249 Final Project Documentation

by Taylor Yang, Sung-Yin Yang

Our goal is to build a Lane detection related application. To fulfill this goal we read several papers and try on different models. In this project, we use both traditional approaches as well as deep-learning approaches (lane segmentation) to design our application.

# Focused areas

We focused both on algorithm and application.

algorithm

For lane segmentation, we chose two deep learning models, and did comparison via data. For lane detection, we also chose both traditional methods and new deep learning methods to compare the results.

application

Our project together provides a lane detection, segmentation and navigation application. The accuracy is above 90 percent and our application can give navigation suggestions which is by telling the steering angle to let the drivers know how to steer the car. All pretrained h5 models are included.

# Overall Design

Our project includes three major parts.

* Traditional lane detection algorithm Open CV.
* Deep Learning lane segmentation algorithm
* Autonomous Lane Navigation in Deep Learning

# 

# Traditional Lane Detection Application

The problem that we are trying to solve in this part is to take a simple car-driving video from youtube as input data and process it to detect the lane within which the vehicle is moving. The application output should find a representative line for both the left and right lane lines and render those lines back to the video.

## Key Technique

### OpenCV

We use OpenCV to process the input images to discover any lane lines and rendering out a representation of the lane.

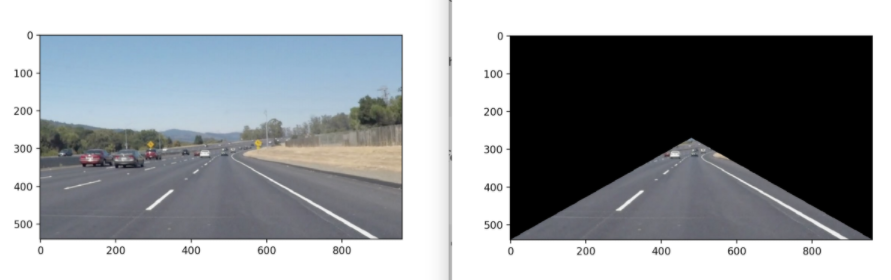
### Numpy / Matplotlib

Since images are dense matrix data, NumPy and matplotlib will be used to do transformations and render the image data.

