

# Data Structure and a Naive Model

Zeyi Wang

September 20, 2016

## 0. Quantitative question

- Can we design a self-driving car?
- Can we predict the appropriate acceleration and steering angle based on data collected by the camera and GPS, such as speed, acceleration, steering angle and camera images?

## 1. Data Structure

45 GB compressed, 80 GB uncompressed (unzip it in Windows 10, not Mac OS)

```
list.files(file.path("comma-dataset"), recursive = T)
```

```
## [1] "camera/2016-01-30--11-24-51.h5" "camera/2016-01-30--13-46-00.h5"
## [3] "camera/2016-01-31--19-19-25.h5" "camera/2016-02-02--10-16-58.h5"
## [5] "camera/2016-02-08--14-56-28.h5" "camera/2016-02-11--21-32-47.h5"
## [7] "camera/2016-03-29--10-50-20.h5" "camera/2016-04-21--14-48-08.h5"
## [9] "camera/2016-05-12--22-20-00.h5" "camera/2016-06-02--21-39-29.h5"
## [11] "camera/2016-06-08--11-46-01.h5" "LICENSE"
## [13] "log/2016-01-30--11-24-51.h5" "log/2016-01-30--13-46-00.h5"
## [15] "log/2016-01-31--19-19-25.h5" "log/2016-02-02--10-16-58.h5"
## [17] "log/2016-02-08--14-56-28.h5" "log/2016-02-11--21-32-47.h5"
## [19] "log/2016-03-29--10-50-20.h5" "log/2016-04-21--14-48-08.h5"
## [21] "log/2016-05-12--22-20-00.h5" "log/2016-06-02--21-39-29.h5"
## [23] "log/2016-06-08--11-46-01.h5"
```

We will need 'h5' package to extract data from \*.h5 files.

```
library("h5")
log <- H5File(file.path("comma-dataset", "log", "2016-06-08--11-46-01.h5"))
```

```
## Warning in H5File(file.path("comma-dataset", "log",
## "2016-06-08--11-46-01.h5")): This function is deprecated, use h5file
## instead
```

```
image <- H5File(file.path("comma-dataset", "camera", "2016-06-08--11-46-01.h5"))
```

```
## Warning in H5File(file.path("comma-dataset", "camera",
## "2016-06-08--11-46-01.h5")): This function is deprecated, use h5file
## instead
```

```
log_names <- list.datasets(log, recursive = TRUE)
image_names <- list.datasets(image, recursive = TRUE)
```

So the function insists that I am deprecating it. Well.

## 1.1 Image Data

The image part are pixel images, 20 pic per sec, 320\*160 pixels per pic, with RGB format.

```
image[image_names] #check the image we loaded
```

```
## DataSet 'X' (18177 x 3 x 160 x 320)
## type: integer
## chunksize: 1024 x 3 x 160 x 320
## maxdim: 18177 x 3 x 160 x 320
```

So the dimension is **Frames \* RGB \* Columns \* Rows**.

Image data are basically **4 d arrays**.

