Programming an Autonomous Driving Car with J.K. Rollin

Jitendra Awasthi & Kam Mirhosseini

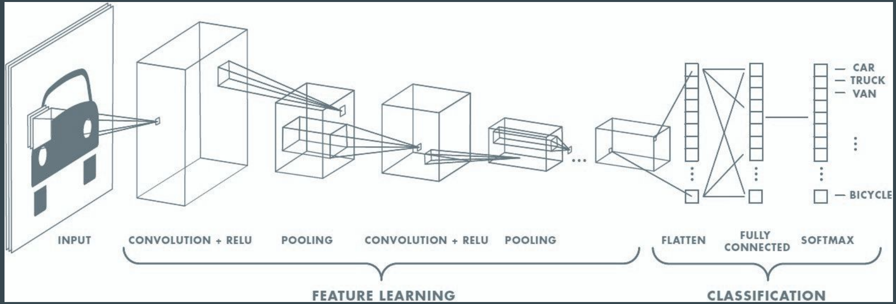
Introduction: Autonomous cars

What is an autonomous car?

An autonomous car is a vehicle capable of sensing the environment around it and operating without human involvement.

- Reduce traffic congestion
- Cut transportation costs (in terms of vehicles, fuel, and infrastructure)
- Reduce traffic accidents
- Free up parking lots for other uses
- Reduce urban CO_2 emissions worldwide

Introduction: Deep Learning & CNNs



Before Deep Learning Era

- HAAR (Viola & Jones, 2001)
- Local Binary Patterns (LBP)(Ojala et al., 1996).

Rise of Deep Learning and CNN

- Recent development in hardware and affordability
- Increase of computational power
- Availability of big data

CNNs were some of the first deep models to solve commercial applications.

One of the first examples of these networks is the AT&T in 1990.

The ImageNet challenge led to some amazing computer vision networks such as, AlexNet and ResNet.

Introduction: The task and approaches

The task: Predicting angle and speed based off of images

Main approaches:

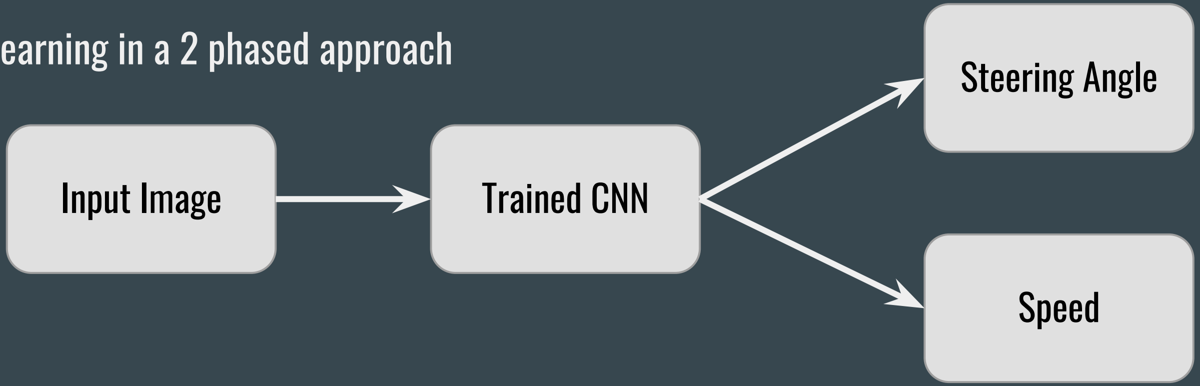
- Rule-based learning
- End-to-end learning

Choice of algorithms:

- Supervised learning
- Unsupervised learning
- Reinforcement Learning

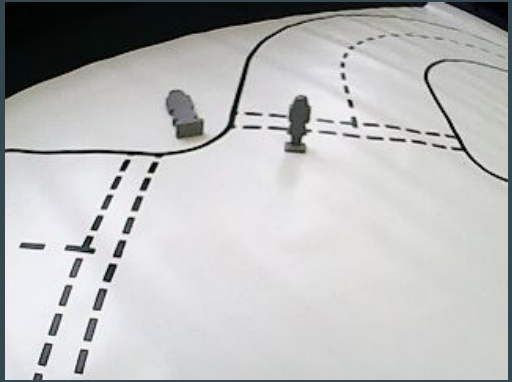
We used end-to-end supervised learning in a 2 phased approach

