# Pablo Sauras Perez Reflection.

# Term 1 – Project 3 – Behavioral Cloning

## 1. Files Submitted

This project submission has the following files:

- **model.py** contains the script to create, train and save the model.
- **drive.py** is the script for driving the car in autonomous model.
- **model.h5** contains the trained neural network.
- **Loss.png** shows the loss function of the train and validation sets.
- **model.png** shows the model visualization.
- **video.mp4** is the video of the test of my trained model in autonomous mode.
- **PabloSauras_Reflection_P3.pdf** is this report.

**Note:** for generating the model visualization, pydot-ng was installed.

## 2. Model Architecture and Training Strategy

### 2.a. Model Architecture

My model is based on the **NVIDIA convolutional neural network** for behavioral cloning that was explained in the lesson, with some changes. **Figure 1** shows the model visualization.

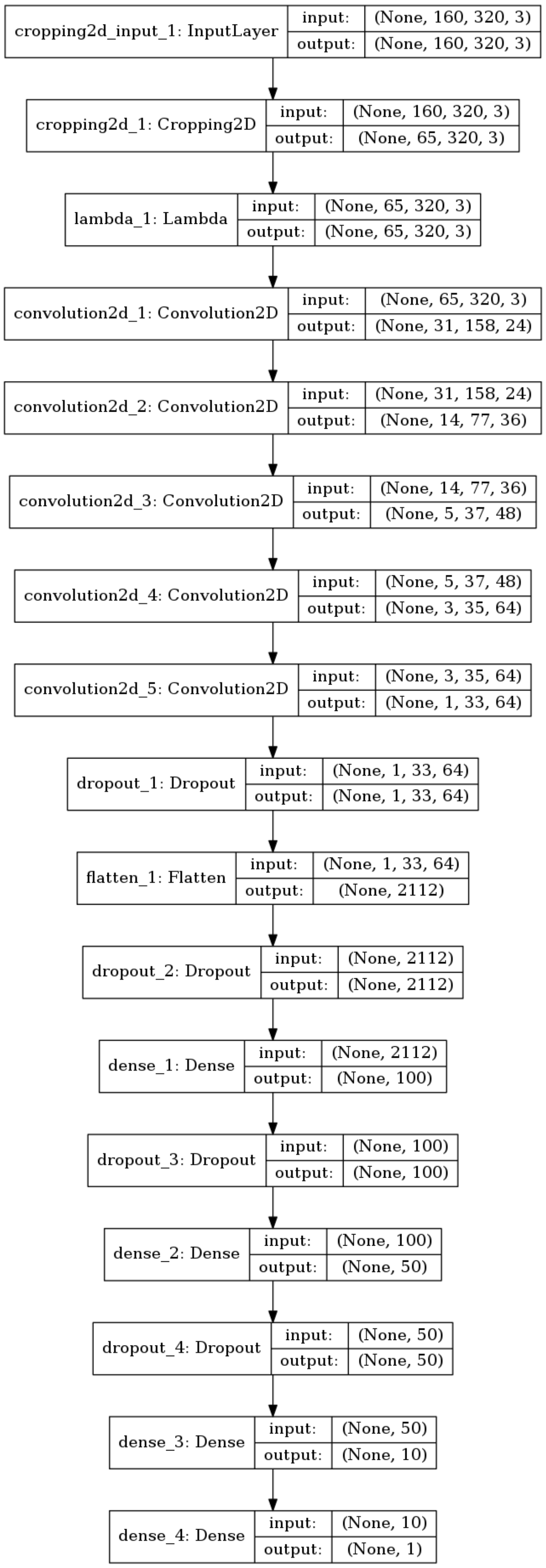
cropping2d_input_1: InputLayer
input: (None, 160, 320, 3)
output: (None, 160, 320, 3)
cropping2d_1: Cropping2D
input: (None, 160, 320, 3)
output: (None, 65, 320, 3)
lambda_1: Lambda
input: (None, 65, 320, 3)
output: (None, 65, 320, 3)
convolution2d_1: Convolution2D
input: (None, 65, 320, 3)
output: (None, 31, 158, 24)
convolution2d_2: Convolution2D
input: (None, 31, 158, 24)
output: (None, 14, 77, 36)
convolution2d_3: Convolution2D
input: (None, 14, 77, 36)
output: (None, 5, 37, 48)
convolution2d_4: Convolution2D
input: (None, 5, 37, 48)
output: (None, 3, 35, 64)
convolution2d_5: Convolution2D
input: (None, 3, 35, 64)
output: (None, 1, 33, 64)
dropout_1: Dropout
input: (None, 1, 33, 64)
output: (None, 1, 33, 64)
flatten_1: Flatten
input: (None, 1, 33, 64)
output: (None, 2112)
dropout_2: Dropout
input: (None, 2112)
output: (None, 2112)
dense_1: Dense
input: (None, 2112)
output: (None, 100)
dropout_3: Dropout
input: (None, 100)
output: (None, 100)
dense_2: Dense
input: (None, 100)
output: (None, 50)
dropout_4: Dropout
input: (None, 50)
output: (None, 50)
dense_3: Dense
input: (None, 50)
output: (None, 10)
dense_4: Dense
input: (None, 10)
output: (None, 1)