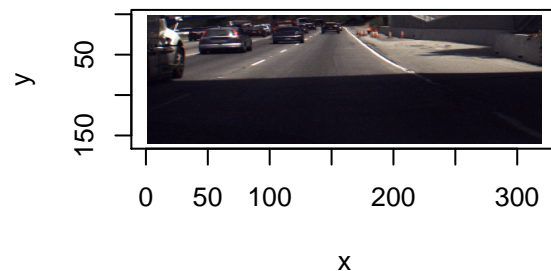
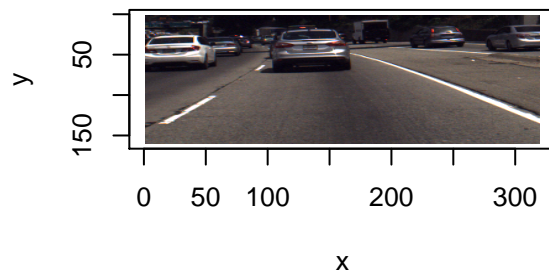
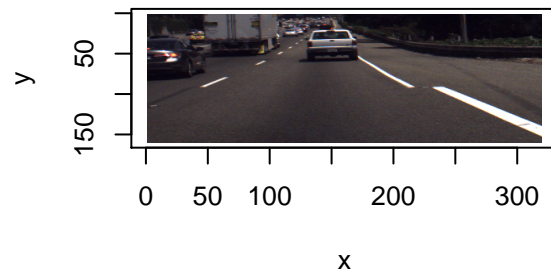
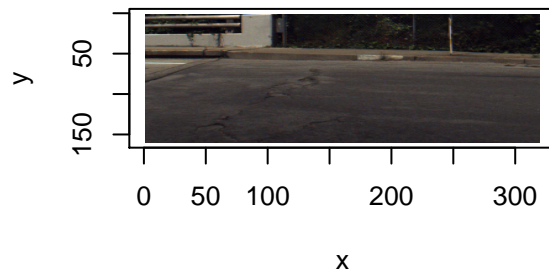
```
range(image[image_names][1,,,])
```

```
## [1] 20 255
```

The values coincide with RGB format.

Actually why don't we check some of the images:

```
library(imager)
par(mfrow = c(2, 2))
plot(as.cimg(aperm(image[image_names][10001,,,], c(4,3,1,2)))) #the start of training set
plot(as.cimg(aperm(image[image_names][13001,,,], c(4,3,1,2)))) #the end of training set
plot(as.cimg(aperm(image[image_names][14002,,,], c(4,3,1,2)))) #the start of test set
plot(as.cimg(aperm(image[image_names][15002,,,], c(4,3,1,2)))) #the end of test set
```



```
par(mfrow = c(1, 1))
```

Difficulties to overcome:

Sometimes the roads are with lines while sometimes they don't;

It could be a truck but it also could be the shadow (cars in the same direction do stay longer though).

## 1.2 Log Data

reference: [link to comma.ai github](#)

The log files are with the same names, but under the “./comma-dataset/log/” folder.

Let's check the first hierarchy of the log:

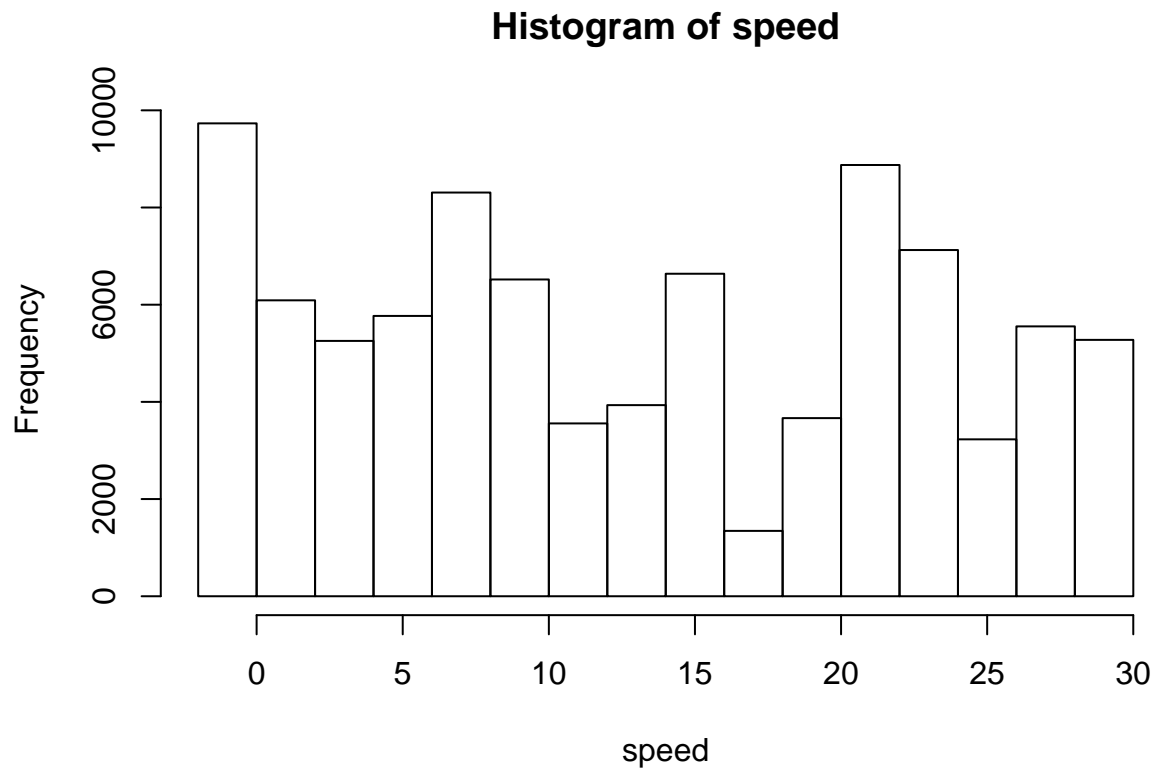
```
log_names
```

```
## [1] "/UN_D_cam1_ptr"      "/UN_D_lidar_ptr"    "/UN_D_radar_msg"
## [4] "/UN_D_rawgps"        "/UN_T_cam1_ptr"     "/UN_T_lidar_ptr"
## [7] "/UN_T_radar_msg"     "/UN_T_rawgps"       "/blinker"
## [10] "/brake"              "/brake_computer"    "/brake_user"
## [13] "/cam1_ptr"           "/car_accel"         "/fiber_accel"
## [16] "/fiber_compass"      "/fiber_compass_x"   "/fiber_compass_y"
## [19] "/fiber_compass_z"    "/fiber_gyro"        "/fiber_temperature"
## [22] "/gas"                "/gear_choice"       "/idx"
## [25] "/rpm"                "/rpm_post_torque"   "/selfdrive"
## [28] "/speed"              "/speed_abs"         "/speed_fl"
## [31] "/speed_fr"           "/speed_rl"          "/speed_rr"
## [34] "/standstill"         "/steering_angle"    "/steering_torque"
## [37] "/times"              "/velodyne_gps"      "/velodyne_heading"
## [40] "/velodyne_imu"
```

There are speed[28], acceleration[14], steering angle[35], steering torque[36].

Check the speed first:

```
temp <- log[log_names[28]]@dim
speed <- log[log_names[28]][1:temp]
rm(temp)
hist(speed)
```



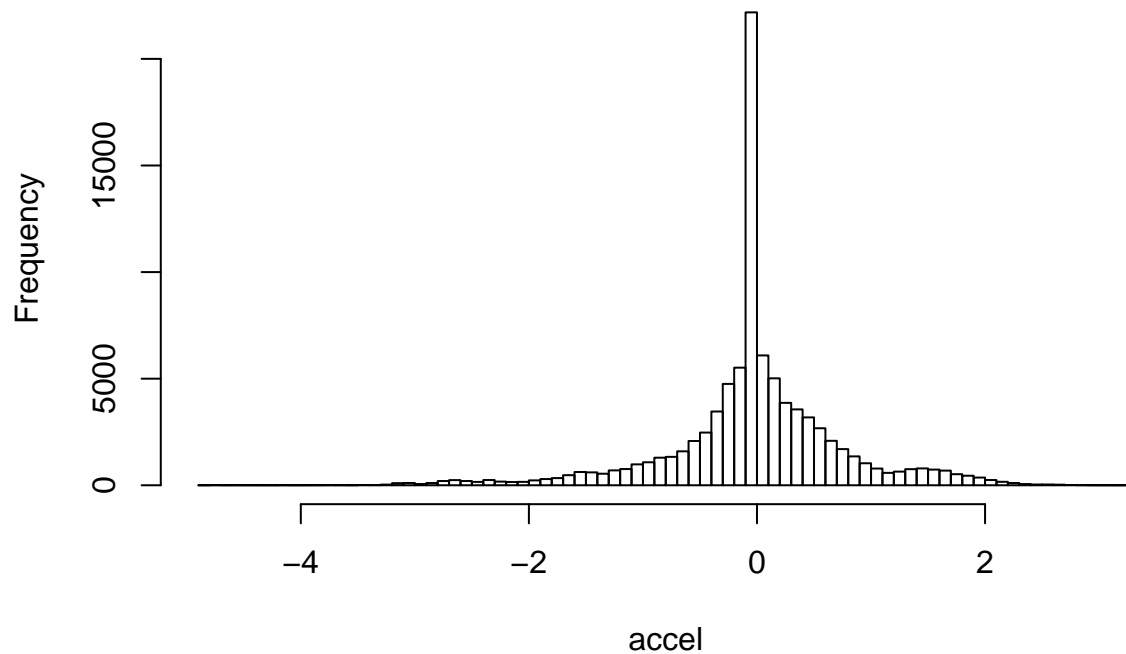
Acceleration:

```
temp <- log[log_names[14]]@dim
accel <- log[log_names[14]][1:temp]
rm(temp)
range(accel)
```

```
## [1] -4.844243  3.290349
```

```
hist(accel, breaks = 100)
```