## Implement Detail

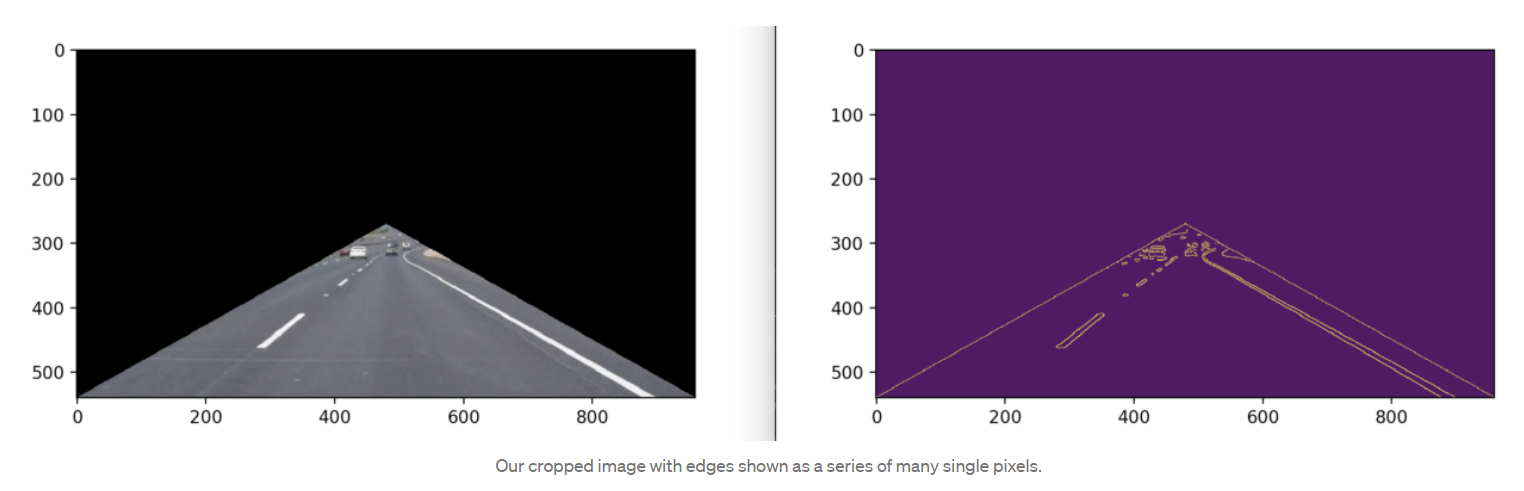
### Cropping the region of Interest

We want a region of interest that fully contains the lane lines. One simple shape that can help us to achieve this goal is to draw a triangle that starts at the bottom left corner of the images and then proceed to the center of the image. Then, we crop each driving image into the shape of a triangle as the region of interest.



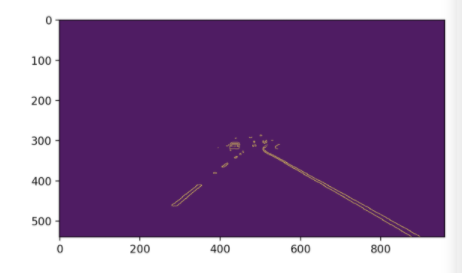
### Convert to different color spaces

Since we are detecting the lane line only, we do not care about the colors of the pictures at all. We only care about the differences between their intensity values. We can convert the image into grayscale to make the edge detection process simpler.



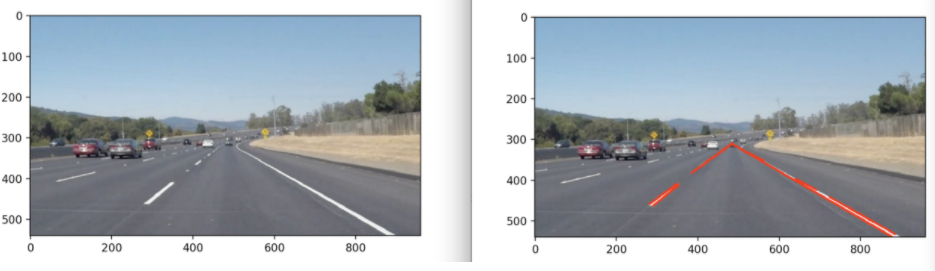
### Detecting Edges in the Cropped Image

We use OpenCV’s library to use Canny edge detection to copped our images with some reasonable thresholds. Thus, we can get simple grayscale images for later lane detection.



### Generate Lane Lines from edge pixels

We will use Hough Transform to transform all of our edge pixels into a different mathematical form. Once this is done, each edge pixel in “Image-space” will become a line or curve in “Hough Space”. Then, we can simply solve for the intersection between lines in hough space, and transform the intersection point back into image space to obtain a line that intersects enough edge pixels. An openCV function can help us to do hough transform easily.



### Repeat the step above to process every image in the videos.

Video before annotated:

<https://drive.google.com/file/d/1z_gZtwV-nirm3bEpsMA5QajfpVnDbgPe/view?usp=sharing>

Video after annotated:

<https://drive.google.com/file/d/1vNAmM1TeS2L4nzdm3OTFrMb5Iph7_A-f/view?usp=sharing>

### Section Task Distribution

Data Prepartion ： Taylor Yang, Sung-Yin Yang

Coding part：Taylor Yang, Sung-Yin Yang

Documentation: Taylor Yang, Sung-Yin Yang

## 

# Lane Segmentation

Our goal is to automatically detect the road or lane boundaries. We will use a machine learning method which is semantic segmentation for this part. We will classify all the pixels in the scene and fit into predefined road categories. We use VGG 16 classifier and implement the model introduced by [Shelhamer, Darrell and Long](https://arxiv.org/pdf/1605.06211.pdf) and also follow the instructions by

azzouz marouen.

