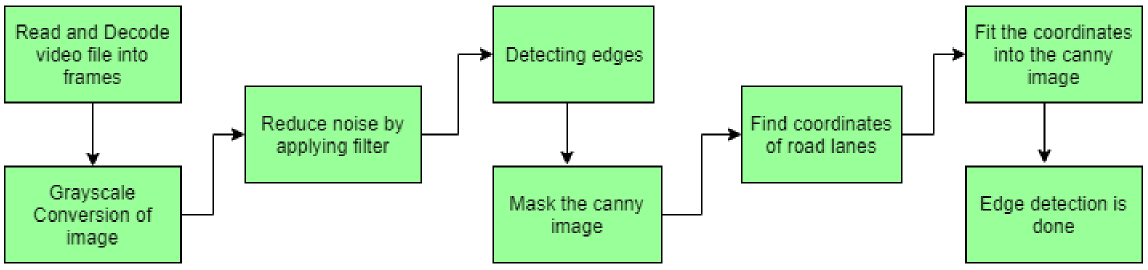
Python is an interpreted, high-level, general-purpose programming language. Created by Guido van Rossum and first released in 1991, Python's design philosophy emphasizes code readability with its notable use of significant whitespace. Its language constructs and object-oriented approach aim to help programmers write clear, logical code for small and large-scale projects.

Python is dynamically typed and garbage-collected. It supports multiple programming paradigms, including procedural, object-oriented, and functional programming. Pythonis often described as a "batteries included" language due to its comprehensive standard library.

3.5 Proposed approach

The first step in our project will be data collection. As discussed above, We plan to use our smartphone in a variety of different road settings to gain a good mix of video. Once We have collected enough video, We can then process the video to split out each individual video frame. These frames are the images We will use to train (and validate) our CNN eventually. We plan to scale down the images (starting at 25% of the original size) to reduce processing time, although if We am successful at training a good model with our approach We may attempt a full-scale run.

The second step will be to create labels for the images. There are a thoughts We have here. We plan to calibrate our camera, undistort the image and perspective transform first. We could then draw over the lines in a very distinctive color, and use some basic computer vision techniques (note that this is on training images only - the point is that after training, the CNN will no longer need any computer vision-based components) in order to make a list of pixels making up the lines. This will hopefully be slightly more accurate than relying on various different thresholds in the image as We can make it specific to that single color channel/hue. Once these pixels are detected, We can use numpy.polyfit() to get back the line's coefficients.


Read and Decode video file into frames
Grayscale Conversion of image
Reduce noise by applying filter
Detecting edges
Mask the canny image
Find coordinates of road lanes
Fit the coordinates into the canny image
Edge detection is done

Flowchart Of Lane Detection

This is a very small portion of data to check against, but will at least allow me tofind out very early on if our labelling technique will work as intended, or if Weneed to take a different approach.

Once We have the data labelled, We will build our convolutional neural network.We plan to begin with a CNN structure similar to that We used in the SDC nanodegree Behavioral Cloning project here, which uses 5 convolutional layers and 4 fully connected layers (along with pooling and dropout within), built with Keras ontop of Tensorflow. There are a few changes We already know We need to make, although there will definitely be a decent amount of iteration on the final CNN We use (which luckily Keras makes substantially easier). First, as already discussed, We will need six outputs at the end for our three coefficients for two separate lanelanes, as opposed to just the single output We had for the steering angle in the Behavioral Cloning project. Second, the images from that project were only 360 x 120, so We may need more convolutional layers, larger strides/subsamples, or larger pooling in order to cut down on the number of parameters in the model. Even with

5 convolutional layers, because We used small kernel sizes (the grouping of pixelsthe CNN considers as a group for a node at each layer), small strides/subsamples, and small pooling, our old CNN model still had nearly 600,000 parameters at thefirst fully connected layer, which would be vastly increased if the image size is alot larger.At this point, after having finished the labelling on a subset of the data,We will test to see whether the CNN actually appears to be learning anything. Although neural networks definitely tend to work better the larger the data size is,if our CNN cannot show at least a little bit of convergence on a subset of thedata, We will need to modify our procedures up to this point. Note that this meansWe will have a training set and a validation set created out of this subset of datausing sklearn's train_test_split. If it is working, great, We will move on to using thefull dataset.

