UDACITY

‹ Return to Classroom

Use Deep Learning to Clone Driving Behavior

REVIEW

CODE REVIEW

HISTORY

Meets Specifications

Well done with overall project! Congratulations and good luck in your next project!

Here you can more about behavioral cloning for self-driving cars:
https://www.youtube.com/watch?v=bD05uGo_sVI
https://www.youtube.com/watch?v=rpxZ87YFg0M&feature=youtu.be (if you did't see this video before 😄)

Here are also interesting discussion about this project from one of Udacity student from the first cohort:
https://medium.com/@vivek.yadav/cloning-a-car-to-mimic-human-driving-using-pretrained-vgg-networks-ac5c1f0e5076#.phu0cncdb
https://chatbotslife.com/learning-human-driving-behavior-using-nvidias-neural-network-model-and-image-augmentation-80399360efee#.586eoyqve
https://chatbotslife.com/using-augmentation-to-mimic-human-driving-496b569760a9#.4rwzs7ozx

And some interesting resources:
https://medium.com/udacity/teaching-a-machine-to-steer-a-car-d73217f2492c#.gvl8tkusw
https://jacobgil.github.io/deeplearning/vehicle-steering-angle-visualizations
http://selfdrivingcars.mit.edu/

Required Files

The submission includes a model.py file, drive.py, model.h5 a writeup report and video.mp4.

Well done! All required files are provided!

Quality of Code

The model provided can be used to successfully operate the simulation.

The code in `model.py` uses a Python generator, if needed, to generate data for training rather than storing the training data in memory. The `model.py` code is clearly organized and comments are included where needed.

Well done with `yield`! `yield` generator is better to use to generate data for training rather than storing all data in memory - it will improve memory

performance. Here are an excellent video and discussion about generators:
https://www.youtube.com/watch?v=bD05uGo_sVI
http://stackoverflow.com/questions/7883962/where-to-use-yield-in-python-best

Well done with comments in model.py - code is very clean and well commented.

Code readability is very important code metric, especially if you are:

1. working in team and other team members will investigate/review your code
2. going to maintain this project for a long time.

Refer please to self-documenting code:
https://en.wikipedia.org/wiki/Self-documenting_code
http://stackoverflow.com/questions/209015/what-is-self-documenting-code-and-can-it-replace-well-documented-code
http://c2.com/cgi/wiki?SelfDocumentingCode

Model Architecture and Training Strategy

The neural network uses convolution layers with appropriate filter sizes. Layers exist to introduce nonlinearity into the model. The data is normalized in the model.

- multiple 2D convolution layers are used
- nonlinearity applied using rELU. Here is a good article about some activation layers if you are interested:
  https://arxiv.org/pdf/1511.07289v1.pdf
  https://keras.io/layers/advanced_activations
  http://yann.lecun.com/exdb/publis/pdf/lecun-98b.pdf
- data is normalized

Well done! :+1:

As an enhancement you can also normalize layers:
https://keras.io/layers/normalization/
http://stackoverflow.com/questions/34716454/where-do-i-call-the-batchnormalization-function-in-keras
https://arxiv.org/abs/1502.03167
This technique can increase learning time but also improve overall performance.

Here is also an excellent discussion about batch size:
http://stats.stackexchange.com/questions/140811/how-large-should-the-batch-size-be-for-stochastic-gradient-descent

Train/validation/test splits have been used, and the model uses dropout layers or other methods to reduce overfitting.

Well done with dropout and train/test/validation split!

Here are more information about train/validation/test splits if you are interested:
http://stats.stackexchange.com/questions/19048/what-is-the-difference-between-test-set-and-validation-set
http://stackoverflow.com/questions/13610074/is-there-a-rule-of-thumb-for-how-to-divide-a-dataset-into-training-and-validatio

And here are also more info about dropout:
http://machinelearningmastery.com/dropout-regularization-deep-learning-models-keras/
http://www.cs.toronto.edu/~rsalakhu/papers/srivastava14a.pdf
https://pgaleone.eu/deep-learning/regularization/2017/01/10/anaysis-of-dropout/

Learning rate parameters are chosen with explanation, or an Adam optimizer is used.

Well done with Adam optimizer!
Here is an excellent article about different gradient descent optimization algorithms:
http://sebastianruder.com/optimizing-gradient-descent/index.html

Training data has been chosen to induce the desired behavior in the simulation (i.e. keeping the car on the track).

Architecture and Training Documentation

The README thoroughly discusses the approach taken for deriving and designing a model architecture fit for solving the given problem.

Good job with README!

The README provides sufficient details of the characteristics and qualities of the architecture, such as the type of model used, the number of layers, the size of each layer. Visualizations emphasizing particular qualities of the architecture are encouraged.