## Key Technique

FCN networks

VGG 16 classifier

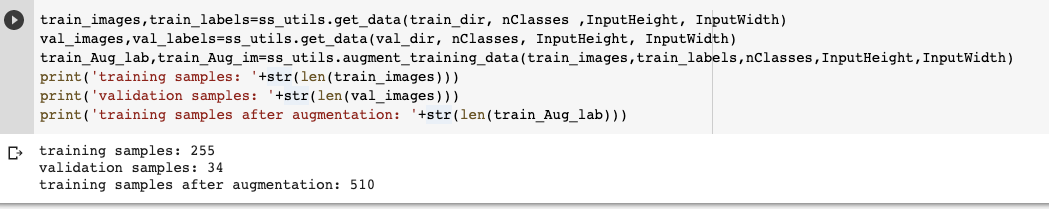
## Implementation Detail

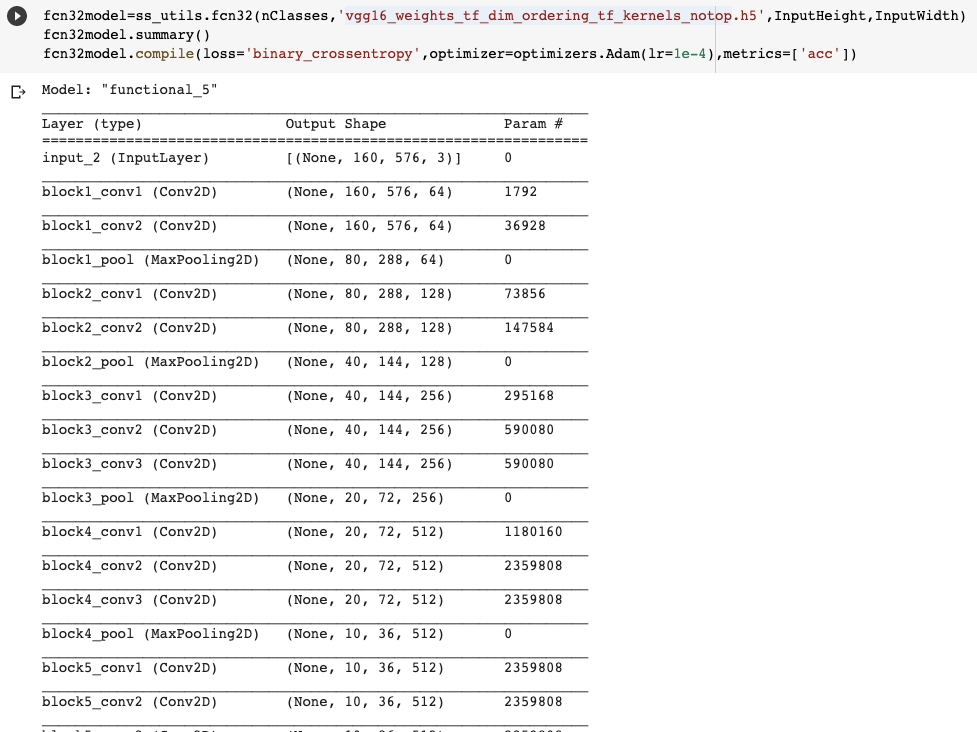
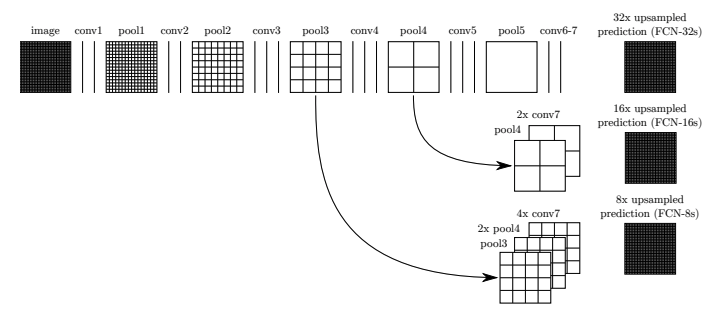
Trained the model on the [Kitti dataset](http://www.cvlibs.net/datasets/kitti/eval_road.php).

