Chalmers University of Technology

Project - Part 3

Road Detection

Spatial Statistics and Image Analysis

Sahana Maddali

June 8, 2021

Abstract

Road detection is one of the many tasks for autonomous vehicles. The KITTI Vision Benchmark Suite provides several data-sets for Computer Vision, of which the road and lane detection benchmark contains annotated images of four categories of lanes and roads : "urban unmarked", "urban marked", "urban multiple marked lanes" and "urban", the last of which is a combination of the other three. In this report, we use the technique of Neural Decomposition to train a neural network for prediction of the region of the image that is a road or a lane. A statistical analysis of the models is conducted and metrics are reported.

1 Introduction

1.1 Data

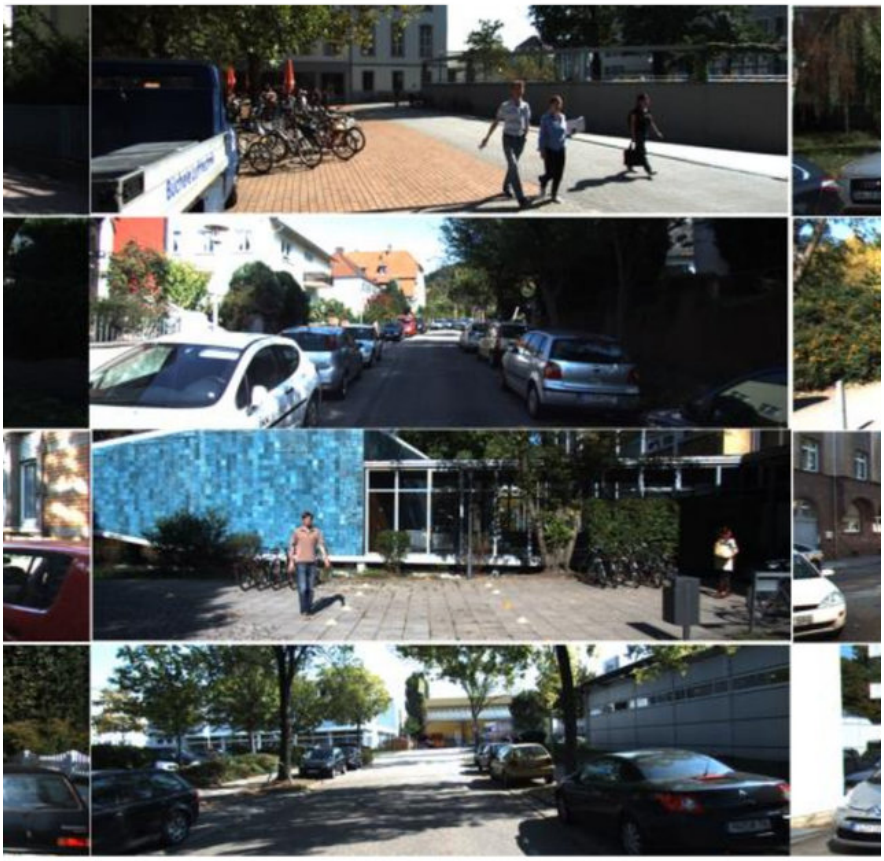

Figure 1: Collage of images from data-set.

A popular dataset for road detection tasks is the KITTI Vision Benchmark Suite `http://cvlibs.net/datasets/kitti/eval_road.php`. The raw data recordings, sorted by category include sequences/videos of a few frames per data point of roads seen during a drive. These have have 3D object labels.

An "Ego-lane" is the lane on which the vehicle is driving on/intends to drive on. Modern vehicles come equipped with technology that can detect marked lanes. The road

and lane benchmark from the KITTI Vision Benchmark Suite provides an annotated data-set of how to detect the ego-lane.

Figure 2: Capture with highlighted Ego-lane.

The output feature list is [P0, P1, P2 ,P3, R0_rect, Tr_velo_to_cam, Tr_imu_to_velo, Tr_cam_to_road]. The first four arguments, namely the four points of a parallelogram enclosing the road (the green area in Figure 2) are P0, P1, P2, P3. The argument R0_rect is the rectifying rotation for reference coordinate (rectification makes images of multiple cameras lie on the same plan). The arguments at the end are several transformations between different cameras.

The training set has 289 images and labels and the test set has 290 images and labels.

1.2 Spectral Decomposition

In [GG17], a technique called Neural Decomposition is introduced. In this technique, a simple network of neurons is used to generalise the time series data with the use a periodic activation functions. A number of neurons, or *units* as referred to in [GG17], are activated with sinusoidal activation function to capture periodicity / seasonality from the data, and other units are activated by non-periodic activation functions to capture trend and other nonlinear components in the data. The following equation gives the mathematical model of this simple neural network :

$$x(t) = \sum_{n=1}^{k} [a_k \cdot sin(w_k t + \phi_k)] + g(t) \tag{1}$$

In short, the amplitude a_k, frequency w_k and phase-shift ϕ_k are defined as learnable parameters in the neural network along with an augmentation function $g(t)$ that models other nonlinear components, which could also be set to 0. The model $x(t)$ is basically a weighted sum of sinusoids that is learned during training of the network. The aim is to achieve a Fourier-like decomposition of the input signal using a neural network, i.e. to *learn the sinusoids* that describe the periodicity in the data fully. The use of the sine function as an activation function has been widely discussed and is known to be quiet problematic during implementation. However many experiments, including ones conducted by [GG17] and [SMB+20] have shown how the periodicity of the sine function can be leveraged to improve machine learning tasks. Both [GG17] and [SMB+20] propose some of many schemes for careful weight initialization and regularization of the network, that help the learned sine function generalize periodicity in the data well.