**Figure 1:** Model visualization

As it can be seen in **Figure 1**, my model **processes** the input in the following way:

- A **Cropping2D** layer (line 116) crops the input so that it removes the sky part of the input images and leaves only the road part of the input images.
- A **Lambda** layer (line 118) is used then to normalize the cropped images and set the mean around 0.

The model continues with **three convolutional** layers (lines 124-126) with **stride 2x2, kernel 5x5,** activation function **RELU** and **depths 24, 36,** and **48.** These are followed by **two non-strided convolutional layers** with **kernel 3x3** and **depths 64** (lines 129, 130)**.**

Following these convolutional layers the model has a **flatten** layer followed by **three fully connected layers** (lines 136, 138, 140) with output size **100, 50, 10.** As it can be seen in the code, in order to **reduce overfitting L2 regularizers** were used in these three layers. In addition, for the same purpose **dropout layers** were used as well.

Finally, a last **fully connected layer** creates the model output.

**Table 1** shows the layers of my final model (based on the **NVIDIA convolutional neural network** for behavioral cloning that was explained in the lesson).

**Table 1.** Model Architecture

| Layer | Input | Stride | Padding | Regularizer | Activation | Output |
|---|---|---|---|---|---|---|
| Input | 160x320x3 | | | | | 160x320x3 |
| Crooping2D | 160x320x3 | | | | | 65x320x3 |
| Lambda | 65x320x3 | | | | | 65x320x3 |
| **Convolution 5x5x24** | **65x320x3** | **2x2** | **VALID** | | **RELU** | **31x158x24** |
| **Convolution 5x5x36** | **31x158x24** | **2x2** | **VALID** | | **RELU** | **14x77x36** |
| **Convolution 5x5x48** | **14x77x36** | **2x2** | **VALID** | | **RELU** | **5x37x48** |
| **Convolution 3x3x64** | **5x37x48** | **1x1** | **VALID** | | **RELU** | **3x35x64** |
| **Convolution 3x3x64** | **3x35x64** | **1x1** | **VALID** | | **RELU** | **1x33x64** |
| Dropout (prob = 0.6) | 1x33x64 | | | | | 1x33x64 |
| **Flatten** | **1x33x64** | | | | | **2112** |
| Dropout (prob = 0.6) | 2112 | | | | | 2112 |
| **Fully Connected** | **2112** | | | **L2 (0.001)** | | **100** |
| Dropout (prob = 0.6) | 100 | | | | | 100 |
| **Fully Connected** | **100** | | | **L2 (0.001)** | | **50** |
| Dropout (prob = 0.6) | 50 | | | | | 50 |
| **Fully Connected** | **50** | | | **L2 (0.001)** | | **10** |
| **Fully Connected** | **10** | | | **L2 (0.001)** | | **1** |

### 2.b. Attempts to reduce overfitting in the model

As it was mentioned in the previous subsection **L2 regularizers** and **dropout** layers were used to reduce overfitting.

The model was also trained and validated on different data sets (**70/30** with respect the recorded training data) to ensure that the model was not overfitting. Also, the model was tested by running the simulator and ensuring that the vehicle could stay on the track (as show in **video.mp4**). Although it is not shown in the video, the model was also tested in the opposite direction from the default driving direction of the simulation. This test was also successful.