## Histogram of accel



Most of the time, the car keeps its speed;

```
sum(accel > 0.8)
```

```
## [1] 9393
```

```
sum(accel < -0.8)
```

```
## [1] 9919
```

```
length(accel)
```

```
## [1] 90870
```

```
accel_tri <- 1 * (accel > 0.8) - 1 * (accel < - 0.8) #this is the "answer" for supervised learning
```

It would be great if we can first of all predict accelerating/slowing down/keeping speed status.

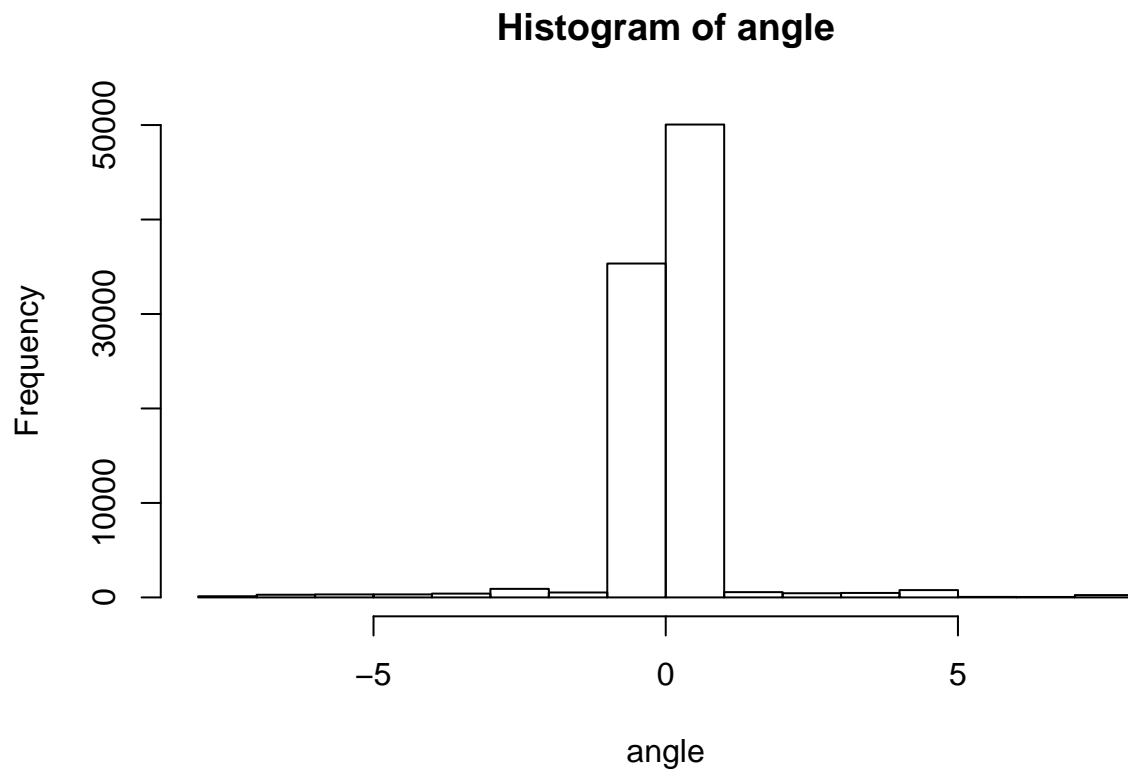
**Steering:**

```
library(scales)
```

```
temp <- log[log_names[35]]@dim  
angle <- log[log_names[35]][1:temp]  
rm(temp)  
angle <- scale(angle)
```

