Project: Behavioral Cloning
William Galindez Arias

Introduction

Design and train a neural network to drive a simulator car. For this purpose, I collected my own training data using the simulator provided, I drove the car 2 laps (clock-wise and counter clock wise). I intentionally added abrupt steering to teach the car how to recover from a situation where it was about to drive outside the road.

Pipeline

1. General Outline: Supervised learning problem, where based on an image a steering angle to drive inside the road is calculated. Thus, getting a relation Input-Image → steering angle
2. The loss: The risk function selected is MSE which is good for this context given that extreme outliers are not expected if during the training data the car more or less was driven inside the road
3. The prediction algorithm: Regression that calculates a steering angle
4. Method utilized: Convolutional Neural network, the one is expected to learn the features of the images is fed and learn the weights to deliver a single output correspondent to the steering angle.

Challenges

- Training algorithm time: The neural network architecture can easily scale in parameters to be learned, making the computational and hardware efforts an aspect to consider.
- Typical Overfitting, under fitting scenarios where being able to interpret the learning curves helps to tune the epochs and find when to stop collecting data.
- The car was driving mostly in straight line all the time, making it unable to steer left or right when needed, possibly the reason that could explain this behavior is the unbalanced dataset that was being used, in which the angles -0.25, 0, 0.25 were present roughly 4x times more than the other measurements. See image below

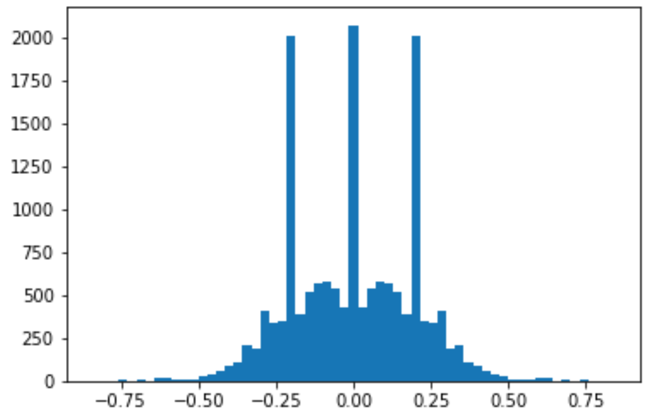


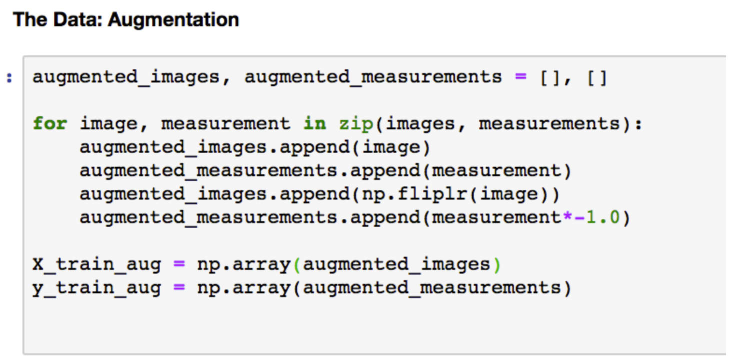
Fig1. Unbalanced Dataset

The unbalanced dataset provided a hint that too much data was being collected (30k images one camera) therefore the most frequent angles encountered in the driving log where dictating or biasing the output of the regression.

Steps followed to overcome the challenges

1. Data Augmentation

As suggested in the video lectures, the training data was augmented by flipping the images and changing the sign of the steering angle in order to help the model generalize better. In addition to this technique, training data driving counter clock wise was also collected.