**Figure 2** shows the loss functions (mean square error) of the training and validation sets. As it is shown, the MSE is low for both sets. Thus, the model does not overfit.

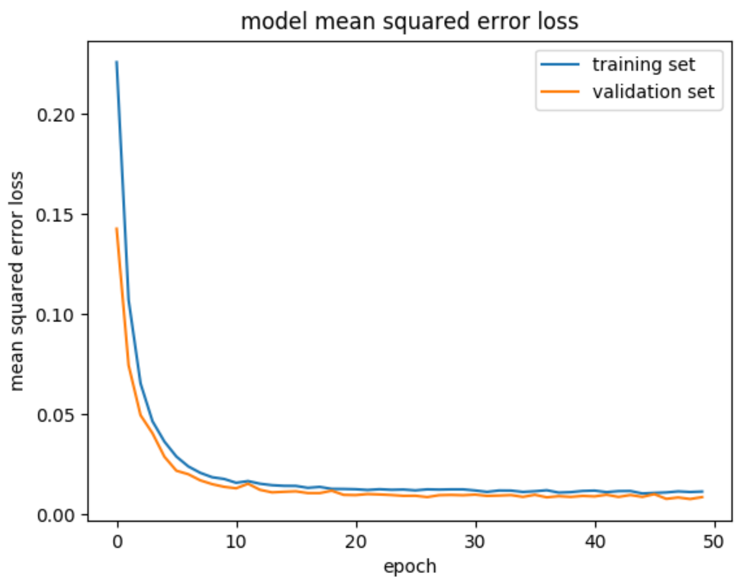

**Figure 2:** Loss of training and validation sets

## 2.c. Model parameter tunning

The model uses an **adam optimizer,** so the learning rate was not tuned manually.
The model was trained over **50 epochs.**

## 2.d. Appropriate training datasets. Creation of Training Sets and Training Process.

Training data was chosen to keep the vehicle on the road. **Center, Left and Right cameras** images with the corresponding **steering angles corrections** were used. In addition, images and steering angles were **flipped** in order to a**ugment the training data** and compensate the fact that the simulator driving direction is counter-clock wise.

I did not need to record data of the car recovering from the left and right sides. I am assuming that in a real world scenario, a driver in a highway (or other road) will not be recording that type of data (as it would be dangerous for the cross-traffic). Thus, **I focused on recording data with the car driving in the center of the track (good driving behavior)**.

**Figure 3** shows an example of training images. **Figure 4** shows the an example of the augmented training images (flipped).

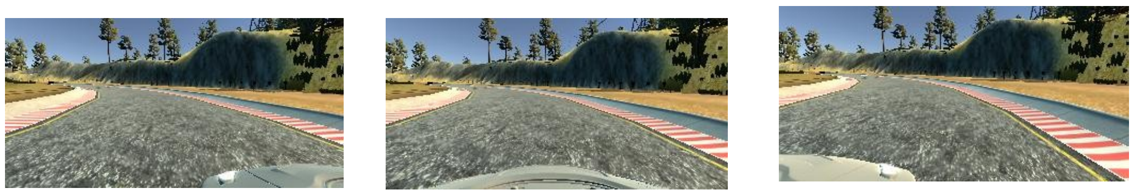

**Figure 3:** Left, Center and Right camera example.

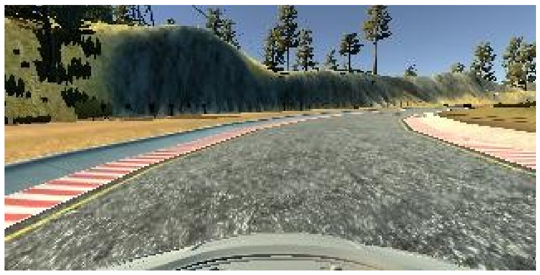

**Figure 4:** Example of data augmentation.
Flipped image

As it was mentioned before, the model was also tested in autonomous mode for clockwise driving direction with successful results. Thus, the **data augmentation** strategy based on images and steering angle flipping worked.

A **Python generator** was used to generate data for training rather than storing the training data in memory.

With respect to the testing of the model, I modified one line of **drive.py**. Following the udacity forum recommendations, the images captured in the simulator were converted from RGB to BGR, so that the images are of the same type as in the training data (drive.py line 66).