Set up the environment on Google colab and checkout GPU

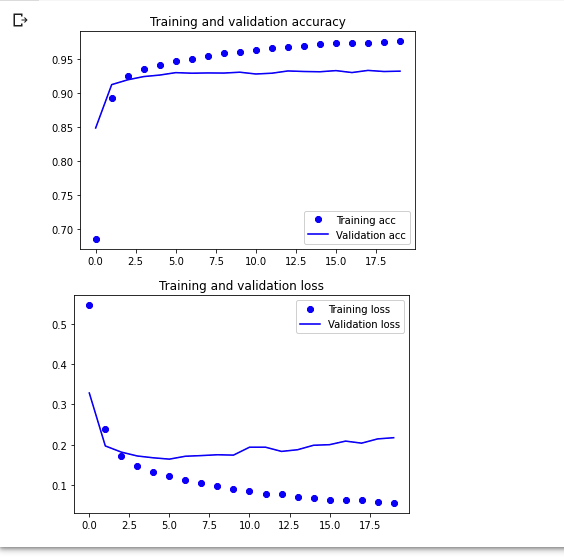
Versions Keras version: 2.4.3

Load data which downloaded previously from Kitti and accessible through Google Drives

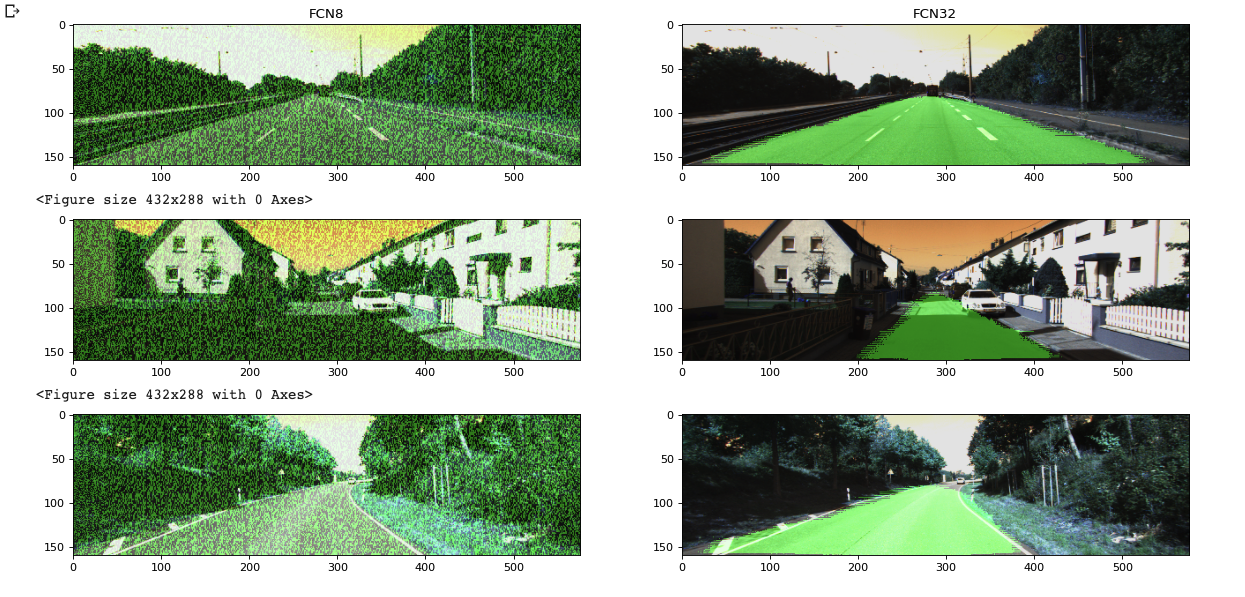


Build the FCN 32 model based on the paper by [Shelhamer, Darrell and Long](https://arxiv.org/pdf/1605.06211.pdf) and instructions from [azzouz marouen](https://github.com/mmarouen)

## Plot out the training and validation accuracy and loss



Build FCN 8 and compare it with FCN 32. Finally combine the result to the original image and plot out to see the result We also tried to build a FCN 8 model but the result did not look good. So we decided to use the FCN 32 model.