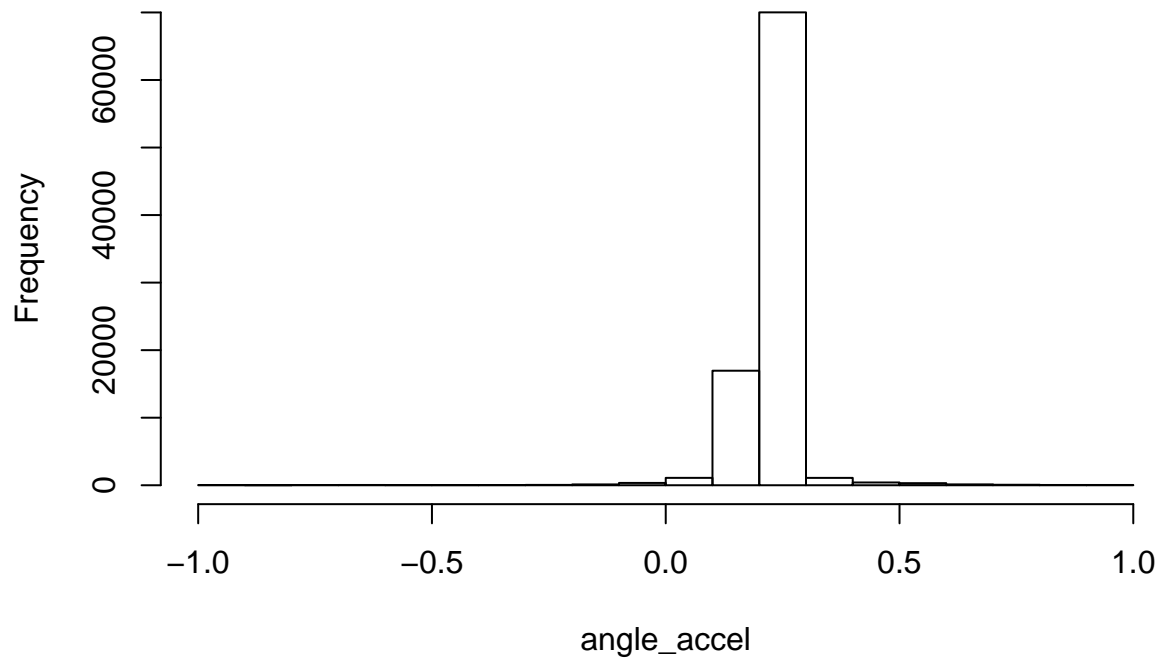
```
temp <- log[log_names[36]]@dim
angle_accel <- log[log_names[36]][1:temp]
rm(temp)
angle_accel <- rescale(angle_accel, to = c(-1, 1))

hist(angle)
```



```
hist(angle_accel)
```

## Histogram of angle\_accel



It would help the training that we scale the variables.