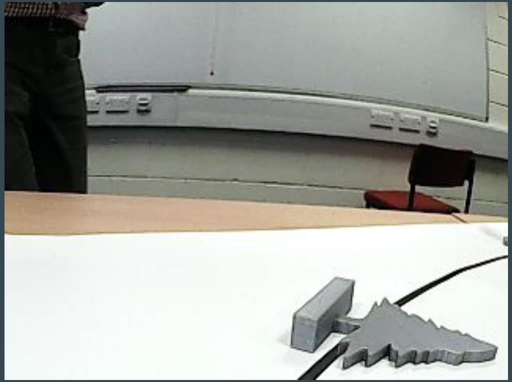
- Object classification
- Regression



Data

1. Total images: 13793
2. We observed that left turn images are under-represented.
3. Left arrows were less than right arrows
4. Tricky images e.g. tree on the side and special scenarios
5. Gender bias towards female figures
6. Inappropriate values in training_norm.csv:
    - Filename 4725 speed should be 0
    - Filename 1447 angle should be close 0.5





Data Processing: Augmentation

- Noise
- Brightness
- Exposure
- Blur
- Zoom
- Pan
- Flip
- Rotation

We had to flip the angle accordingly.

As there was more right turns on the training data, the threshold for applying random flip augmentation was increased to 0.7 in order to have even number of left/right turns.

Models

Tools used in our models

APIs:

- Tensorflow Keras functional API
- For the Nvidia model, sequential API

Layers:

- CNNs
- Dense
- Dropout (Srivastava et al., 2014)
- Batch Normalisation

Activation function: ELU and Sigmoid

Optimizer:

- The Adam optimizer (Kingma & Ba, 2014)

Regularizer:

- Tried regularisation but we did not get any improvement

Early stopping in label detection with a set patience

Checkpoint callbacks to be able to go back to a previous model if overfitting occurs.

Model 1: Nvidia Model (Bojarski et al., 2016)

Input and preprocessing: Full image resized to half

Before trying to create our own model we tried out the Nvidia Model.

We changed the output dense layer to give 2 predictions instead of 1, which gave us our first kaggle submission.

Final Loss: 0.07582

Score: 0.05893

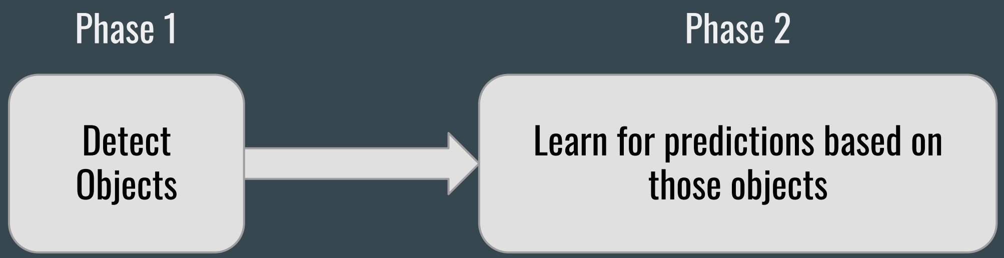
Score: 0.04830

Layer (type)	Output Shape	Param #
conv2d_5 (Conv2D)	(None, 31, 98, 24)	1824
conv2d_6 (Conv2D)	(None, 14, 47, 36)	21636
conv2d_7 (Conv2D)	(None, 5, 22, 48)	43248
conv2d_8 (Conv2D)	(None, 3, 20, 64)	27712
dropout_2 (Dropout)	(None, 3, 20, 64)	0
conv2d_9 (Conv2D)	(None, 1, 18, 64)	36928
flatten_1 (Flatten)	(None, 1152)	0
dropout_3 (Dropout)	(None, 1152)	0
dense_4 (Dense)	(None, 100)	115300
dense_5 (Dense)	(None, 50)	5050
dense_6 (Dense)	(None, 10)	510
dense_7 (Dense)	(None, 2)	22

Total params: 252,230
Trainable params: 252,230
Non-trainable params: 0

None

Transfer Learning



We removed the last layers of each model and concatenate the last layers and added extra dense layers as needed.

We froze the entire model except the new added layers and we trained the new layers for a few epochs to warm up the new layers.

We then unfroze the entire model and trained with a low learning rate.