## Section Task Distribution

Data Prepartion ： Taylor Yang, Sung-Yin Yang

Coding part：Taylor Yang, Sung-Yin Yang

Documentation: Taylor Yang, Sung-Yin Yang

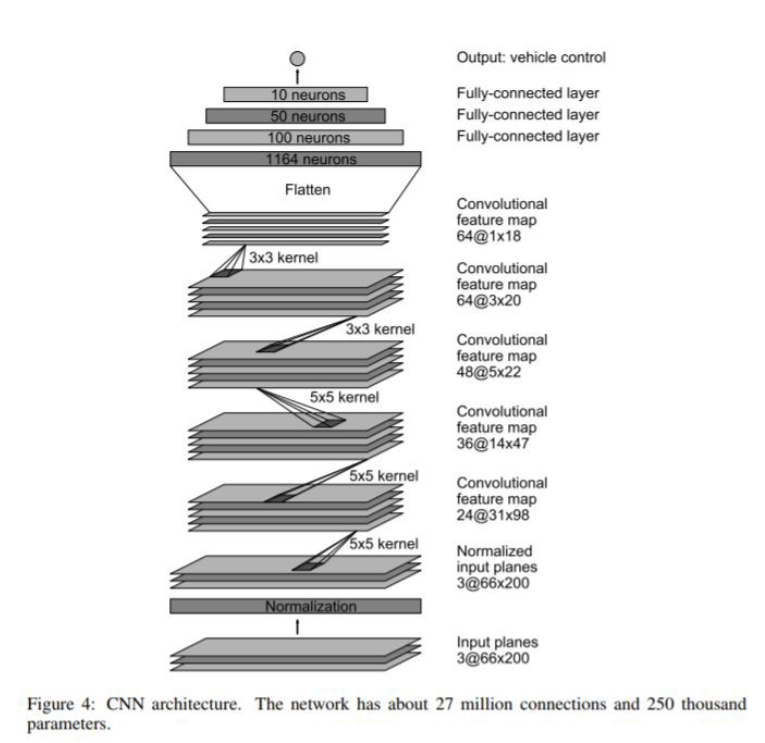
# 

# Lane Navigation in Deep Learning

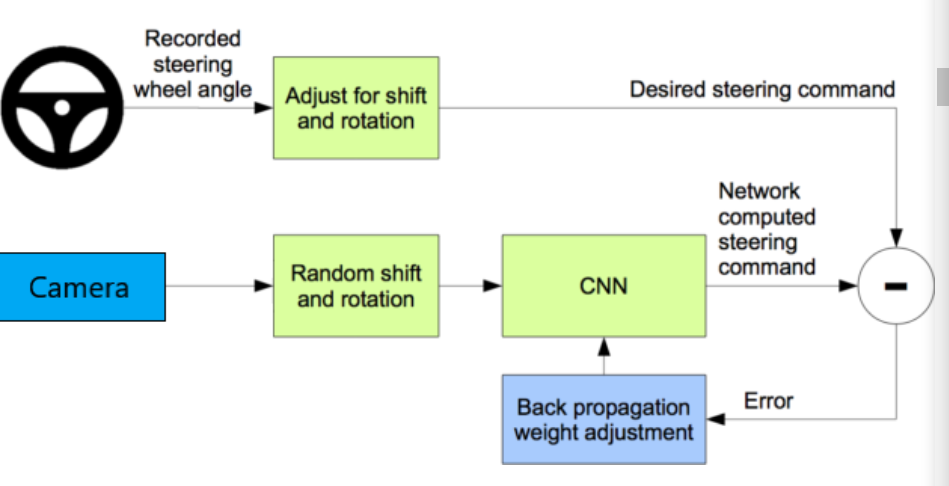
This application is based on Nvidia’s paper which trains a (CNN) to map raw pixels from a front-facing camera directly to steering commands. Out input will be the video images from DashCam and the output will be the steering angle of the car. This model would use the video image and predict the steering angle of the car.

## Key Technique

### Convolutional Neural Network



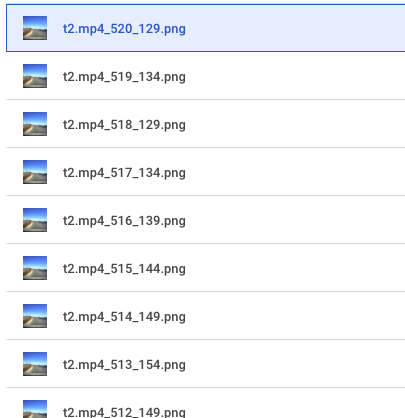
The above image is from Nvidia’s paper. It contains only 30 layers. The input image is 66\* 200 pixel image. First, the image would be normalized, then passed through 5 groups of convolutional layers. Finally, the image would pass through 4 fully connected neural layers and generate a single output, which is the steering angle of the car.



