BEHAVIORAL CLONING PROJECT

Model Architecture and Training Strategy

Introduction

The general approach to this project was to iterate through various combinations of model and data refinement until successful self-driving was achieved for at least 10 laps around the track.

The first model architecture was based on Transfer Learning. The idea was to use a set of pretrained convolutional layers along with a custom classification layer to model driving behaviour. This would allow me to focus on tuning the classifier along with the data without needing to also train convolutions. The intial approach was also to use the full size simulator images (320x160 pixels); though later that was changed.

The initial architecure used the convolutional layers of VGG16 along with a 3-layer classifier. This yielded moderate results but also the realization that VGG16 was too heavyweight to run on CPU-only to successfully guide the car. This realization took some time because the resource requirements of VGG16 were occludded by the Nvidia GPU on my laptop.

The initial approach also used grouping the steering angles into 201 classes. This allowed a 0.01 resolution in the steering angle in the range -1 to +1. This led to a sparse distribution of the data, very low accuracy rates in the output, and somewhat erratic steering behaviour. After awhile and some analysis of simulator behaviour and steering angle distribution, I came to the conclusion that 0.01 resolution was really not necessary so some experimentation of number of classes was needed.

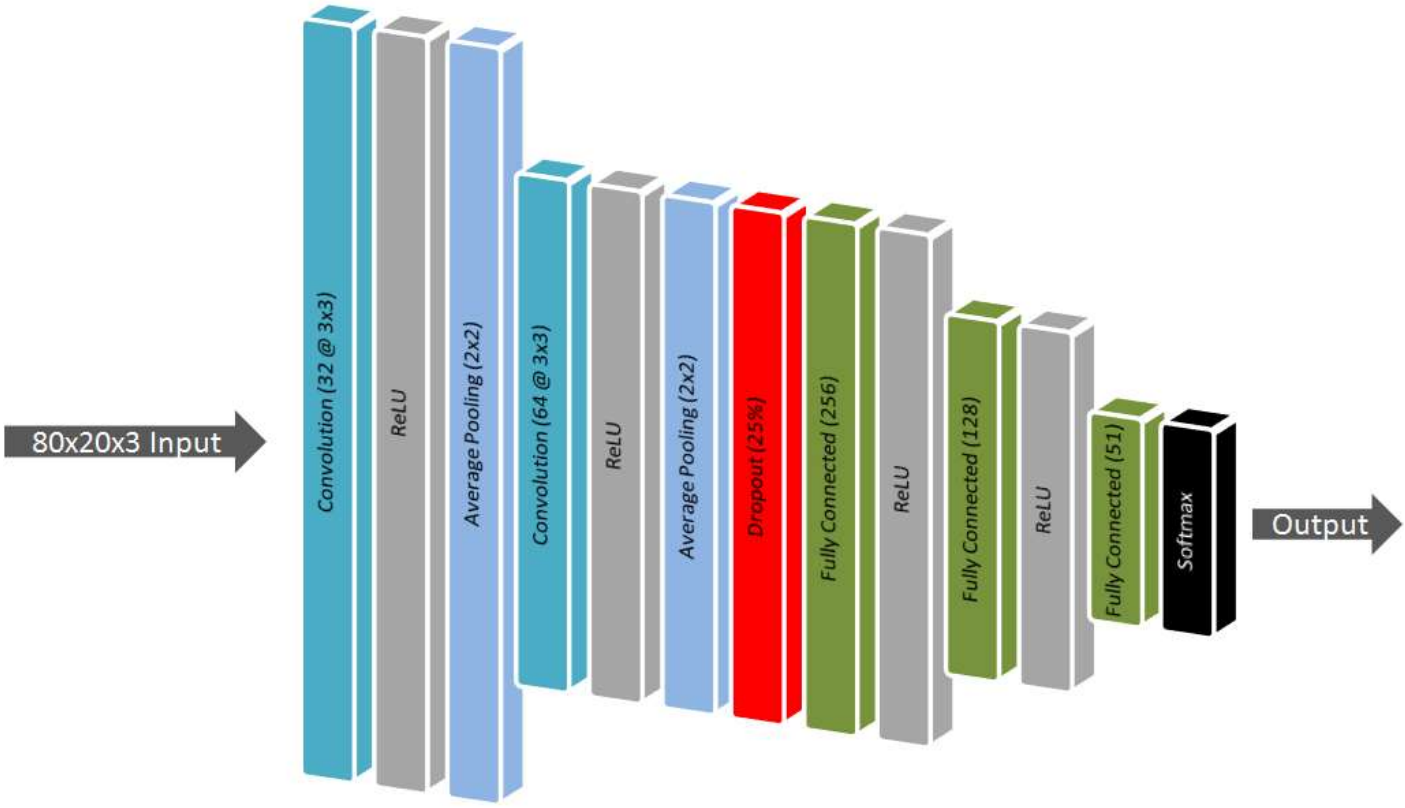
The most iterations yielded a bias towards 0 steering (driving straight ahead) and steering to the left. Data analysis yielded the same bias the the data.