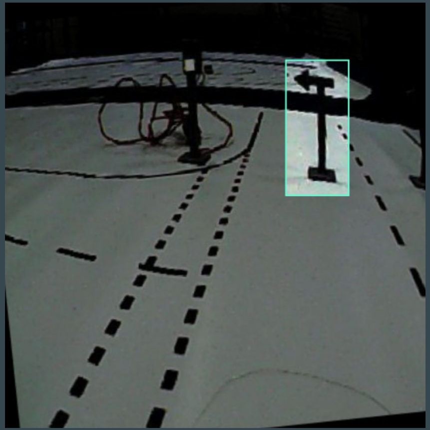
We tried a combination of training approaches.

Label Detector Model Based on Nvidia

We used LabelIMG and Roboflow to label the objects and make augmentations.

Our labels consisted of:

1. Person
2. Red Light
3. Green Light
4. Right Sign
5. Left Sign
6. Tree
7. Box
8. Car



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 58, 78, 24)        1824
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 27, 37, 36)        21636
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 12, 17, 48)        43248
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 10, 15, 64)        27712
_________________________________________________________________
dropout (Dropout)            (None, 10, 15, 64)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 8, 13, 64)         36928
_________________________________________________________________
flatten (Flatten)            (None, 6656)              0
_________________________________________________________________
dropout_1 (Dropout)          (None, 6656)              0
_________________________________________________________________
dense (Dense)                (None, 100)               665700
_________________________________________________________________
dense_1 (Dense)              (None, 50)                5050
_________________________________________________________________
dropout_2 (Dropout)          (None, 50)                0
_________________________________________________________________
dense_2 (Dense)              (None, 10)                510
_________________________________________________________________
dense_3 (Dense)              (None, 8)                 88
=================================================================
Total params: 802,696
Trainable params: 802,696
Non-trainable params: 0
_________________________________________________________________
None
```

Model 2: The Twin Engine Model

Transfer Learning using Label Detector for speed

- Full image
- Resized

Label Detector Model Based on Nvidia

Conv 24 | Conv 36 | Conv 48 | Conv 64 | Conv 64 | Flatten | Dropout

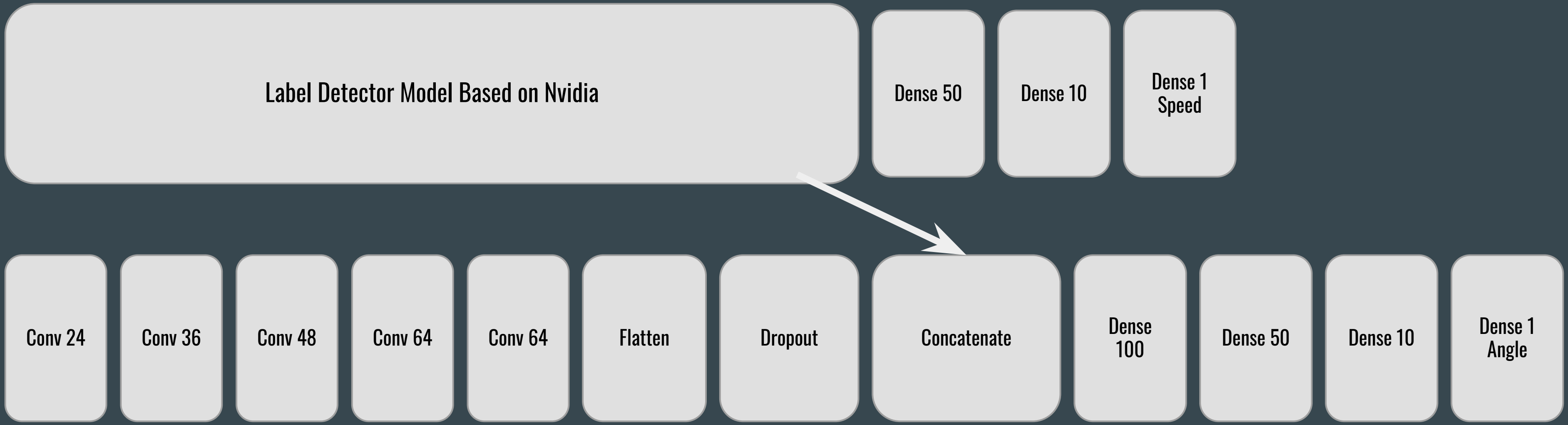
Concatenate | Dense 100 | Dense 50 | Dense 10 | Dense 2