The predicted angle is then compared with the desired angle to give the video image. The error would be fed back into the CNN training process in backpropagation. As the graph shows above, this process is repeated in a loop until the loss is low enough. In fact, this is a typical image recognition training process, except that the output is a numerical value instead of a type of classified object.

## Implementation Detail

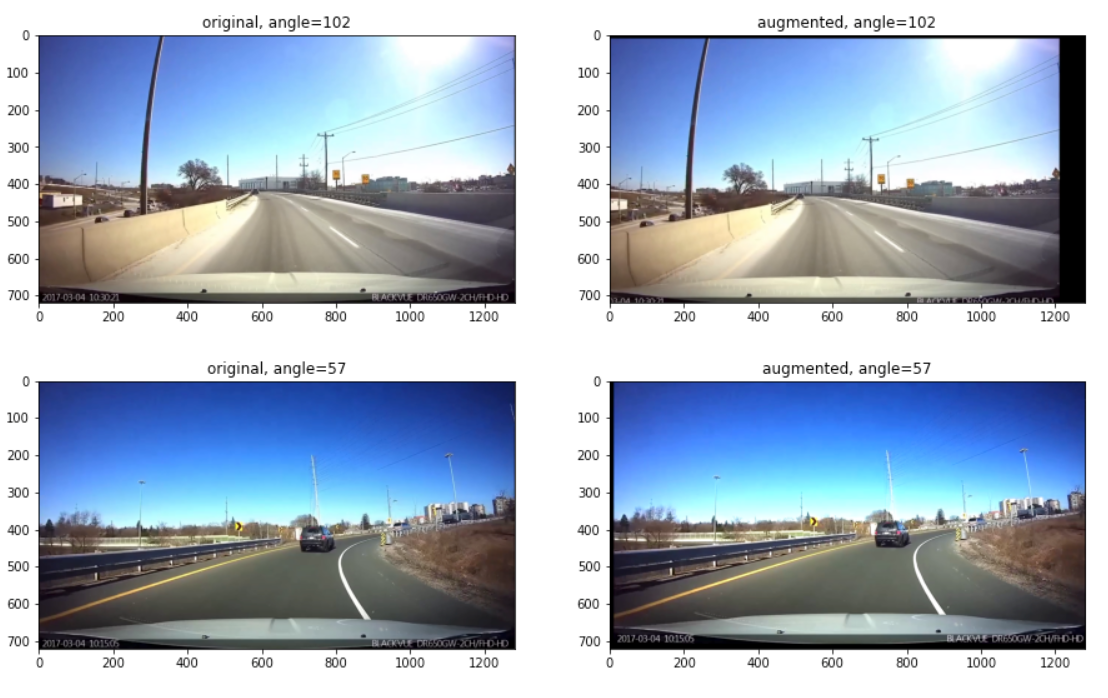
### 1. Data Acquisition



Save Previous lane result as a new dataset.

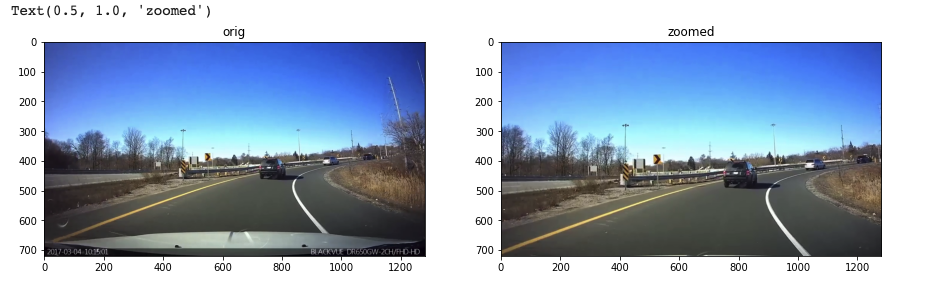
A steering angle is added.