Finally, an attempt was made to use alternate views of the road. The first attempt was to use the side cameras with adjusted steering angle data. This did not yield substantially better results. The second attempt split the simulator image into thirds: left, right and center, and adjust the steering angle appropriately. This attempt successfully navigated the first corner 9 times out of 10 but resulted in the car weaving back and forth across the road. However, the classifier still seemed to have an obsession for unqiue features on the landscape; sometimes veering off the road at very specific areas.

After the first 20 iterations, several changes to architecture and approach were made:

- Switch from VGG16 to custom, lightweight convolution layers
- Keep the 3-layer classifier but experiment with the layer sizes
- To reduce the number of parameters to tune, use an Optimizer such as RMSPRop or Adam
- Experiment with a reduced number of classes (less than 201) This would require some data analysis
- Data, data, data. Augment the data as required if the behaviour in the simulator is not as required
- Crop the simulator images to focus on the road and hide as much distractions as possible.
- Avoid using the side cameras.
- Something had to be done to reduce the bias towards driving straight ahead.

Model Architecture



Layer Type	Layer Shape	Output Shape	#Parameters	Input
Convolution2D	32*3x3	(None, 20, 80, 32)	896	80x20x3 (Image)
ReLU Activation			0	
AveragePooling2D	2x2	(None, 10, 40, 32)	0	
Convolution2D	64@3x3	(None, 10, 40, 64)	18,496	
ReLU Activation			0	
AveragePooling2D	2x2	(None, 5, 20, 64)	0	
Dropout	25%	(None, 5, 20, 64)	0	
Flatten		(None, 6,400)	0	
Fully connected	256	(None, 256)	1,638,656	
ReLU Activation			0	
Fully connected	128	(None, 128)	32896	
ReLU Activation			0	
Fully connected	51	(None, 51)	6579	
Softmax		(None, 51)	0	

The entire model is enclosed in the simple_model() function at line 364 of the TrainSimulator.py code.

Solution Design

After initially trying VGG16 trained on ImageNet, plus a custom 3-layer classifier with some success, I decided to try a smaller model with the same size classifier, but less convolutions. Assuming that the size of VGG16 convolutional layers was dictated by the volume of features needed to be extracted from ImageNet's 1000 classes, I assumed that using a the simulator would need at least 3 times less given the target number of classes. I experimented with between 2 and 4 convolutional layers and 2x2 or 4x4 pooling layers to find a balance between reducing the number of parameters in the model and being able to recognize enough features to drive the simulator. I used the consistent improvement, and rate of improvement of the validation accuracy during training epochs to guage whether the model was getting closer to a fit. This approach was used over several batch sizes between 64 and 1024 to properly assess whether the model was a proper fit.

The number of output classes was reduced to 51 since it was determined that this should give a 1 degree resolution in steering angle while smoothing out the steering by consolidating multiple close values into a single class. The steering angles were also rescaled before attempting training to a linear integer range from 0 to 50 (see scale_labels() function at line 64). This not only allowed one-hot encoding but also provided an easy means of deriving the steering angle from the output array index using the formula:

angle = (class_index/mid) - 1

where, mid = (number_classes - 1)/2

This can be seen in the steering_angles() function at line 34 of the drive.py code.

The ReLU was used to introduce non-linearity into the model. Dropout (25%) was used to reduce the level of overfitting in the model.

Preprocessing

Initially, image data was reduced to zero-mean using the training set mean per color channel. This was simplified to dividing by color depth and subtracting 0.5 which is simpler to process but seems just as effective. Images are also cropped both the reduce available data and also focus on the road. The viewport chosen was a 320x80 pixel size located from (0,60) in the image. This gave the full width of the image while removing anything above the horizon and the car's hood.

Cropped images were then resized to 80x20 full color, while keeping the aspect ratio of the previous image. The decision to keep the aspect ratio was made to avoid introducing another parameter to tune in the event that it impacted the model results.