Model for angle detection

- Half Image
- Resized
- Grayscale

Kaggle Score: 0.03358

Model 2: Experiment



Single Engine Model

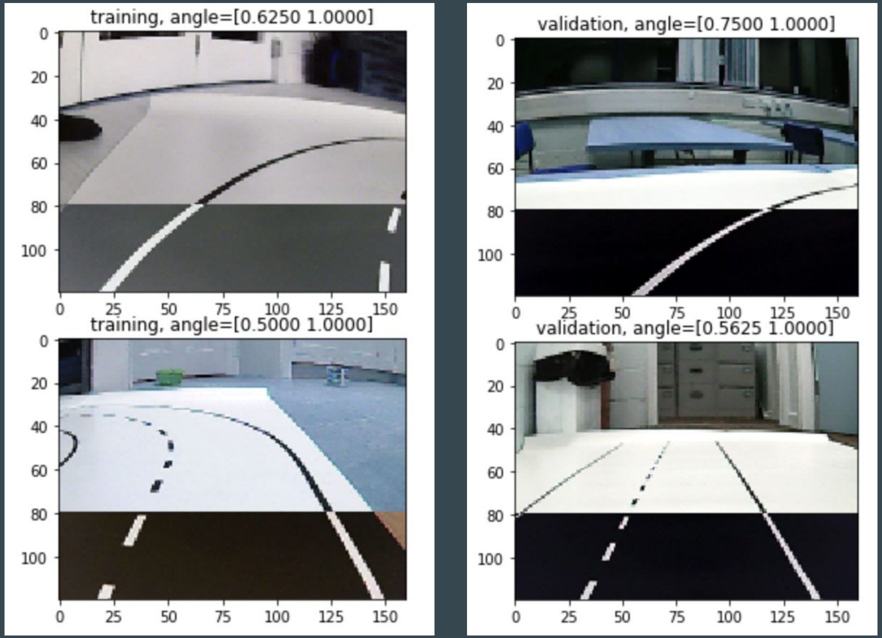
We were giving two sets of images to the twin engine model so we decided to get the same effect from a single image.

At this stage, we decided to try segmentation.

We inverted the lower segment of the images hoping it would learn both angle and objects in a single model.

Once again, the angle was not predicted based on the arrows.

However, we came back to this model later



We moved on from the Nvidia model to our own model

Divide and Rule (PREDICT)

Angle:

Full Image
Preprocessing:

- Grayscale
- Inverted lower ⅓ of the image
- Resize & Normalise

Label Detection
Transfer Learning

Speed:

Full Image
Preprocessing:

- Resize & Normalise

Label Detection
Transfer Learning

Model 3: Twin Engine v.2.0

Phase 1: Label Detection for Speed related objects

For Speed we used labels:

1. Person
2. Red Light
3. Green Light
4. Tree
5. Box
6. Car

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[(None, 120, 160, 3)]	0
conv2d (Conv2D)	(None, 118, 158, 32)	896
conv2d_1 (Conv2D)	(None, 116, 156, 32)	9248
max_pooling2d (MaxPooling2D)	(None, 58, 78, 32)	0
conv2d_2 (Conv2D)	(None, 56, 76, 48)	13872
conv2d_3 (Conv2D)	(None, 54, 74, 48)	20784
max_pooling2d_1 (MaxPooling2	(None, 27, 37, 48)	0
conv2d_4 (Conv2D)	(None, 25, 35, 64)	27712
conv2d_5 (Conv2D)	(None, 23, 33, 64)	36928
max_pooling2d_2 (MaxPooling2	(None, 11, 16, 64)	0
flatten (Flatten)	(None, 11264)	0
dropout (Dropout)	(None, 11264)	0
dense (Dense)	(None, 256)	2883840
dense_1 (Dense)	(None, 100)	25700
batch_normalization (BatchNo	(None, 100)	400
dense_2 (Dense)	(None, 50)	5050
dense_3 (Dense)	(None, 10)	510
angle (Dense)	(None, 1)	11

Total params: 3,024,951
Trainable params: 3,024,751
Non-trainable params: 200

Model 3: Twin Engine v.2.0

Phase 1: Label Detection for Angle related objects

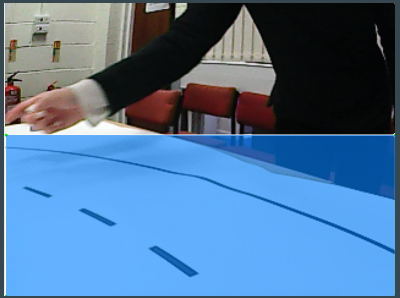
For Angle we used:

1. Right Sign
2. Left sign

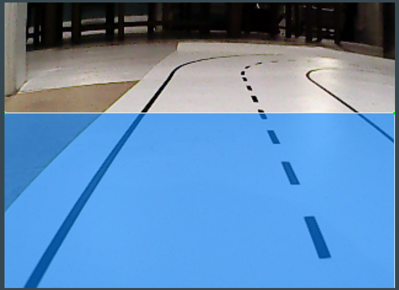
We also experimented labelling the road to detect angles. Starting with 0, 0.5 and 1

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[(None, 120, 160, 1)]	0
