# Introduction

Autonomous vehicles could improve the accessibility and connectivity of the public transport system. One example would be enabling people with disabilities to reach their desired destinations in a safe and timely manner. This serves as a key motivation for this project.

# Objective

The objective of this project is to explore various deep learning models trained images from a single-lens camera to enable the JetBot to navigate a simplified road track and potentially detect objects (e.g. traffic lights.

# Dataset

## Data Collection

The data is collected by driving the JetBot around a track. The training input data is 224x224x3 images. The labels are [Left, Right] motor values.

The labels would then be one-hot encoded to [Left, Center, Right] which allows the team to control the speed of the vehicle instead of following motor values predicted by the models. Data with zero motor values are removed.

Train, test, and validation split with stratify to ensure the split has the same aspect of label ratio in each data set. The splits are 60%, 24%, and 16% respectively.

|  |  |  |  |
| --- | --- | --- | --- |
| **Train** | **Validation** | **Test** | **Total** |
| 4800 | 1920 | 1280 | 8000 |

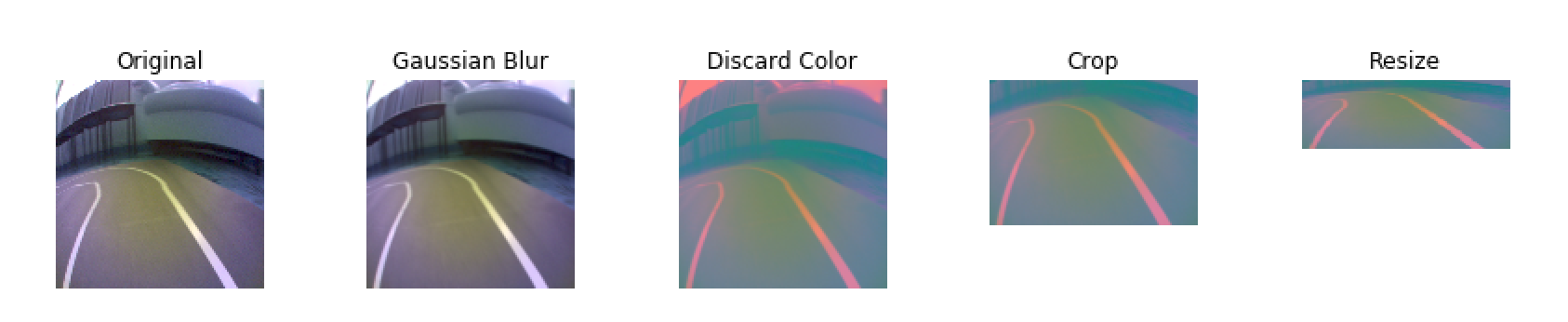
## 

## Data Processing

### Preprocessing

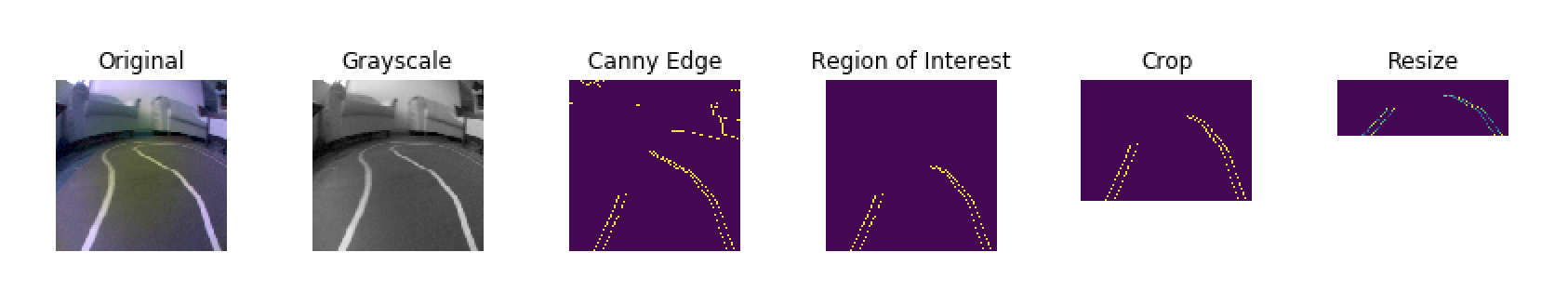
Pre-processing was performed to put more focus on the road and allow the models to perform better. The two following methods were prepared for the task.

### Method 1: Cropping and Discard Color

****This method is recommended for better performance on Nvidia’s PilotNet. Gaussian Blur is applied to the image to achieve a bokeh effect followed by a colour transformation from BGR to YUV and colour normalization. The top of the image is cropped away to focus on the road and finally resized to the shape [66,224,3].

