Exercise 2: End-to-End Self-Driving via Convolutional Neural Networks

In [ ]:

```
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list all files u

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/) that gets preserved
# You can also write temporary files to /kaggle/temp/, but they won't be saved outside of t
```

Importing neccessary Libraries

In [96]:

```
import os
import cv2
import torch
import torch.nn as nn
from torch.utils.data import DataLoader,Dataset
import torchvision.transforms as transforms
from PIL import Image
import pandas as pd
```

The steering turning radius is stored as pandas dataframe

In [80]:

```
target = pd.read_csv("/kaggle/input/car-steering-angle-prediction/driving_dataset/angles.tx
```

There are roughly 45000 images. We will keep first 25000 for training, next 10000 for validation and last 10000 for testing our model.

In [117]:

```
class Train_Dataset(Dataset):#Inherits from torch.utils.data.Dataset
    def __init__(self):
        #default directory where data is loaded
        self.filepath = '../input/car-steering-angle-prediction/driving_dataset/'
#       From the directory stores first 25000 images for training
        self.filenames = [self.filepath + target[0][i] for i in range(25000)]

    def __len__(self):
        return len(self.filenames)

    def __getitem__(self,index):
        filename = self.filenames[index]
        img = cv2.imread(filename)
        #Resizing images to(66,200)
        resized = cv2.resize(img,(66,200),interpolation = cv2.INTER_AREA)
        #normalizing image
        transform = transforms.Compose([transforms.ToTensor()])
        transf_img = transform(resized)
        mean, std = transf_img.mean([1,2]), transf_img.std([1,2])
        transform_norm = transforms.Compose([transforms.ToTensor(),transforms.Normalize(mea
        norm_img = transform_norm(resized)
        #return the image converted to a numpy array and its corresponding steering angle
        return norm_img.float(),target[1][index]
```

In [118]:

```
class ConvNet(torch.nn.Module):
    def __init__(self):
        super().__init__()
        # first three convolution layer with 5*5 kernal and stride = 2
        self.conv1 = nn.Conv2d(3, 24, 5, stride = (2,2))
        self.conv2 = nn.Conv2d(24, 36, 5, stride = (2,2))
        self.conv3 = nn.Conv2d(36, 48, 5, stride = (2,2))
        # Next two convolution layer with 3*3 kernal and stride = 1
        self.conv4 = nn.Conv2d(48, 64, 3, stride = (1,1))
        self.conv5 = nn.Conv2d(64, 64, 3, stride = (1,1))
        # flatten the image to pass
        self.flat = nn.Linear(1152, 1164)
        self.lin1 = nn.Linear(1164,100)
        self.lin2 = nn.Linear(100, 50)
        self.lin3 = nn.Linear(50, 10)
        self.out = nn.Linear(10, 1)

    def forward(self,x):
        x = self.conv1(x)
        x = self.conv2(x)
        x = self.conv3(x)
        x = self.conv4(x)
        x = self.conv5(x)
        x = x.view(x.shape[0], -1)
        x = self.flat(x)
        x = self.lin1(x)
        x = self.lin2(x)
        x = self.lin3(x)
        x = self.out(x)
        return x
```

In [119]:

```
train_data = Train_Dataset()
train_loader = torch.utils.data.DataLoader(train_data,batch_size = 60, shuffle = True)
net = ConvNet()
optimizer = torch.optim.Adam(net.parameters(),lr = 1e-3)
criterion = torch.nn.MSELoss()               # Mean squared loss is used
```

In [120]:

```
for epoch in range(4):
    avg_train_rmse = []
    for i,sample_batched in enumerate(train_loader):
        optimizer.zero_grad()
        yhat = net(sample_batched[0])
        loss = criterion(yhat.squeeze(),sample_batched[1].type(torch.float))
        avg_train_rmse.append(loss)
        loss.backward()
        optimizer.step()
    print(f'Avg RMSE after epoch {epoch} is {(sum(avg_train_rmse)**(0.5))/len(avg_train_rms
```