If the model does appear to be minimizing the error (mean-squared error is ourplan), then We will move on to the next step, which is testing the model on completely new data it has never seen before. We will first use the regular project video from the Advanced Lane Lines project and see how the CNN's predictions compare to the computer vision-based model We used before. We will compareboth its performance from a visual standpoint (how close it tracks to the lines versus the CV-based model) and from a speed standpoint (how long does it take to process a video vs. the CV-based model). If all goes well, We will also try the CNN's predictions on the challenge video and hard challenge video from the Advanced Lane Lines project, and hope that the end result is a model that can predict faster, and better, than the computer vision-based model, and even become nearly imperceptibly different than a human being would determine it to be.

CHAPTER-4 PERFORMANCE ANALYSIS

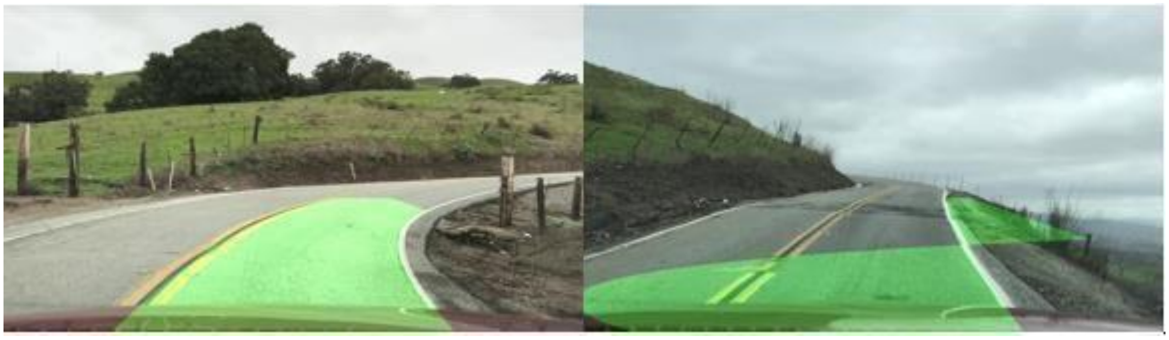
4.1 Evaluation and Validation

After 20 epochs, our model finished with MSE for training of 0.0046 and validation of 0.0048, which was significantly lower than any previous model's we had tried (although a bit of apples and oranges against the models usingsix polynomial coefficients as labels). We first tried the trained model against one of our own videos, one of the hilly and curved roads for which the modelhad potentially seen up to 20% of the images for, although likely much less – from the image statistics earlier, we had to throw out a large portion of the images from these videos, so even though we ran it on one in five images, the model probably only saw 5-10% of them. Fascinatingly, the model performed great across the entire image, only losing the right side of the lane at one point when the line became completely obscured by leaves. The model actually performed near perfectly even on a lot of the areas we knew we had previously had to throw out, because our CV- based model could not appropriately make labels for them. Note that because there is not a "ground- truth" for our data, we cannot directly compare to that model from a loss/accuracy perspective, but as the end result is very visual, we will see which model produces the better result. Part of this comes down to robustness
– our pure CV model failed to produce lane lines past the first few secondsof a Challenge video in our previous project. If this model can mostly succeed on the Challenge video (i.e. no more than a few seconds without the lane

shown) without having been specifically trained on images from that video, it will have exceeded this benchmark. A second benchmark will be the speed of the model – the CV-based model canonly generate roughly 4.5 frames per second, which compared to 30 fps video incoming is much slower than real-time. The model will exceed this benchmark if the writing of the video exceeds 4.5 fps.

4.2 Justification

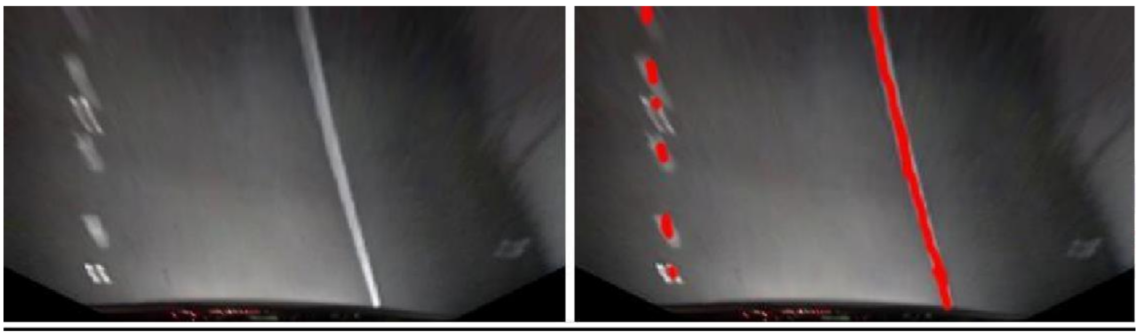
However, the fact remained that the model had in fact seen some of those images. What about trying it on the challenge video created by Udacity forthe Advanced Lane Lines project? It had never been trained on a single frameof that video. Outside of a small hiccup going under the overpass in the video, the model performed great, with a little bit of noise on the right side where the separated lane lines were. It had passed our first benchmark –outperforming our CV- based model, which had failed on this video.