### 1.3 Timeline

The “/cam1\_ptr” in log file records the timeline.

```
temp <- log[log_names[13]]@dim
timeline_image <- log[log_names[13]][1:temp]
rm(temp)
length(timeline_image)
```

```
## [1] 90870
```

```
range(timeline_image)
```

```
## [1] 0 18176
```

Simple calculation:

There are 90870 time points in log, 18176 time points in image;

100Hz for log; 20Hz for image;

$90870/100 = 18176/20 = 908.8 \text{ sec} = 15.1 \text{ min}$ ;

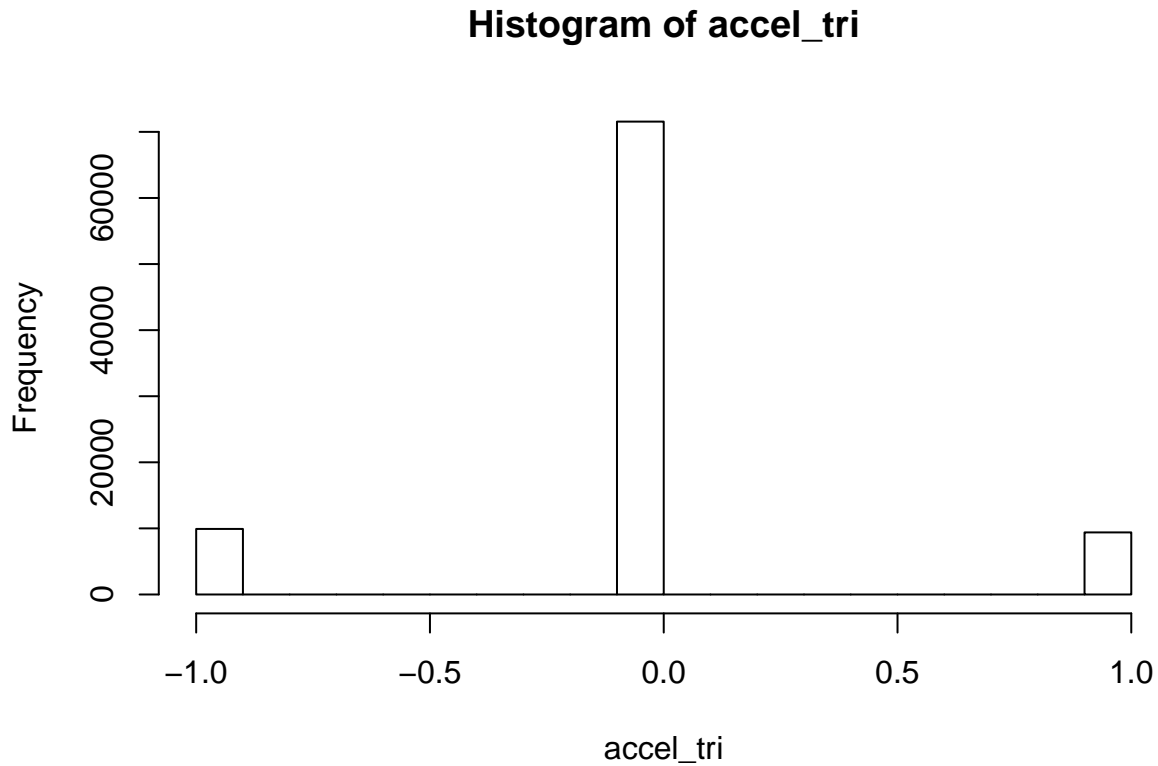
It is the shortest driving set.

## 2. A Naive Model

We will predict acceleration status, but we won't use image for now.