The Data: Augmentation

```
augmented_images, augmented_measurements = [], []

for image, measurement in zip(images, measurements):
    augmented_images.append(image)
    augmented_measurements.append(measurement)
    augmented_images.append(np.fliplr(image))
    augmented_measurements.append(measurement*-1.0)

X_train_aug = np.array(augmented_images)
y_train_aug = np.array(augmented_measurements)
```

Fig2. Code snippet to augment training data

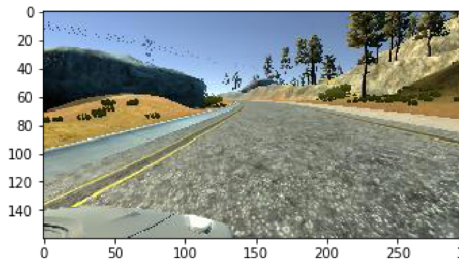


Fig3. Original Image

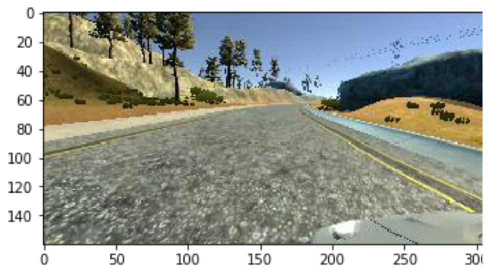


Fig4. Flipped Image

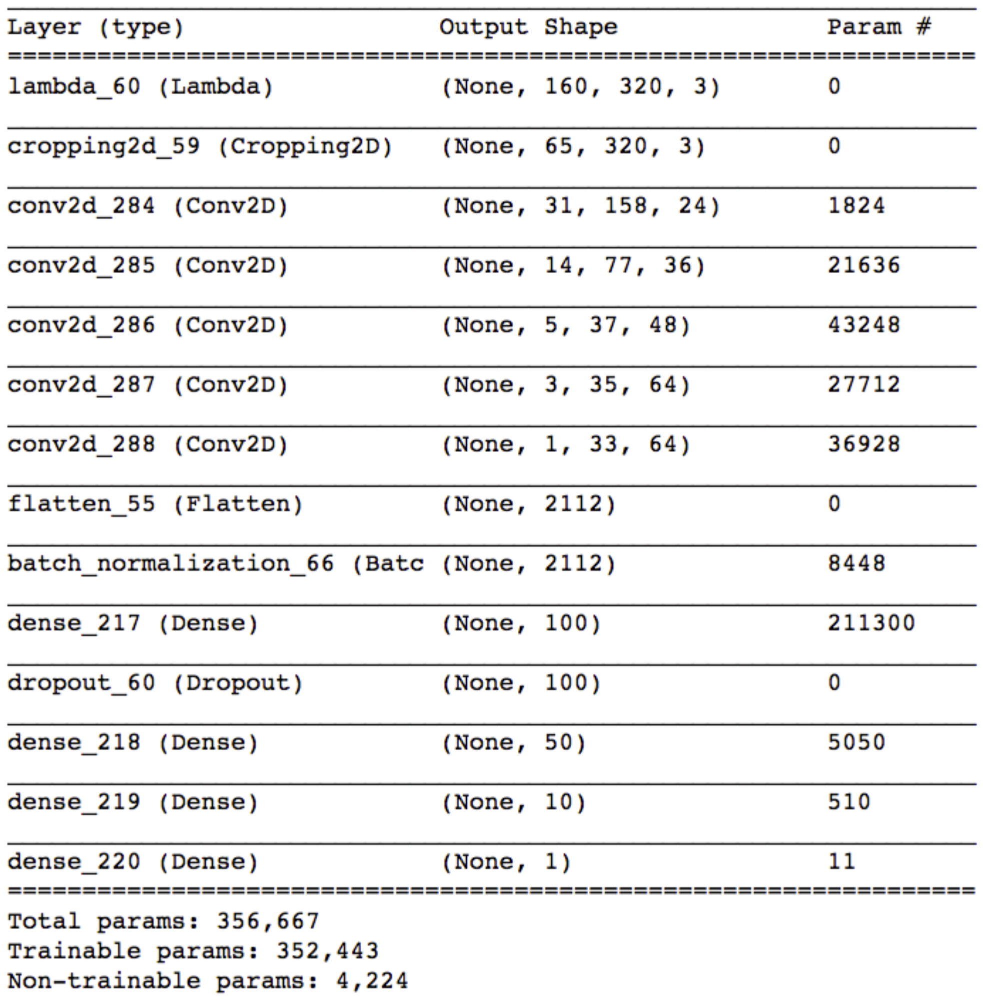
2. Convolutional Neural Network Architecture

In general terms, the model architected was mainly based in the NVIDIA Paper shared in the video lecture, nevertheless there are some differences implemented upon experimentation in the local environment. Such differences and the reason to use them are:

- *Keras* Batch Normalization Layer: Based on experimentation was found that this layer provides some sort of regularization and using it before the fully connected layers provided more stability to the model, possibly because as the theory implies, this layer was helping to reduce the co-variance of the model before these layers, therefore helping out to learn the parameters more independently.