Good Prediction vs. Poor Prediction

Our second benchmark was with regards to speed, and especially when including GPU acceleration, the deep learning model crushed the earlier model
– it generated lane line videos at between 25-29 fps, far greater than the 4.5fps for the CV model. Even without GPU acceleration, it still averaged 5.5fps, still beating out the CV model. Clearly, GPU acceleration is key in unlocking the potential of this model, running almost real-time with 30 fps video. With regards to both robustness and speed, the deep learning-based model is a definite improvement on the usual CV-based techniques.

An example of bad image. The model performed quiet good in it



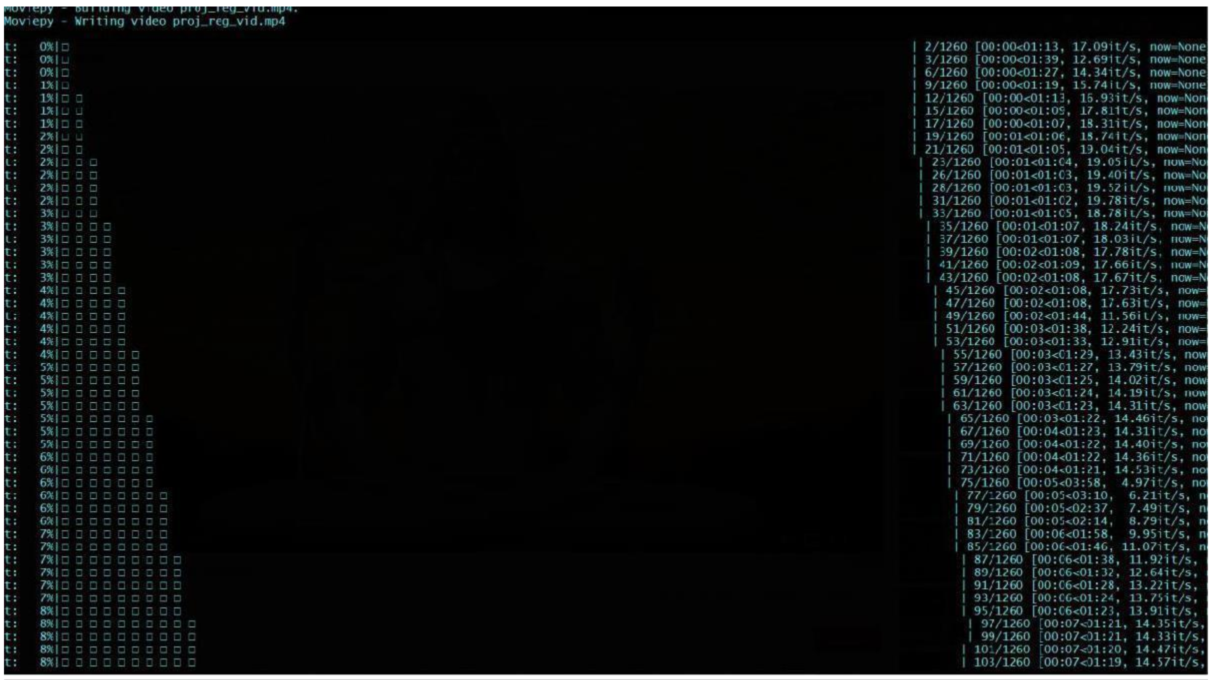
Lanes Detected in above image

CHAPTER-5 RESULTS

5.1 The Final Model

Although we had made a CNN previously that ended in fully-connected layers, we had never before made a fully convolutional neural network, and there were some challenges in getting the underlying math to work for our layers. Unlike in the forward pass in normal Convolution layers, Keras's Deconvolution layers flip around the backpropagation of the neural network to face the opposite way, and therefore need to be more carefully curated to arrive at the correct size (includingthe need to specify the output size). We chose to make our new model a mirror of itself, with Convolutional layers and Pooling slowly decreasing in size layers, with the midpoint switching to Upsampling (reverse-pooling) and Deconvolution layers of the same dimensions. The final deconvolution layer ends with one filter, which is because we only wanted a returned image in the 'G' color channel, aswe were drawing our predicted lanes in green (it later is stacked up with zeroed-out 'R' and 'B' channels to merge with the original road image). Choosing toinput 80x160x3 images (smaller images were substantially less accurate in their output, likely due to the model being unable to identify the lane off in the distance very well) without gray- scaling (which tended to hide yellow lines onlight pavement), We also normalized the incoming labels by just dividing by 255 (such that the labels were from 0 to 1 for 'G' pixel values).