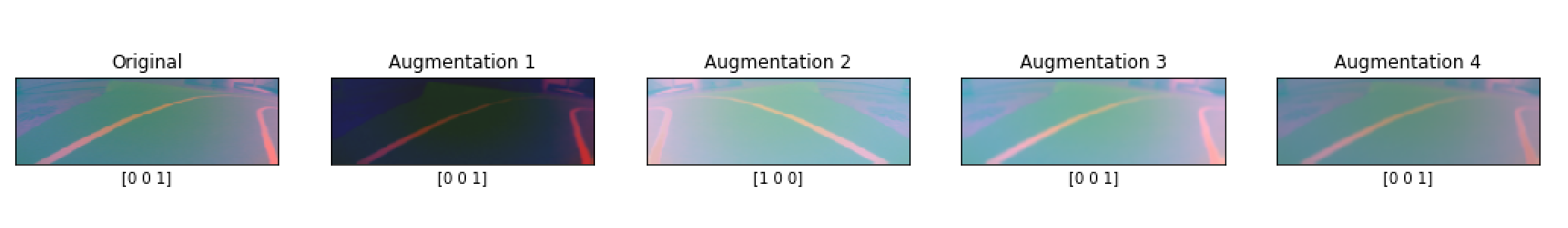
### 

### Method 2: Lane Masking using OpenCV



For this method, grayscale and canny edge detection is applied to detect edges, after which only the information within the specified region of interest is retained. The image is finally cropped and resized to the shape [66,224,1].

### Augmentation



The preprocessed image is fit into the model with a custom image generator. To generate data with more variety and make the model robust towards different scenarios, Blurring & Brightness augmentation was added to handle glare, channel shifting to simulate sunlight, streetlight & under shades, Random zoom of 10% for variety and Horizontal flip along with label flipping of left and right images to create more data for turning.

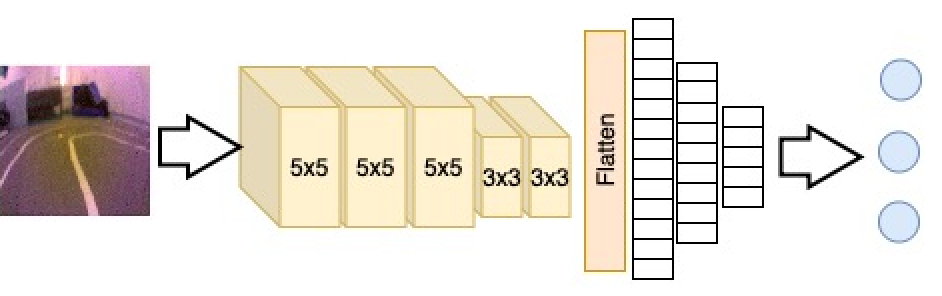
# Model Development Log

## 

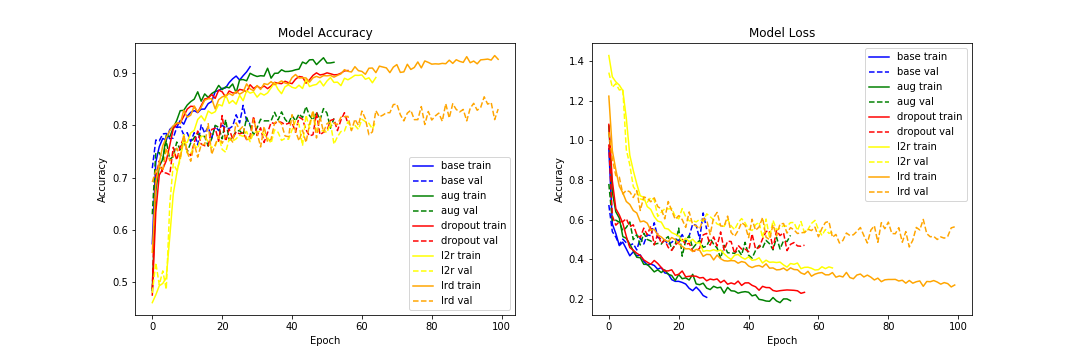
## Model 1 - PilotNet (Lane Following)

To follow the lane, the model needs to know where the lane lines are. Therefore this is a computer vision problem. For this model, Nvidia’s PilotNet is referenced. PilotNet is Nvidia’s end-to-end deep learning steering system.