conv2d (Conv2D)	(None, 118, 158, 32)	320
conv2d_1 (Conv2D)	(None, 116, 156, 32)	9248
max_pooling2d (MaxPooling2D)	(None, 58, 78, 32)	0
conv2d_2 (Conv2D)	(None, 56, 76, 48)	13872
conv2d_3 (Conv2D)	(None, 54, 74, 48)	20784
max_pooling2d_1 (MaxPooling2	(None, 27, 37, 48)	0
conv2d_4 (Conv2D)	(None, 25, 35, 64)	27712
conv2d_5 (Conv2D)	(None, 23, 33, 64)	36928
max_pooling2d_2 (MaxPooling2	(None, 11, 16, 64)	0
flatten (Flatten)	(None, 11264)	0
dropout (Dropout)	(None, 11264)	0
dense (Dense)	(None, 256)	2883840
dense_1 (Dense)	(None, 100)	25700
batch_normalization (BatchNo	(None, 100)	400
dense_2 (Dense)	(None, 50)	5050
dense_3 (Dense)	(None, 10)	510
angle (Dense)	(None, 1)	11

Total params: 3,024,375
Trainable params: 3,024,175
Non-trainable params: 200



0



0.5

1

Model 3: Twin Engine v.2.0

Phase 2: Transfer Learning

In phase 2 we trained to predict for angle and speed and combined them using all the steps for Transfer learning.

This model was more successful and resulted in our best kaggle score of 0.03197

Why we did not use this was because the model was big in terms of inference time.

We tried many approaches to cut down the model.

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[(None, 120, 160, 1)]	0
conv2d (Conv2D)	(None, 118, 158, 32)	320
conv2d_1 (Conv2D)	(None, 116, 156, 32)	9248
max_pooling2d (MaxPooling2D)	(None, 58, 78, 32)	0
conv2d_2 (Conv2D)	(None, 56, 76, 48)	13872
conv2d_3 (Conv2D)	(None, 54, 74, 48)	20784
max_pooling2d_1 (MaxPooling2	(None, 27, 37, 48)	0
conv2d_4 (Conv2D)	(None, 25, 35, 64)	27712
conv2d_5 (Conv2D)	(None, 23, 33, 64)	36928
max_pooling2d_2 (MaxPooling2	(None, 11, 16, 64)	0
flatten (Flatten)	(None, 11264)	0
dropout (Dropout)	(None, 11264)	0
dense (Dense)	(None, 256)	2883840
dense_1 (Dense)	(None, 100)	25700
batch_normalization (BatchNo	(None, 100)	400
dense_2 (Dense)	(None, 50)	5050
dense_3 (Dense)	(None, 10)	510
angle (Dense)	(None, 1)	11

Total params: 3,024,375
Trainable params: 3,024,175
Non-trainable params: 200

Twin Engine v2.1

The layer we removed was *dense(256)* layer with 2883840 number of parameters.

This model started to perform faster, however, integrating this model with the code we realised we still need to further reduce the inference time and the kaggle score went down.