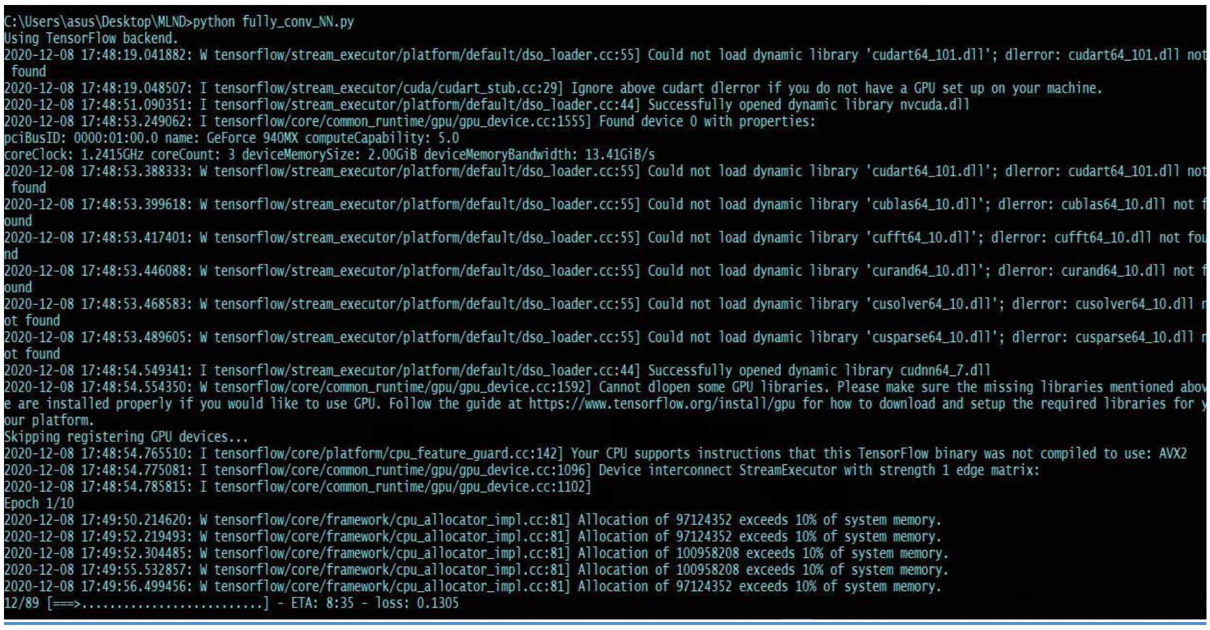
The final model is within the "fully_conv_NN.py" file. We stuck with RELU activation and some of the other convolution parameters (strides of (1,1) and 'valid' padding had performed the best) from our prior models, but also addedmore extensive dropout. We had wanted to use dropout on every Convolutional and Deconvolutional layer, but found it used up more memory than we had. Wealso tried to use Batch Normalization prior to each layer but found it also usedup too much memory, and instead We settled for just using it at the beginning. A more interesting discovery, given that using MSE for loss had previously failed, was that it performed much better than any other loss function with this new model. Also intriguing was that adding *any* type of image augmentation with ImageDataGenerator, whether it be rotations, flips, channel shifts, shifts along either the horizontal or vertical axes, etc., did not result in a more robust model, and often had worse results on any test images we looked at. Typically, we expect the image augmentation to improve the final model, but in this case, skipping any augmentation (although we kept the generator in any way without it,as it is good practice) led to a better model. Channel shifts helped with shadows, but worsened overall performance. This is fed into the "draw_detected_lanes.py" file, in which the model predicts the lane, is averaged over five frames (to account forany odd predictions), and then merges with the original road image from a video frame. Our CNN will be being trained similarly to a regression-type problem, in which it will be given the polynomial coefficients on the training images, and attempt to learn how to extract those values from the given road images. The loss it will minimize is the difference between those actual coefficients and what it predicts (likely using mean- squared error). It will then use only limited computer vision techniques in order to draw the lane back onto the image. As noted above regarding benchmarking against our computer vision-based result, We will also evaluate it directly against that model in both accuracy and speed, as well as on even more challenging videos than our CV- based model was capable of doing.



```
Moviepy - Building video proj_reg_vid.mp4.
Moviepy - Writing video proj_reg_vid.mp4

t:   0%|□                         | 2/1260 [00:00<01:13, 17.09it/s, now=None
t:   0%|□                         | 3/1260 [00:00<01:39, 12.69it/s, now=None
t:   0%|□                         | 6/1260 [00:00<01:27, 14.34it/s, now=None
t:   1%|□                         | 9/1260 [00:00<01:19, 15.74it/s, now=None
t:   1%|□ □                       | 12/1260 [00:00<01:13, 16.93it/s, now=Non
t:   1%|□ □                       | 15/1260 [00:00<01:09, 17.81it/s, now=Non
t:   1%|□ □                       | 17/1260 [00:00<01:07, 18.31it/s, now=Non
t:   2%|□ □                       | 19/1260 [00:01<01:06, 18.74it/s, now=Non
t:   2%|□ □                       | 21/1260 [00:01<01:05, 19.04it/s, now=Non