- Dropout Layer: Regularization method added after the dense layer with 100 fully connected nodes, the placing of this layer in the specified position was made in order to help the model generalize better in the layer where more parameters where present thus "losing" information in this layer was affordable and the benefit of helping the model not to memorize the values was more visible here

2.1 Model Summary



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
lambda_60 (Lambda)           (None, 160, 320, 3)       0         
_________________________________________________________________
cropping2d_59 (Cropping2D)   (None, 65, 320, 3)        0         
_________________________________________________________________
conv2d_284 (Conv2D)          (None, 31, 158, 24)       1824      
_________________________________________________________________
conv2d_285 (Conv2D)          (None, 14, 77, 36)        21636     
_________________________________________________________________
conv2d_286 (Conv2D)          (None, 5, 37, 48)         43248     
_________________________________________________________________
conv2d_287 (Conv2D)          (None, 3, 35, 64)         27712     
_________________________________________________________________
conv2d_288 (Conv2D)          (None, 1, 33, 64)         36928     
_________________________________________________________________
flatten_55 (Flatten)         (None, 2112)              0         
_________________________________________________________________
batch_normalization_66 (Batc (None, 2112)              8448      
_________________________________________________________________
dense_217 (Dense)            (None, 100)               211300    
_________________________________________________________________
dropout_60 (Dropout)         (None, 100)               0         
_________________________________________________________________
dense_218 (Dense)            (None, 50)                5050      
_________________________________________________________________
dense_219 (Dense)            (None, 10)                510       
_________________________________________________________________
dense_220 (Dense)            (None, 1)                 11        
=================================================================
Total params: 356,667
Trainable params: 352,443
Non-trainable params: 4,224
```

Fig5. Architecture of the CNN

2.2 Learning Curves

Try & Error plus the reading of learning curves dictated how to set the number of epochs. Adam optimizer was utilized and MSE loss for the risk function.

Training Data and validation data where shuffled before training the model and split following the 80/20 proportion.

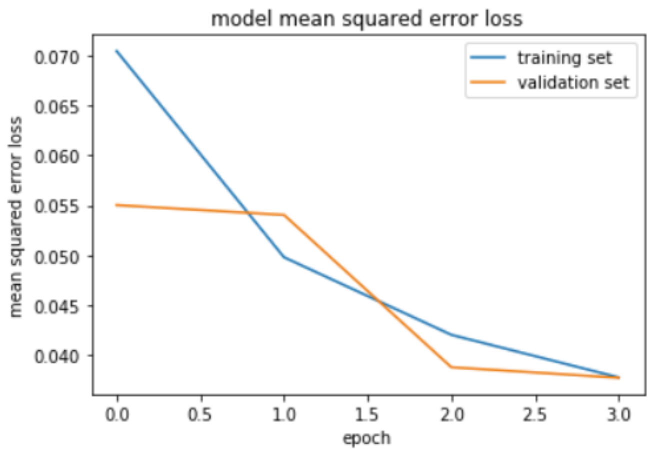


Fig6. Learning Curve of the model

In the different experiments ran was observed that after the third epoch the model started to over-fit.

3. The simulation

In the file run1.mp4 can be observed the car completing one lap, although the car shows some risky maneuverings, the transfer learning was a success because that is exactly how the human driver who was collecting the data performed the driving by using a mouse to steer. Important remark is the car is able to recover from a ‘risky’ behavior.

4. **Suggestions implemented not necessarily game changing**

- The Generator function was implemented, although it didn't offer a very noticeable improvement in the performance, possibly because the local hardware where the simulation was running possess 64GB of RAM, Core i7 QuadCore 4th Gen.
- The Center camera, left and right were used in some simulations, turned on and off to see improvements in the results without further significance