Kaggle Score: 0.04030

Twin Engine v2.2

We modified the first two convolutional layers by changing the *filter size (5, 5)* and added stride (2, 2). We added the *dense(256)* layer back.

Speed Parameters: 3,024,951 → + Stride & Step Size → Speed Parameters: 383,267

Angle Parameters: 3,024,375 → + Stride & Step Size → Angle Parameters: 383,223

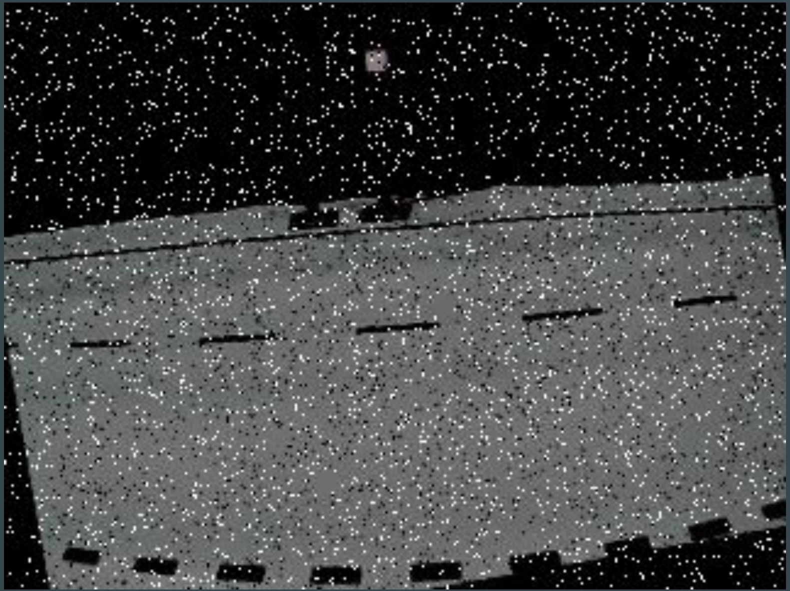
The Final Day

The kaggle submission for the Twin Engine model that used labels, was 0.06192

We realised the new images added recently were distorted by over-augmentation to a point where different arrows and traffic lights could not be distinguished.

We removed the overly augmented images and trained overnight on the remaining data without training for object detection.

In the morning we made a better representation of all special scenarios into one folder. We re-trained the model for speed and angle separately and combined them. We saw improvement on some special scenarios but we did not realise this training resulted in some association between straight road and (speed=0).



How we would do the project if we had to do it again

Begin the project by focusing on the dataset

Make sure every scenario is properly represented

To have better control on training we would use a custom data generator.

Retraining the model on special subset is tricky and needs more attention.

Labelling the obstacles as 1 category (may or may not help). Try anyways!

We learnt a lot by trying various approaches, however, if we focused on one model and identified the problems we would have had a better performance.

Too much excitement over kaggle :)

Redesigning the training data

Original: 13793 → + Flipped → Total: 27586

\+

Special Images: 1879 → Augmentation → Special Images: 5638 → + Flipped → Special Images: 11276

=

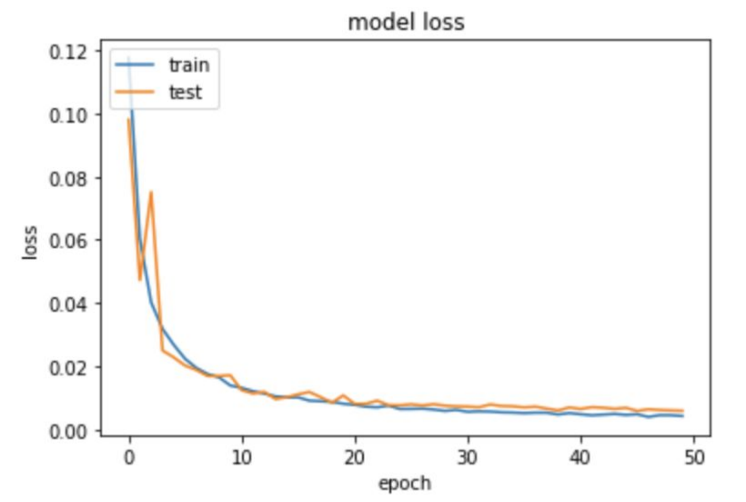
Training data: 38862

We wrote a custom data generator to have more control over the training data

Revisiting The Sweet & Simple Model

Input and Preprocessing:

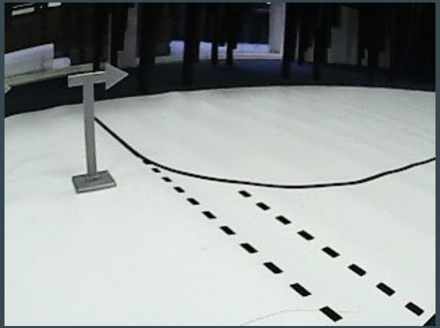
- Full Image
- Inverted the lower (Approx.) ⅓ of the image
- Resized to half
- Normalised



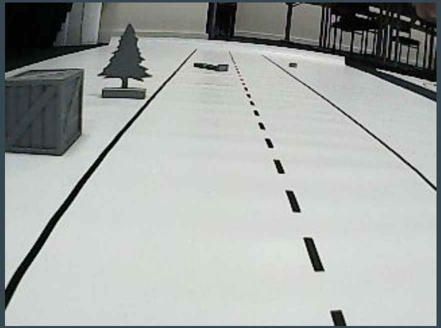