1.3 Metrics

We use the following metrics to report results. All metrics are calculated using the sklearn package in python.

- *explained_ variance_ score* : It explains the part of the modelâs total variance that is explained by factors that are actually present and is not due to error variance. The best possible score is 1.0, lower values are show worse performance.
- *mean_ absolute_ error* : It gives the absolute averaged difference between predicted and true values. The closer this score is to 0, the better.
- *r2_ score* : This is the coefficient of determination, denoted r^2 and pronounced "R squared". It is the proportion of the variance in the dependent variable that is predictable from the independent variable(s). The score is usually between 0 and 1, where 1 is the best score.

1.4 Methodology

The methodology in this report is as follows :

- Download pre-defined dataset.
- Generate input features for a neural network by reshaping the images.
- Design a neural network to train on the data.
- Evaluate metrics
- Report metrics

1.5 Neural Networks

Similar to the implementation in [SMB$^+$20], the neural network illustrated in Figure 3 is proposed. The augmentation function is set to 0.

The inputs to the network are grayscale and normalines flattened images. Eacch image is a vector of length 466616. The outputs of the network are flattened information from the calibration files, i.e. vectors of length 96. The model is written in python, and the Functional API from Keras is used to build the model.

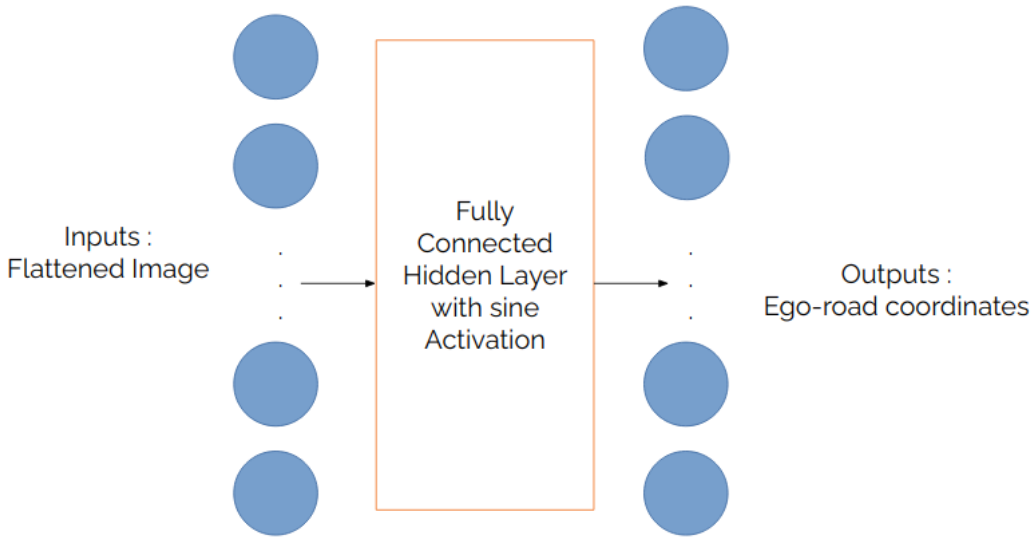

Figure 3: Proposed Neural Network

2 Proposed Models

The following gives an example of how the models are built using python and Keras. Definitions 1 and 2 give the parameters of the models.

```
# Define the layers
# Input branch
branch_ip = Input(shape=(ip_size,))
# Dense layer of n = 32 units, sine activation function and a
# pre-determined kernel initialization function.
# 5 such layers are defined.
branch1 = Dense(32, activation = tensorflow.sin, kernel_initializer=
    initializer)(branch_ip)
branch2 = Dense(32, activation = tensorflow.sin, kernel_initializer=
    initializer)(branch1)
branch3 = Dense(32, activation = tensorflow.sin, kernel_initializer=
    initializer)(branch2)
branch4 = Dense(32, activation = tensorflow.sin, kernel_initializer=
    initializer)(branch3)
branch5 = Dense(32, activation = tensorflow.sin, kernel_initializer=
    initializer)(branch4)
# Output layer is Dense of 96 units and ReLU activation
branch_op = Dense(96, activation = 'relu', kernel_initializer=
    initializer)(branch5)

# Define the model
model = Model(inputs=branch_ip, outputs=branch_op)
# Print model structure
model.summary()
```