### Model Architecture and summaries



### Hyperparameters Fine Tuning and Training Results



|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Hyperparams** | **Prec. (Left)** | **Prec. (Center)** | **Prec. (Right)** | **Rec. (Left)** | **Rec. (Center)** | **Rec. (Right)** | **F1 (Left)** | **F1 (Center)** | **F1 (Right)** |
| Base | 0.82 | 0.74 | 0.83 | 0.79 | 0.87 | 0.56 | 0.8 | 0.8 | 0.67 |
| Augmentation | 0.87 | 0.76 | 0.88 | 0.77 | 0.91 | 0.64 | 0.82 | 0.83 | 0.74 |
| Dropout (0.4) | 0.88 | 0.75 | 0.87 | 0.75 | 0.92 | 0.61 | 0.81 | 0.82 | 0.72 |
| L2 Reg (0.001) | 0.88 | 0.75 | 0.84 | 0.71 | 0.9 | 0.68 | 0.79 | 0.82 | 0.75 |
| All + LR Decay | 0.83 | 0.81 | 0.88 | 0.85 | 0.87 | 0.73 | 0.84 | 0.84 | 0.8 |

## Model 2 - PilotNet + LSTM (Lane Following)

A driver would make steering decisions with some memory of what he saw earlier. Therefore this model proposes to have PilotNet being integrated with Long Short Term Memory (LSTM) neural network. LSTM has the ability to perform a prediction task based on the information from previous time steps.

### Input

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Preprocessing Method** | **Sequence** | **Height** | **Width** | **Channels** |
| Colour Discarding | 5 | 66 | 200 | 3 |
| Line Extraction | 5 | 66 | 200 | 1 |

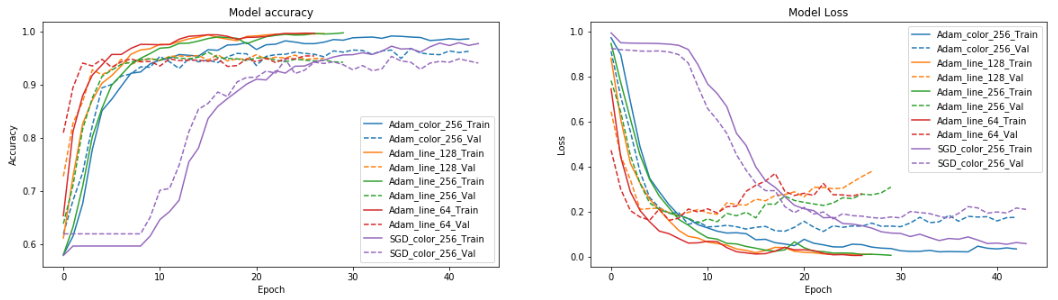
There are around 60% straight labels, 23% left labels, and 17% right labels. The train test split for training this model is 80% train and 20% test. Five timesteps were decided due to the nature of the hardware that the model is deployed to. There is also preprocessing involved to turn images into sequence of five images for the model.

### Model Architecture

|  |  |
| --- | --- |
|  |  |

The above shows the model architecture printed through Keras. The left image shows the PilotNet’s CNN layers, whereas the right image shows LSTM with 128 units and four dense layers. The four dense layers belonged to PilotNet.

### Hyperparameters Fine Tuning and Training Results



Adam optimizer with the first preprocessing method (colour) and batch size 256 was the first trained model.

A noticeable trend in these graphs is that the model tends to start overfitting around 10 epochs. An attempt to regularize the model was tried by reducing the batch size. Though the losses seem to be lower than its higher batch size counterpart, the following F1 scores showed there is not much impact.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Preprocess** | **Optimizer** | **Batch Size** | **Acc (Train)** | **Acc (Val)** | **F1 (Left)** | **F1 (center)** | **F1 (Right)** |
| Crop 0.3, Color YUV | SGD, LR 0.01, Momentum 0.9, Nesterov | 256 | 97.747 | 94.071 | 0.54 | 0.74 | 0.48 |
| Crop 0.3, Color YUV | Adam, LR 0.001 | 256 | 98.636 | 96.285 | 0.65 | 0.79 | 0.61 |
| **Crop 0.3, Line Extract** | **Adam, LR 0.001** | **256** | **99.763** | **94.229** | **0.73** | **0.82** | **0.7** |
| Crop 0.3, Line Extract | Adam, LR 0.001 | 128 | 99.644 | 94.941 | 0.69 | 0.8 | 0.68 |
| Crop 0.3, Line Extract | Adam, LR 0.001 | 64 | 99.684 | 95.415 | 0.68 | 0.81 | 0.68 |

The bold F1 score result in the table is the model we deployed into the bot for the final demo. F1 score was chosen as a benchmark due to highly imbalanced classes. Stochastic Gradient Descent (SGD) with momentum was chosen as it is a very popular optimization algorithm and a potential alternative to Adam. SGD took longer to converge and the F1 score for left and right is noticeably worse than with Adam.

Batch size reduction also does not see a noticeable improvement in the F1 scores. There is a slight improvement when line extraction preprocessor was employed before training.

## Model 3 - Traffic Light Detection (Object Detection)

The Traffic Light Detection model is trained using Tensorflow Object Detection API on Colab. Traffic lights are labelled with 3 classes: red, amber and green. 60 images from each class are captured using JetBot for training. The model is using SSD Mobilenet V2 with Coco dataset as the pre-trained weight for training. Coco dataset is chosen as the weight is pre-trained with traffic light image hence it is good to use for transfer learning.