Items to describe:

- Type of model used - well done 👌
- Number of layers - well done 👌
- Size of each layer - - well done 👌

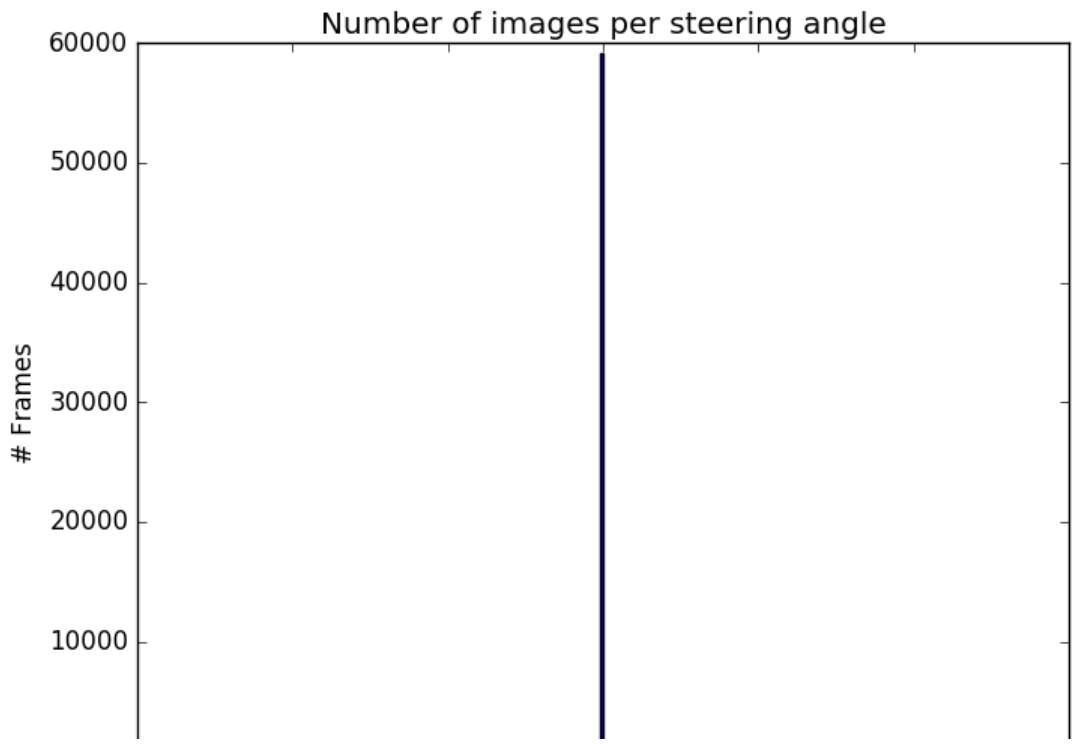
You cant also try to use the following Keras function for vizualization:
https://keras.io/visualization/#model-visualization

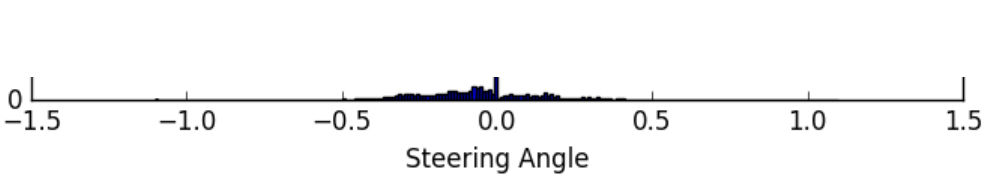
The README describes how the model was trained and what the characteristics of the dataset are. Information such as how the dataset was generated and examples of images from the dataset must be included.

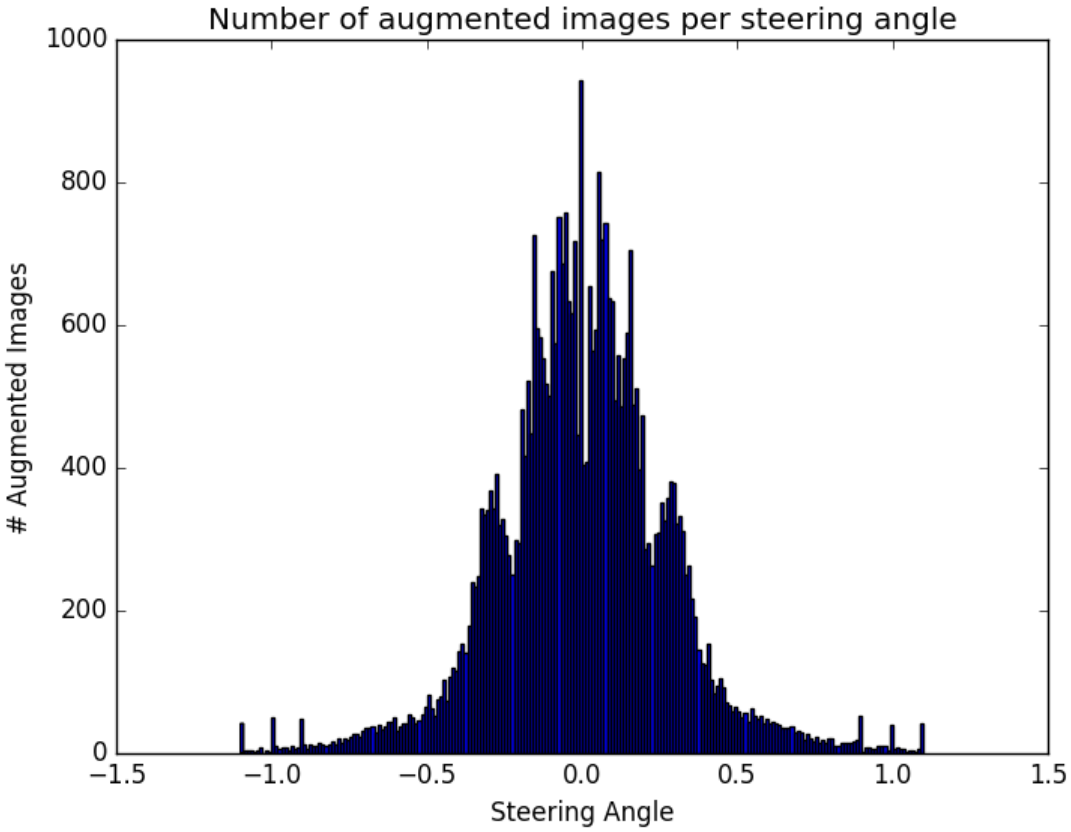
Well done with description of dataset generation process and examples of preprocessed images included in readme!