### 2. Image Augmentation

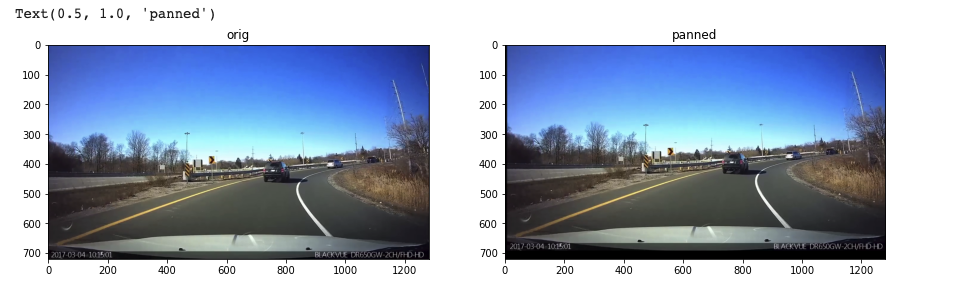


Since we only have a few hundred images, to train a deep network, we need a lot more images. Instead of running our car, let's try to augment our data. There are a couple of ways to do that.

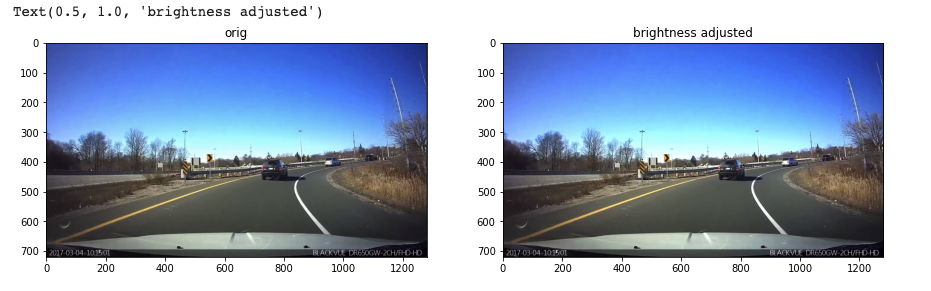
1. Zoom: Crop out a smaller image from the center



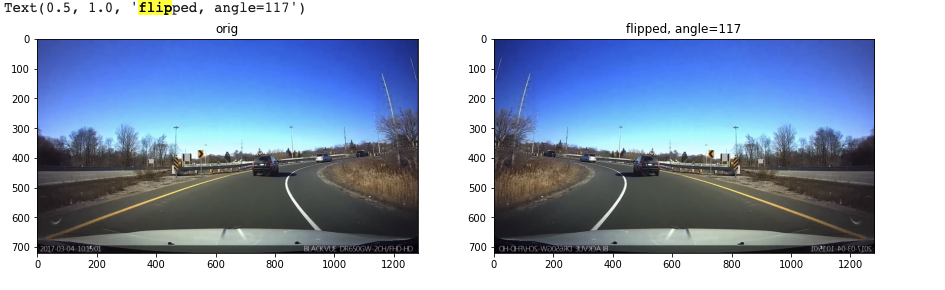
1. Pan: crop out a smaller image from the left or right side



1. adjust the brightness of the image

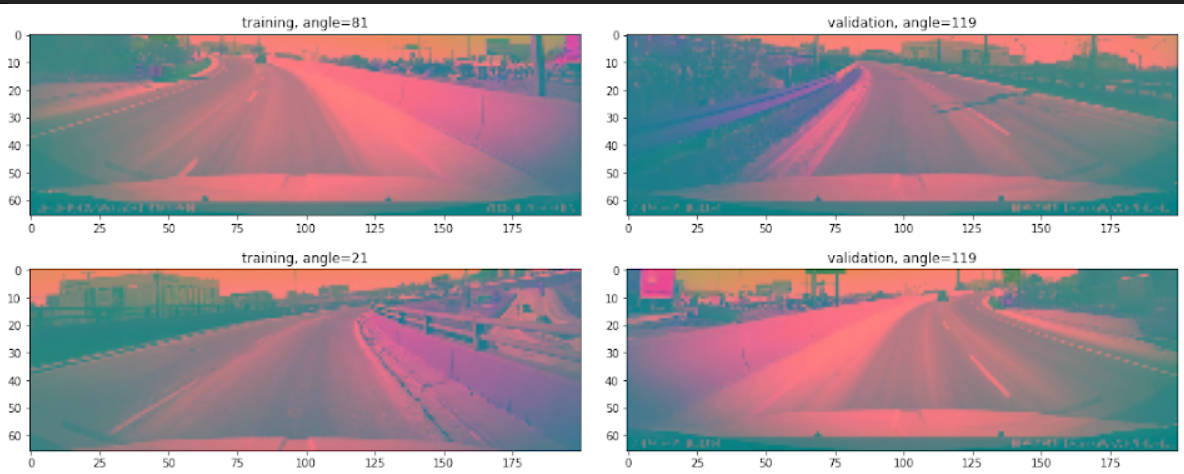


1. flip the image horizontally, i.e do a left to right flip and change the steering angle correspondingly