Listing 1: Example model definition

Feature	*Value*
Input Layer	*shape = (466616,)*
Hidden Layers	*5 Dense(64) Layers*
Hidden Activation	*Sine*
Output Layer	*Dense(96) Layer*
Output Activation	*ReLu*
Trainable Parameters	*29,886,368*
Optimizer	*Adam(0.001)*
Loss	*Mean Absolute Error*
Epochs	*25*
Epoch metrics	*Mean loss, MAE accuracy*

Definition 1 (Model 1)**.**

Feature	*Value*
Input Layer	*shape = (466616,)*
Hidden Layers	*5 Dense(32) Layers*
Hidden Activation	*Sine*
Output Layer	*Dense(96) Layer*
Output Activation	*ReLu*
Trainable Parameters	*14,939,136*
Optimizer	*Adam(0.01)*
Loss	*Mean Absolute Error*
Epochs	*25*
Epoch metrics	*Mean loss, MAE accuracy*

Definition 2 (Model 2)**.**

3 Results

3.1 Reported metrics during training

During training, the mean of the loss, labeled "Loss", and the mean absolute error, labeled "MAE" are reported. The model minimises the mean absolute error, i.e. the loss function is the mean absolute error.

```
Epoch 000: Loss: 0.358, MAE: 35.793%
Epoch 005: Loss: 0.352, MAE: 35.239%
Epoch 010: Loss: 0.350, MAE: 34.950%
Epoch 015: Loss: 0.350, MAE: 34.950%
Epoch 020: Loss: 0.349, MAE: 34.949%
Execution time: 7566.314916265997
CPU times: user 5h 31min 28s, sys: 41min 11s, total: 6h 12min 40s
Wall time: 2h 6min 15s
```

Figure 4: Training of Model 1. It takes 2 h 6 mins to train the network.

```
Epoch 000: Loss: 0.348, MAE: 34.837%
Epoch 005: Loss: 0.554, MAE: 55.429%
Epoch 010: Loss: 0.348, MAE: 34.814%
Epoch 015: Loss: 0.345, MAE: 34.505%
Epoch 020: Loss: 0.349, MAE: 34.921%
Execution time: 4646.180346936
CPU times: user 3h 17min 37s, sys: 35min 24s, total: 3h 53min 1s
Wall time: 1h 17min 40s
```

Figure 5: Training of Model 2. It takes 1 h 17 mins to train the network.

Both models are comparable when it comes to minimising the loss, although Model 2 performs just slightly better.

3.2 Reported metrics after training

The following table shows the metrics of the model based on the prediction on test set. Both models show that the explained variance score is closer to 0 than to 1 implying that the model's total variance is not explained very well by actual factors. The model was optimised for mean absolute error, and both models show a low score. The R^2 error is also closer to 0 in both models, which is not a great score.

Model	EVS	MAE	R^2
Model 1	0.39	0.33	0.39
Model 2	0.25	0.35	0.25

Table 4: Reported metrics after training of models.

The low mean absolute error score shows that the prediction error is < 0.5 units of distance. However, the unit of distance is quiet large, and this result could be relatively

poor. We need to take a closer look at the MAE. The results is shown in the table below. The reported values are the means of absolute error on each variable, i.e. an average of absolute difference between 96 predicted values an 96 true values for each image in the training set.

Model	Estimate	Values
Model 1	Mean of MAE per image	[85.74, 0.0056, 0.0056, . . . , 0.0056]
Model 2	Mean of MAE per image	[85.74, 0.0, 0.0, . . . , 0.0]

Table 6: Average MAE for each image in training set.

Here we see that the error on the prediction for the fist image in the test set is the highest, i.e. 85.73 for Model 1 and 85.74 for Model 2. The rest of the images have an error of around 0.0. So in Table 4, the reported errors are the error on prediction of the first image divided by the length of the test set, which is 290. So in fact, the reported metric on the MAE is a very good value, and the network performs very well on road detection. In fact the smaller network (Model 2) performs far better, detecting ego-lane coordinates with zero error.

Figure 6: An image from test set - um_000082.png.

Figure 7: Polygon (in bright green) determined by detected points $P0, P1, P2$ and $P3$ using Model 2.

4 Conclusion

The method of neural decomposition was tested in road and lane detection tasks. The networks performed excellently on detecting road edge coordinates and corresponding camera transformations. Of the two models that were tested, both predicted the pixel values of the ego-road with a Mean Absolute Error of 0.0 for 289/290 images in the test set. The success of neural decomposition was tested and confirmed for road detection task on the KITTI Road Benchmark data-set.

References

[GG17] Godfrey, Luke B. ; Gashler, Michael S.: Neural Decomposition of Time-Series Data for Effective Generalization. In: *IEEE Transactions on Neural Networks and Learning Systems* (2017), 1â13. `http://dx.doi.org/10.1109/tnnls.2017.2709324`. – DOI 10.1109/tnnls.2017.2709324. – ISSN 2162–2388

[SMB+20] Sitzmann, Vincent ; Martel, Julien N. ; Bergman, Alexander W. ; Lindell, David B. ; Wetzstein, Gordon: Implicit Neural Representations with Periodic Activation Functions. In: *Proc. NeurIPS*, 2020