### 

### Performance

Performance of the detection works great near the 3000 steps mark. The model is able to yield an accuracy of 65% before 5000 steps. However, the model still performs well in detecting traffic light labels and box locations.

|  |
| --- |
|  |

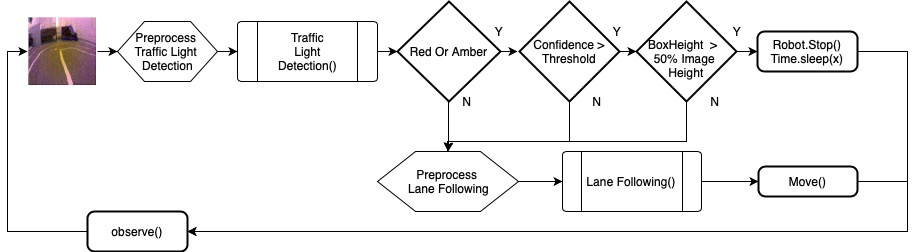
**Sample Detection**

|  |
| --- |
|  |

### Integration

Firstly, trained SSD frozen graph is exported to TFLite graph using a special function from TensorFlow folder, export\_tflite\_ssd\_graph.py and then convert to tflite using tflite\_convert.

The integration of the models will be done via rule based pseudocode in JetBot’s driving function. The function will inference both models (Lane Following and Traffic Light detection) by sequence as described in the flowchart below:



Tensor Shape: (1,300,300,3)

# Model Deployment

The JetBot is currently run from the Jupyter Notebook. During testing of the bot with various models, it became apparent that downloading and uploading of models is not efficient and prone to errors. Therefore the team loaded models into Github. The notebook will pull the latest models down to the bot.

After pulling down the latest models, the bot would have to make two decisions, what the preprocessed image dimension should be and is the model a sequence or cnn model. These could be decided based on the input layer of the models.

The team was unable to convert the LSTM model into the TFLite model, as the conversion could not recognize certain operations within the Keras model. Whereas PilotNet model has no issue converting to TFLite model.

## Deployment Platform

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Maker** | **OS** | **TF Version** | **Camera** | **Storage** |
| **JetBot 1** | NYP | Jetpack 4.2 | 1.13.1 | 8mp FOV 160 | Sandisk Extreme 64gb |
| **JetBot 2** | Waveshare | Jetpack 4.3 | 1.14 | 8mp FOV 160 | Sandisk Ultra Class 1 64gb |

# Areas for Improvements

Based on the results above, there are two broad areas that could be improved upon. One would be data and the other tweaking of model layers.

## Data

During data collection, there were manual and minute adjustments at the corners. This could have resulted in the bot having difficulty deciding whether to turn left or right.

## Models

Both types of models exhibit overfitting during training at around the same epoch numbers. This brings a question of could the model be too complex? PilotNet is used to process images that are more complex than this project’s simplified track. Therefore, could there be some dense layers that could be removed, or lesser kernels could be deployed.

# Source Code Repositories

<https://github.com/relaxauto/self-driving-bot>

# Contributions

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Project Member | Discussion | Research | Track Building | Data Collection | Data Processing | Helper Functions | Modeling & Tuning | Evaluation & Analysis | Deployment | Documentation |
| Ng Chong Soon | Y | Y |  |  | Y | Y | Y | Y | Y | Y |
| Lee Chun Wai | Y | Y | Y | Y | Y | Y | Y | Y |  | Y |
| Lee Kok Chuan | Y |  | Y | Y |  |  | Y | Y |  | Y |
| Yap Wei Tian | Y |  | Y | Y |  |  | Y |  |  | Y |

# 

# References

1. Mariusz Bojarski, Philip Yeres, Anna Choromanska, Krzysztof Choromanski, Bernhard Firner, Lawrence Jackel, and Urs Muller, “Explaining How a Deep Neural Network Trained with End-to-End Learning Steers a Car”, April 27 2017. URL: <https://arxiv.org/abs/1704.07911>, arXiv:arXiv:1704.07911.
2. Hesham M. Eraqi, Mohamed N. Moustafa, Jens Honer, “End-to-End Deep Learning for Steering Autonomous Vehicles Considering Temporal Dependencies”, Nov 01 2017. URL: <https://openreview.net/pdf?id=rys_hvLRW>
3. Vijay Ramakrishnan, “Road Lane Line Detection using Computer Vision models”, July 2017. URL: [https://www.kdnuggets.com/2017/07/road-lane-line-detection-using-computer-vision-models.htm](https://www.kdnuggets.com/2017/07/road-lane-line-detection-using-computer-vision-models.html)