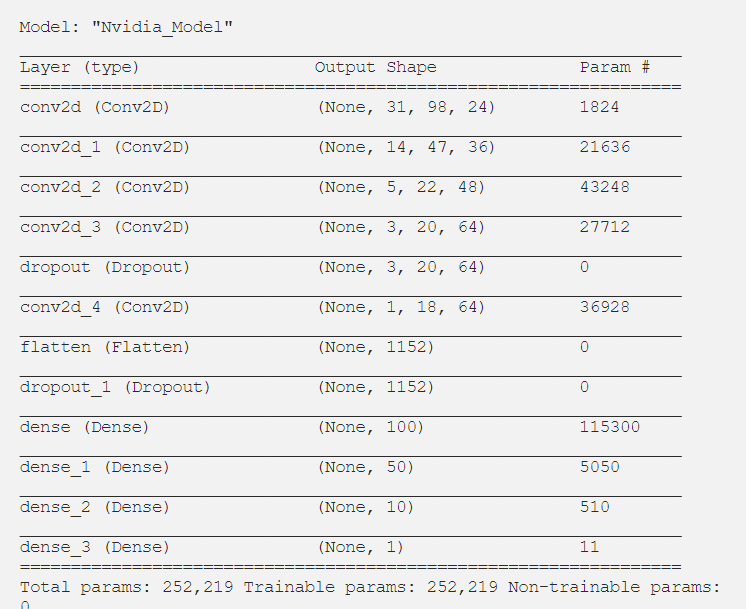
### 3. Image Process

The Nvidia model accepts input image in 200\* 66-pixel resolution. Thus, we need to change our image into a suitable color space and size. First, we would crop out the top haft of the image because it is not relevant to the steering angle. Secondly, we would change the image to YUV color space.



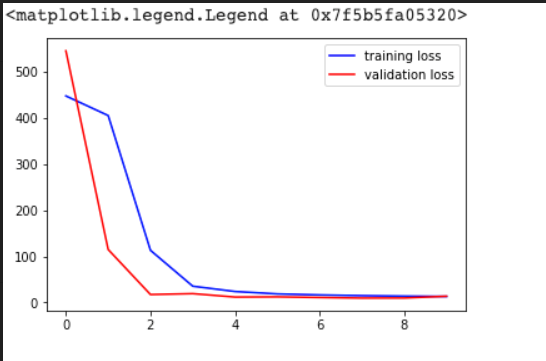
### 4. Training

We print out the parameter list. It shows that it has 250,000 parameters.

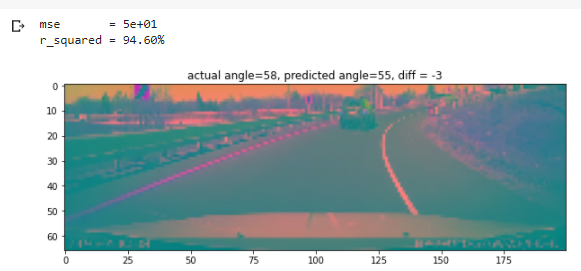


## Evaluation

It is good to see that training loss and validation loss declined rapidly together, and both of them stay low after epoch 6. It seems that there is no overfitting issue because validation loss stayed low with training loss.



Another metric that seems to perform well is the R^2 metric. As we can see in our model, we have an R^2 of 93% even with 800 images, which is primarily because we used image augmentation.



## Section Task Distribution

Data Prepartion ： Taylor Yang, Sung-Yin Yang

Coding part：Taylor Yang, Sung-Yin Yang

Documentation: Taylor Yang, Sung-Yin Yang