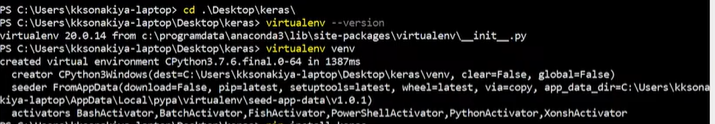
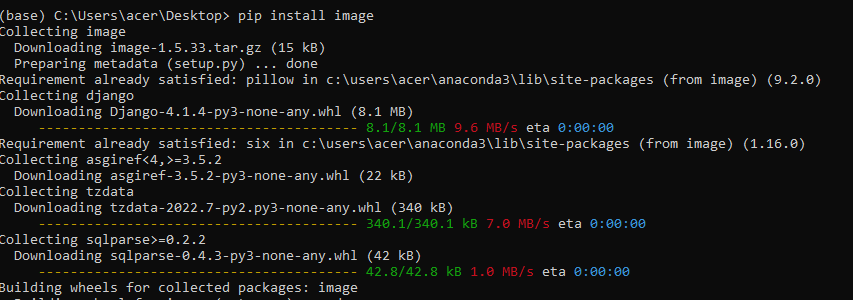
**6th Report**

This session and the next two will be devoted to writing the code and testing it on Udacity. We started by installing Anaconda, which is a software containing all the modules we will need to use. We started by using Virtualenv in our Shell, which is a tool for creating isolated Python virtual environments. virtualenv creates a folder that contains all the executables needed to use the packages that a Python project might require.



Then I installed my environment and created my dependencies, which are the following:

Tensorflow, matplotlib, numpy, Image, opencv-python, sklearn, jupyter, scipy, flask-socketio.



I train the model using a convolutional neural network predicting one linear output. As I said above, the architecture is built in the build\_method method. In order to reduce the overfitting, I used data augmentation by flipping each image of the dataset and by using the left/right camera. I also add dropout just after the last convolution and after the first Fully connected layer. The model used an Adam optimizer, so the learning rate was not tuned manually. Then, for the architecture, the size and the number of filters have been set by many trial and error. For the training data, I used a combination of the central, left and right camera, each one, randomly flip. The training data is a subset of 85% of the total dataset.

THE MODEL.PY :

|  |
| --- |
|  |
|  | import cv2  import csv |
|  | import os |
|  | import numpy as np |
|  | import random |
|  | from random import shuffle |
|  |  |
|  | from sklearn.model\_selection import train\_test\_split |
|  |  |
|  | from tensorflow.python.keras.models import Model |
|  | from tensorflow.python.keras.models import Sequential |
|  | from tensorflow.python.keras.layers import Conv2D, Dense, Flatten, Cropping2D, Lambda, Dropout |
|  |  |
|  | DATA\_PATH = "data/driving\_log.csv" |
|  | DATA\_IMG = "data/" |
|  |  |
|  | def build\_model(): |
|  | """ |
|  | Build keras model |
|  | """ |
|  | model = Sequential() |
|  | model.add(Lambda(lambda x: (x / 127.5) - 1., input\_shape = (160, 320, 3))) |
|  | model.add(Cropping2D(cropping=((70, 25), (0, 0)), input\_shape = (160, 320, |
|  | model.add(Conv2D(8, 9, strides=(4, 4), padding="same", activation="elu")) |
|  | model.add(Conv2D(16, 5, strides=(2, 2), padding="same", activation="elu")) |
|  | model.add(Conv2D(32, 4, strides=(1, 1), padding="same", activation="elu")) |
|  | model.add(Flatten()) |
|  | model.add(Dropout(.6)) |
|  | model.add(Dense(1024, activation="elu")) |
|  | model.add(Dropout(.3)) |
|  | model.add(Dense(1)) |
|  |  |
|  |
|  | model.compile(loss="mse", optimizer="adam") |
|  |  |
|  | return model |
|  |  |
|  | def get\_data(log\_content, index\_list, batch\_size, strict=True): |
|  | """ |
|  | Return Data from the simulator |
|  | \*\*input: |
|  | \*log\_content (2dim Array) Data from the log file |
|  | \*index\_list: Index to used to create each batch |
|  | \*batch\_size (Int) Size of each batch |
|  | """ |
|  | images, rotations = [], [] |
|  | while True: |
|  | if not strict: |
|  | shuffle(index\_list) |
|  | for index in index\_list: |
|  | # Futur angle correction |
|  | angle\_correction = [0., 0.25, -.25] |
|  |  |
|  | i = random.choice([0, 1, 2]) # [Center, Left, Right] |
|  | img = cv2.imread(os.path.join(DATA\_IMG, log\_content[index][i]).replace(" ", "")) |
|  |  |
|  | if img is None: continue |
|  |  |
|  | img = cv2.cvtColor(img, cv2.COLOR\_BGR2RGB) |
|  |  |
|  | rotation = float(log\_content[index][3]) |
|  |  |
|  |
|  |
|  |
|  |
|  | rotation = rotation + angle\_correction[i] |
|  |
|  | if random.random() > 0.5: |
|  | img = cv2.flip(img, 1) |
|  | rotation = rotation \* -1 |
|  |
|  | images.append(img) |
|  | rotations.append(rotation) |
|  |  |
|  | if len(images) >= batch\_size: |
|  | yield np.array(images), np.array(rotations) |
|  |  |
|  | images, rotations = [], [] |
|  |  |
|  | def main(): |
|  | """ |
|  | Main function to train the model |
|  | """ |
|  |  |
|  | with open(DATA\_PATH, "r") as f: |
|  | content = [line for line in csv.reader(f)] |
|  |  |
|  | random\_indexs = np.array(range(len(content))) |
|  | train\_index, valid\_index = train\_test\_split(random\_indexs, test\_size=0.15) |
|  |  |
|  | model = build\_model() |
|  |  |
|  | print("Train size = %s" % len(train\_index)) |
|  | print("Valid size = %s" % len(valid\_index)) |
|  |  |
|  | BATCH\_SIZE = 64 |
|  |  |
|  | model.fit\_generator( |
|  | generator=get\_data(content, train\_index, BATCH\_SIZE, strict=False), |
|  | steps\_per\_epoch=len(train\_index) / BATCH\_SIZE, |
|  | validation\_data=get\_data(content, valid\_index, BATCH\_SIZE, strict=True), |
|  | validation\_steps=len(valid\_index) / BATCH\_SIZE, |
|  | epochs=12) |
|  |  |
|  | model.save("model.h5") |
|  |  |
|  |  |
|  |  |
|  | if \_\_name\_\_ == '\_\_main\_\_': |
|  | main() |