t:   2%|□ □ □                      | 23/1260 [00:01<01:04, 19.05it/s, now=No
t:   2%|□ □ □                      | 26/1260 [00:01<01:03, 19.40it/s, now=No
t:   2%|□ □ □                      | 28/1260 [00:01<01:03, 19.52it/s, now=No
t:   2%|□ □ □                      | 31/1260 [00:01<01:02, 19.78it/s, now=No
t:   3%|□ □ □                      | 33/1260 [00:01<01:05, 18.78it/s, now=No
t:   3%|□ □ □ □                     | 35/1260 [00:01<01:07, 18.24it/s, now=N
t:   3%|□ □ □ □                     | 37/1260 [00:01<01:07, 18.03it/s, now=N
t:   3%|□ □ □ □                     | 39/1260 [00:02<01:08, 17.78it/s, now=N
t:   3%|□ □ □ □                     | 41/1260 [00:02<01:09, 17.66it/s, now=N
t:   3%|□ □ □ □                     | 43/1260 [00:02<01:08, 17.67it/s, now=N
t:   4%|□ □ □ □ □                    | 45/1260 [00:02<01:08, 17.73it/s, now=
t:   4%|□ □ □ □ □                    | 47/1260 [00:02<01:08, 17.63it/s, now=
t:   4%|□ □ □ □ □                    | 49/1260 [00:02<01:44, 11.56it/s, now=
t:   4%|□ □ □ □ □                    | 51/1260 [00:03<01:38, 12.24it/s, now=
t:   4%|□ □ □ □ □                    | 53/1260 [00:03<01:33, 12.91it/s, now=
t:   4%|□ □ □ □ □ □                   | 55/1260 [00:03<01:29, 13.43it/s, now
t:   5%|□ □ □ □ □ □                   | 57/1260 [00:03<01:27, 13.79it/s, now
t:   5%|□ □ □ □ □ □                   | 59/1260 [00:03<01:25, 14.02it/s, now
t:   5%|□ □ □ □ □ □                   | 61/1260 [00:03<01:24, 14.19it/s, now
t:   5%|□ □ □ □ □ □                   | 63/1260 [00:03<01:23, 14.31it/s, now
t:   5%|□ □ □ □ □ □ □                  | 65/1260 [00:03<01:22, 14.46it/s, no
t:   5%|□ □ □ □ □ □ □                  | 67/1260 [00:04<01:23, 14.31it/s, no
t:   5%|□ □ □ □ □ □ □                  | 69/1260 [00:04<01:22, 14.40it/s, no
t:   6%|□ □ □ □ □ □ □                  | 71/1260 [00:04<01:22, 14.36it/s, no
t:   6%|□ □ □ □ □ □ □                  | 73/1260 [00:04<01:21, 14.53it/s, no
t:   6%|□ □ □ □ □ □ □                  | 75/1260 [00:05<03:58,  4.97it/s, no
t:   6%|□ □ □ □ □ □ □ □                 | 77/1260 [00:05<03:10,  6.21it/s, n
t:   6%|□ □ □ □ □ □ □ □                 | 79/1260 [00:05<02:37,  7.49it/s, n
t:   6%|□ □ □ □ □ □ □ □                 | 81/1260 [00:05<02:14,  8.79it/s, n
t:   7%|□ □ □ □ □ □ □ □                 | 83/1260 [00:06<01:58,  9.95it/s, n
t:   7%|□ □ □ □ □ □ □ □                 | 85/1260 [00:06<01:46, 11.07it/s, n
t:   7%|□ □ □ □ □ □ □ □ □                | 87/1260 [00:06<01:38, 11.92it/s,
t:   7%|□ □ □ □ □ □ □ □ □                | 89/1260 [00:06<01:32, 12.64it/s,
t:   7%|□ □ □ □ □ □ □ □ □                | 91/1260 [00:06<01:28, 13.22it/s,
t:   7%|□ □ □ □ □ □ □ □ □                | 93/1260 [00:06<01:24, 13.75it/s,
t:   8%|□ □ □ □ □ □ □ □ □                | 95/1260 [00:06<01:23, 13.91it/s,
t:   8%|□ □ □ □ □ □ □ □ □ □               | 97/1260 [00:07<01:21, 14.35it/s,
t:   8%|□ □ □ □ □ □ □ □ □ □               | 99/1260 [00:07<01:21, 14.33it/s,
t:   8%|□ □ □ □ □ □ □ □ □ □              | 101/1260 [00:07<01:20, 14.47it/s,
t:   8%|□ □ □ □ □ □ □ □ □ □              | 103/1260 [00:07<01:19, 14.57it/s,
```