```
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 120, 160, 3)]     0
_________________________________________________________________
conv2d (Conv2D)              (None, 58, 78, 32)        2432
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 27, 37, 32)        25632
_________________________________________________________________
dropout (Dropout)            (None, 27, 37, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 25, 35, 48)        13872
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 23, 33, 48)        20784
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 11, 16, 48)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 9, 14, 64)         27712
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 7, 12, 64)         36928
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 3, 6, 64)          0
_________________________________________________________________
flatten (Flatten)            (None, 1152)              0
_________________________________________________________________
dropout_1 (Dropout)          (None, 1152)              0
_________________________________________________________________
dense (Dense)                (None, 256)               295168
_________________________________________________________________
dense_1 (Dense)              (None, 100)               25700
_________________________________________________________________
batch_normalization (BatchNo (None, 100)               400
_________________________________________________________________
dense_2 (Dense)              (None, 50)                5050
_________________________________________________________________
dense_3 (Dense)              (None, 10)                510
_________________________________________________________________
dense_4 (Dense)              (None, 2)                 22
=================================================================
Total params: 454,210
Trainable params: 454,010
Non-trainable params: 200
```

The Sweet & Simple Model Predictions on Test Data

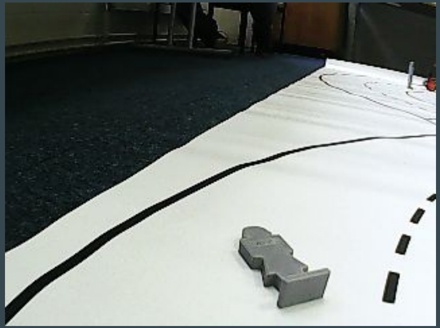
A = 0.92   S = 0.99



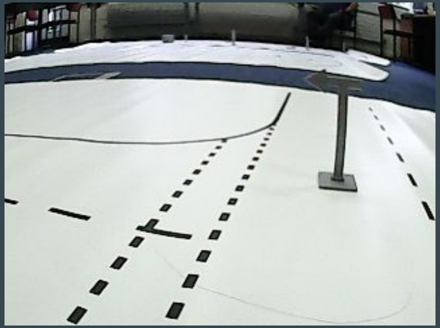
A = 0.45   S = 0.99



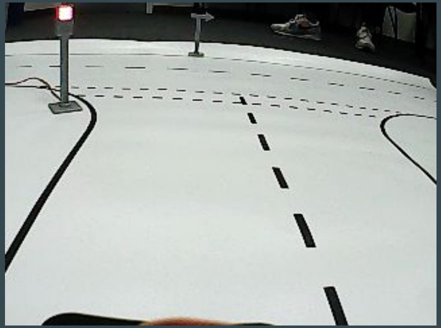
A = 0.61   S = 0



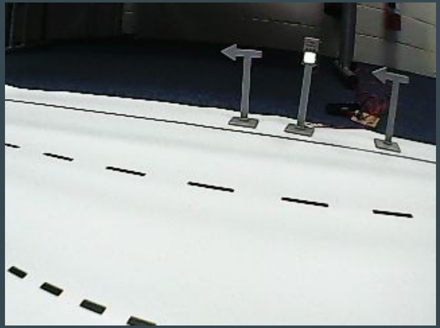
A = 0.03   S = 0.99



A = 0.48   S = 0



A = 0.18   S = 0.99



Thank you for listening

...