So it is a helpful warmup for convolutional network training.

```
hist(accel_tri)
```



The naive model:

**Acceleration status ~ 10sec speed + 10sec steering angle + 10sec angle acceleration**

### 2.1 Training, Validation and Testing set

```
set.seed(160920)
index_train <- sample(50001:65000, size = 10000)
index_validation <- setdiff(50001:65000, index_train)
index_test <- 70001:75000

set_train <- matrix(0, 10000, 31)
set_train[,1] <- as.matrix(accel_tri[index_train])
for (i in 1:10) {
  set_train[,i + 1] <- speed[index_train-i*100]
  set_train[,i + 11] <- angle[index_train-i*100]
  set_train[,i + 21] <- angle_accel[index_train-i*100]
}
```



```

set_validation <- matrix(0, 5000, 31)
set_validation[,1] <- as.matrix(accel_tri[index_validation])
for (i in 1:10) {
  set_validation[,i + 1] <- speed[index_validation-i*100]
  set_validation[,i + 11] <- angle[index_validation-i*100]
  set_validation[,i + 21] <- angle_accel[index_validation-i*100]
}

set_test <- matrix(0, 5000, 31)
set_test[,1] <- as.matrix(accel_tri[index_test])
for (i in 1:10) {
  set_test[,i + 1] <- speed[index_test-i*100]
  set_test[,i + 11] <- angle[index_test-i*100]
  set_test[,i + 21] <- angle_accel[index_test-i*100]
}

```

## 2.2 Fitting with random forests

```

library(randomForest)
model_naive <- randomForest(set_train[,2:31], as.factor(set_train[,1]))

sum(predict(model_naive, set_validation[,2:31]) == set_validation[,1])/5000 #validation

```

```
## [1] 0.9926
```

```
sum(predict(model_naive, set_test[,2:31]) == set_test[,1])/5000 #test
```

```
## [1] 0.6182
```

The validation set is contaminated.

```
sum(predict(model_naive, set_test[set_test[,1] == 0, 2:31]) == set_test[set_test[,1] == 0, 1])/sum(set_test[set_test[,1] == 0, 1])
```

```
## [1] 0.8944928
```

```
sum(predict(model_naive, set_test[set_test[,1] == 1, 2:31]) == set_test[set_test[,1] == 1, 1])/sum(set_test[set_test[,1] == 1, 1])
```

```
## [1] 0.006711409
```

```
sum(predict(model_naive, set_test[set_test[,1] == -1, 2:31]) == set_test[set_test[,1] == -1, 1])/sum(set_test[set_test[,1] == -1, 1])
```

```
## [1] 0
```

So we can see that the image data is needed.

### 3. Plan this week

The project is difficult in two meanings:

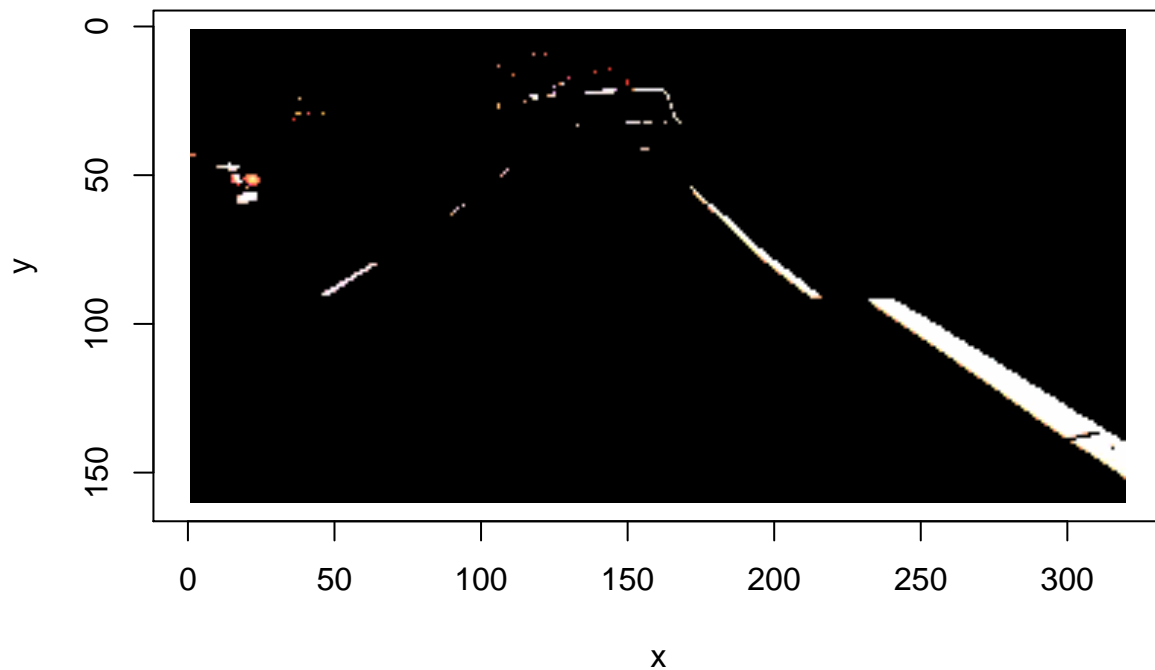
- We need to deal with image data;
- Convolutional learning is always difficult.