Note please that there is an excellent Model Checkpoint function in Keras that allow you to save the model after each epoch and later you can just choose the best one:
https://keras.io/callbacks/#modelcheckpoint

Also as an example, you can provide the histogram of steering angles to be sure that it well balanced. If you collect data with keyboard a lot of zero values should be presented in it. In this case, you should balance by removing, for example, some portion of zero values training data:

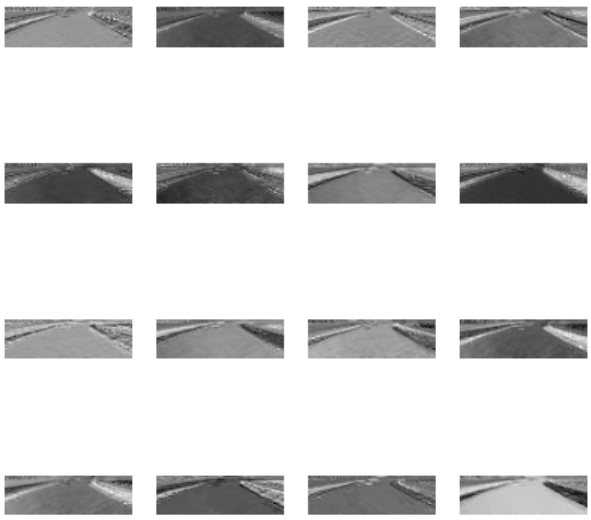




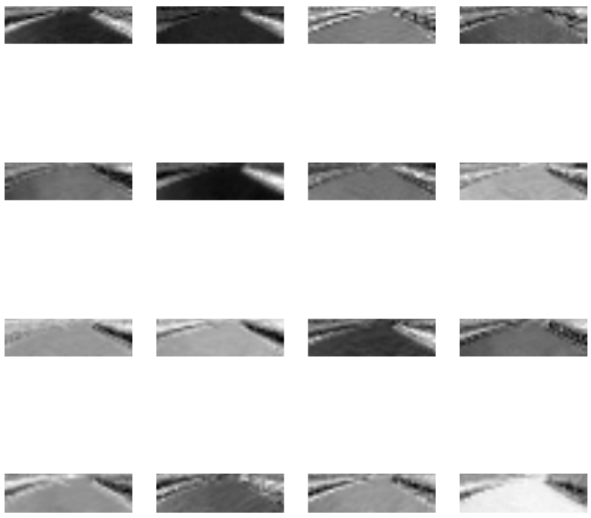


Another type of images you can provide is a layers visualization like this:

Layer 1



Layer 2



You can use for this visualization the following code for example:

```
# Example of image for layers visualization
test_fn = "path_to_image"

def visualize_layer(layer):
    '''
    Function to visualize layers
    layer - layer name
    '''
    model_vis = Model(input=model.input, output=model.get_layer(layer).output)

    # Image augmentation as in model - HERE  YOU SHOULD IMPLEMENT YOUR AUGMENTATION PROCESS
    img = plt.imread(test_fn)[55:135]
    img= imresize(img, (66, 200), interp='bilinear', mode=None)
    img = cv2.cvtColor(img, cv2.COLOR_RGB2HLS)
    img = np.expand_dims(img, axis=0)

    features = model_vis.predict(img)
    print("Shape of features: ", features.shape)

    # plot features
    plt.subplots(figsize=(5, 5))
    for i in range(16):
        plt.subplot(4, 4, i+1)
        plt.axis('off')
        plt.imshow(features[0,:,:,i], cmap='gray')
    plt.show()

visualize_layer('Convolution1')
```

```
visualize_layer('Convolution1')
visualize_layer('Convolution2')
```

Simulation

No tire may leave the drivable portion of the track surface. The car may not pop up onto ledges or roll over any surfaces that would otherwise be considered unsafe (if humans were in the vehicle).

DOWNLOAD PROJECT

RETURN TO PATH