Detecting Lanes in a video



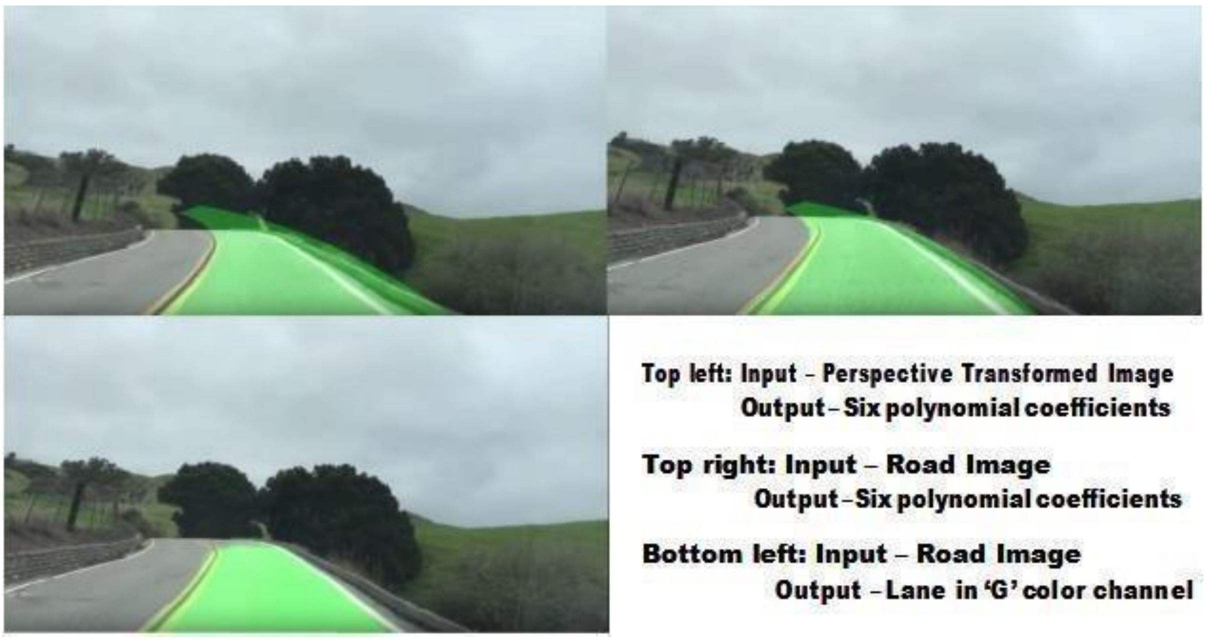
```
C:\Users\asus\Desktop\MLND>python fully_conv_NN.py
Using TensorFlow backend.
2020-12-08 17:48:19.041882: W tensorflow/stream_executor/platform/default/dso_loader.cc:55] Could not load dynamic library 'cudart64_101.dll'; dlerror: cudart64_101.dll not
 found
2020-12-08 17:48:19.048507: I tensorflow/stream_executor/cuda/cudart_stub.cc:29] Ignore above cudart dlerror if you do not have a GPU set up on your machine.
2020-12-08 17:48:51.090351: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened dynamic library nvcuda.dll
2020-12-08 17:48:53.249062: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1555] Found device 0 with properties:
pciBusID: 0000:01:00.0 name: GeForce 940MX computeCapability: 5.0
coreClock: 1.2415GHz coreCount: 3 deviceMemorySize: 2.00GiB deviceMemoryBandwidth: 13.41GiB/s
2020-12-08 17:48:53.388333: W tensorflow/stream_executor/platform/default/dso_loader.cc:55] Could not load dynamic library 'cudart64_101.dll'; dlerror: cudart64_101.dll not
 found
2020-12-08 17:48:53.399618: W tensorflow/stream_executor/platform/default/dso_loader.cc:55] Could not load dynamic library 'cublas64_10.dll'; dlerror: cublas64_10.dll not f
ound
2020-12-08 17:48:53.417401: W tensorflow/stream_executor/platform/default/dso_loader.cc:55] Could not load dynamic library 'cufft64_10.dll'; dlerror: cufft64_10.dll not fou
nd
2020-12-08 17:48:53.446088: W tensorflow/stream_executor/platform/default/dso_loader.cc:55] Could not load dynamic library 'curand64_10.dll'; dlerror: curand64_10.dll not f
ound
2020-12-08 17:48:53.468583: W tensorflow/stream_executor/platform/default/dso_loader.cc:55] Could not load dynamic library 'cusolver64_10.dll'; dlerror: cusolver64_10.dll n
ot found
2020-12-08 17:48:53.489605: W tensorflow/stream_executor/platform/default/dso_loader.cc:55] Could not load dynamic library 'cusparse64_10.dll'; dlerror: cusparse64_10.dll n
ot found