In addition, we will need to predict acceleration and steering angle in the end.

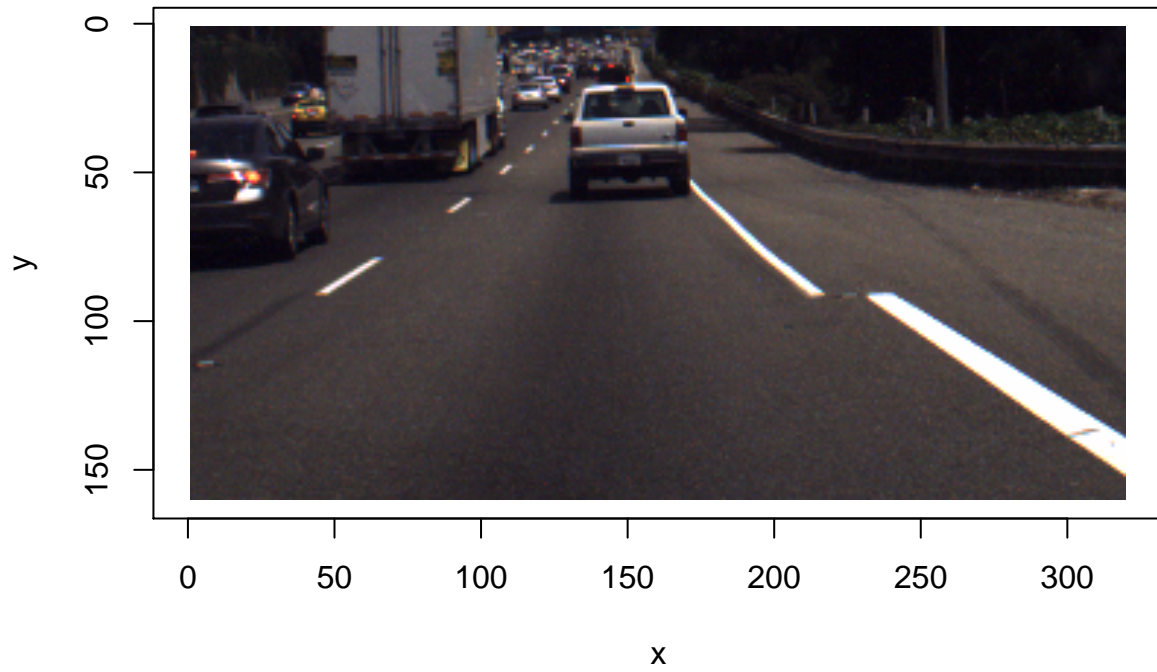
#### Tasks:

- Example codes for angle prediction; visualizing the results
- Simple ideas for image data: stamping out by grid regions; outlier detection
- Papers regarding convolutional learning
- A slightly improved model with images involved

```
M <- aperm(image[image_names][13001,,,], c(4,3,1,2))
MM <- M[, , 1, 1] >= 235
MMM <- M
for (i in 1:3) {
  MMM[, , 1, i] <- MM * M[, , 1, i]
}
plot(as.cimg(MMM))
```



```
plot(as.cimg(M))
```



- Angle prediction could be easier
- Any data on distance to the car in front?