Avg RMSE after epoch 0 is 1.5243878364562988

Avg RMSE after epoch 1 is 1.499646782875061

Avg RMSE after epoch 2 is 1.4494885206222534

Avg RMSE after epoch 3 is 1.387422800064087

For the Validation Set

In [121]:

```
class Val_Dataset(Dataset):#Inherits from torch.utils.data.Dataset
    def __init__(self):
        #default directory where data is loaded
        self.filepath = '../input/car-steering-angle-prediction/driving_dataset/'
#         self.filenames = os.listdir(self.filepath)
        self.filenames = [self.filepath + target[0][i] for i in range(25000, 35000)]

    def __len__(self):
        return len(self.filenames)

    def __getitem__(self,index):
        filename = self.filenames[index]
        img = cv2.imread(filename)
        #Resizing images to(66,200)
        resized = cv2.resize(img,(66,200),interpolation = cv2.INTER_AREA)
        #normalizing image
        transform = transforms.Compose([transforms.ToTensor()])
        transf_img = transform(resized)
        mean, std = transf_img.mean([1,2]), transf_img.std([1,2])
        transform_norm = transforms.Compose([transforms.ToTensor(),transforms.Normalize(mea
        norm_img = transform_norm(resized)
        #return the image converted to a numpy array and its corresponding steering angle
        return norm_img.float(),target[1][index]
```

In [122]:

```
val_data = Val_Dataset()
val_loader = torch.utils.data.DataLoader(val_data,batch_size = 60, shuffle = True)

avg_val_rmse = []
for i,sample_batched in enumerate(val_loader):
    yhat_val = net(sample_batched[0])
    loss_val = criterion(yhat_val.squeeze(),sample_batched[1].type(torch.float))
    avg_val_rmse.append(loss_val)
print(f'Avg RMSE on validation set is {(sum(avg_val_rmse)**(0.5))/len(avg_val_rmse)}')
```

Avg RMSE on validation set is 5.460453033447266

Test data

In [123]:

```
class Test_Dataset(Dataset):#Inherits from torch.utils.data.Dataset
    def __init__(self):
        #default directory where data is loaded
        self.filepath = '../input/car-steering-angle-prediction/driving_dataset/'
#         self.filenames = os.listdir(self.filepath)
        self.filenames = [self.filepath + target[0][i] for i in range(35000, 45406)]

    def __len__(self):
        return len(self.filenames)

    def __getitem__(self,index):
        filename = self.filenames[index]
        img = cv2.imread(filename)
        #Resizing images to(66,200)
        resized = cv2.resize(img,(66,200),interpolation = cv2.INTER_AREA)
        #normalizing image
        transform = transforms.Compose([transforms.ToTensor()])
        transf_img = transform(resized)
        mean, std = transf_img.mean([1,2]), transf_img.std([1,2])
        transform_norm = transforms.Compose([transforms.ToTensor(),transforms.Normalize(mea
        norm_img = transform_norm(resized)
        #return the image converted to a numpy array and its corresponding steering angle
        return norm_img.float(),target[1][index]
```

In [124]:

```
test_data = Test_Dataset()
test_loader = torch.utils.data.DataLoader(test_data,batch_size = 60, shuffle = True)

avg_test_rmse = []
for i,sample_batched in enumerate(test_loader):
    yhat_test = net(sample_batched[0])
    loss_test = criterion(yhat_test.squeeze(),sample_batched[1].type(torch.float))
    avg_test_rmse.append(loss_test)
print(f'Avg RMSE on Test set is {(sum(avg_test_rmse)**(0.5))/len(avg_test_rmse)}')
```

Avg RMSE on Test set is 4.780969142913818

For normalization Link used: https://www.geeksforgeeks.org/how-to-normalize-images-in-pytorch/ (https://www.geeksforgeeks.org/how-to-normalize-images-in-pytorch/)