2020-12-08 17:48:54.549341: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened dynamic library cudnn64_7.dll
2020-12-08 17:48:54.554350: W tensorflow/core/common_runtime/gpu/gpu_device.cc:1592] Cannot dlopen some GPU libraries. Please make sure the missing libraries mentioned abov
e are installed properly if you would like to use GPU. Follow the guide at https://www.tensorflow.org/install/gpu for how to download and setup the required libraries for y
our platform.
Skipping registering GPU devices...
2020-12-08 17:48:54.765510: I tensorflow/core/platform/cpu_feature_guard.cc:142] Your CPU supports instructions that this TensorFlow binary was not compiled to use: AVX2
2020-12-08 17:48:54.775081: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1096] Device interconnect StreamExecutor with strength 1 edge matrix:
2020-12-08 17:48:54.785815: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1102]
Epoch 1/10
2020-12-08 17:49:50.214620: W tensorflow/core/framework/cpu_allocator_impl.cc:81] Allocation of 97124352 exceeds 10% of system memory.
2020-12-08 17:49:52.219493: W tensorflow/core/framework/cpu_allocator_impl.cc:81] Allocation of 97124352 exceeds 10% of system memory.
2020-12-08 17:49:52.304485: W tensorflow/core/framework/cpu_allocator_impl.cc:81] Allocation of 100958208 exceeds 10% of system memory.
2020-12-08 17:49:55.532857: W tensorflow/core/framework/cpu_allocator_impl.cc:81] Allocation of 100958208 exceeds 10% of system memory.
2020-12-08 17:49:56.499456: W tensorflow/core/framework/cpu_allocator_impl.cc:81] Allocation of 97124352 exceeds 10% of system memory.
12/89 [==>.........................] - ETA: 8:35 - loss: 0.1305
```

Training phase

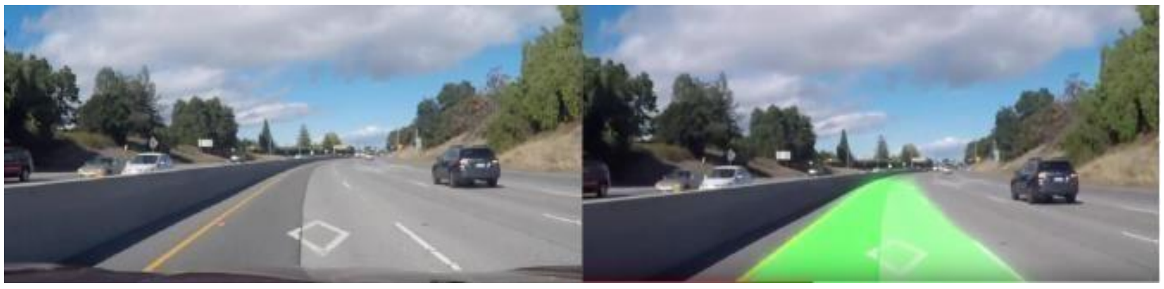
CHAPTER-6 CONCLUSION

6.1 More Visualizations

Below we have included some additional visualizations, comparing the various stages of our own model as well as in comparison to our original model using typical computer vision techniques.


Top left: Input – Perspective Transformed Image
Output – Six polynomial coefficients
Top right: Input – Road Image
Output – Six polynomial coefficients
Bottom left: Input – Road Image
Output – Lane in 'G' color channel

The CV-based model believed both lines to be on the right side of the lane, hence only a faint line and not a full lane drawn. Some of this comes downto weaknesses in the algorithm there, which lacked checks to see whether the lanes were separate from each other.



6.1.2 Reflection

The project began with collecting driving video, which we then extracted the individual frames from. After curating the data to get rid of various blurry orother potentially confusing images, we calculated the calibration needed toundistort our images, and perspective transformed them to be able to calculatethe lines. After additional image processing to improve the dataset at hand, we then created six coefficient labels, three each for both lane lines. Next, we created a program to make those labels into re-drawn lanes, and then had to improve our original label checking algorithm to work better for curves. Following this, any still poorly labeled images were removed from the dataset.

After checking histograms of the coefficient labels, we realized we needed additional curved line images, and gathered additional data for curved lines, as well as from a different camera, in order to help even out the distribution.After finding they still needed a better distribution, we found ranges of the labels to iterate through and create additional training images through rotationof the originals.

The next step was to actually build and train a model. We built a somewhatsuccessful model using perspective-transformed images, built a slightly improved model by feeding in regular road images, but still was not at a sufficient levelof quality. After trying to use activation maps of the convolutional layers,we moved on to a fully convolutional model. After changing the training labels tobe the ‘G’ color channel containing the detected lane drawing, a robust model was created that was faster and more accurate than our previous model basedon typical computer vision techniques.

Two very interesting, but very challenging issues arose during this project. We had never before used our own dataset in training a model, and curating agood dataset was a massive time commitment, and especially due to the limits of the early models We used, often difficult to tell how sufficient of a dataset we had. The second challenge was in settling on a model – we originally worried we would have to also somehow train the neural network to detect perspective transformation points or similar. Instead, we learned forthe first time how to use a fully convolutional neural network, and it solvedthe problem.

6.1.3 Improvements and future work

One potential improvement to the model could be the use of a recurrent neuralnetwork (RNN). The current version of our model uses an averaging across five frames to smooth out any issues on a single frame detection, outside ofthe actual neural network itself. On the other hand, a RNN would be able to directly look at previous frames in order to learn that what was detected in a previous frame matters to the current frame. By doing so, it would potentially lose any of the more erratic predictions entirely. We have not yet used a RNN architecture, but we plan to do so eventually for future projects.

REFERENCES

Reference workloads for modern deep learning

methods https://paperswithcode.com/task/lane-

detection/codeless https://towardsdatascience.com/lane-

detection

https://journalofcloudcomputing.springeropen.com/articles

https://doi.org/10.1016/j.icte.2020.07.007

Robust lane detection using two-stage feature extraction with curve
fitting Pattern Recognit., 59 (2016), pp. 225-233
Article

Narote Sandipann P , et al.A review of recent advances in lane detection and departure warning
system Pattern Recognit., 73 (2018), pp. 216-234

Hirano Masahiro, et al.Networked high-speed vision for evasive maneuver
assist ICT Express, 3 (4) (2017), pp. 178-182

Dubey Amartansh, Bhurchandi K.M.Robust and real time detection of curvy lanes (curves) with
desired slopes for driving assistance and autonomous vehicles

Abdelhamid Mammeri, Azzedine Boukerche, Guangqian Lu, Lane detection and tracking system
based on the MSER algorithm, hough transform and kalman filter, in: Proceedings of the 17th
ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile
Systems, 2014.

Li Mingfa, Li Yuanyuan, Jiang MinLane detection based on connection of various feature
extraction methods
Advances in Multimedia 2018 (2018)

Kim Jihun, Lee MinhoRobust lane detection based on convolutional neural network and
random sample consensus
International Conference on Neural Information Processing, Springer, Cham (2014)

Zou Qin, et al.Robust lane detection from continuous driving scenes using deep neural
networks IEEE Trans. Veh. Technol. (2019)