Data collection

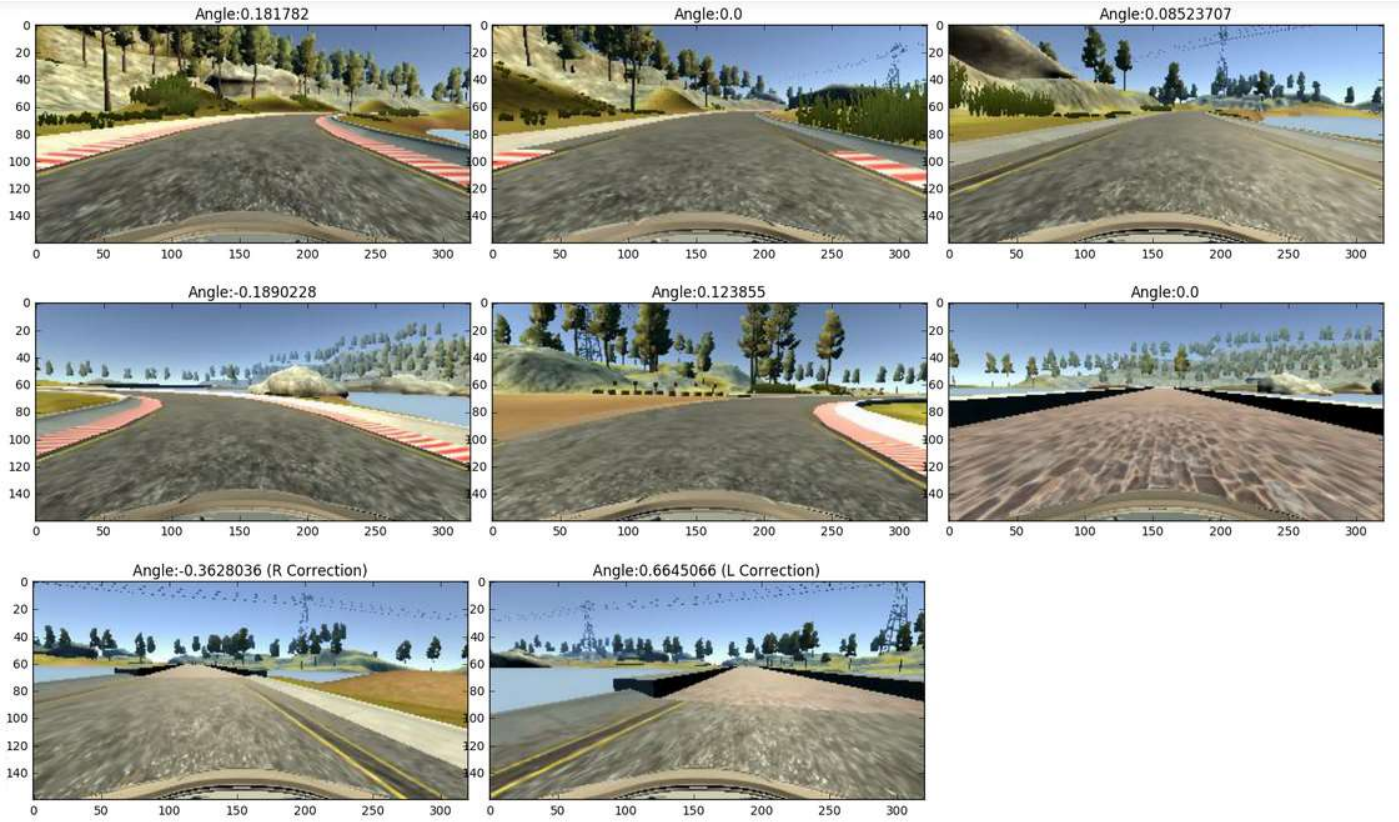
An incremental approach was taken to data collection after the initial 20 attempts with the Udacity provided data. It was decided to start from scratch with fresh data built as needed.

First 2 laps of track 1 were captured: one in forward direction, then a second after doing a three-point turn and driving in the opposite direction. After testing with this data and identifying its difficiencies/difficulties, more data was added. Testing was done after training for a varying number of epochs (5, 20, 25 epochs). It was found that the deeper training, as long as it did not represent overtraining, would result in better performance in some areas, but not others. After this approach, additional data was collected to reduce deficiences.

Three subseqent rounds of data collection were used:

- Corrections consisting of pointing the car at the side of the road, on each side, and drving back towards the center.
- Corrections consisting of aligning the car with the side of the road and driving back to center.
- Having my friend (a video gamer) record 3 additional laps of the track since his driving was better than mine.

This resulted in progressively better ability to handle turns while staying close to center of the road. Using a combination of my friend's driving data and additional training actually resulted in smoother performance (less zig-zagging).



Usage of track 2 (release and beta versions)

For initial trials, I did use track 2 for additional data. However, I decided in the latter stages to focus on track 1 data for this submission. After testing on track 2, I found that tha model was able to negotiate quite a bit of the track before crashing into the side; this without ever having seen it before. This provides a good indication of the generalization of the model.

The model was also tested on the newest 2-lane track of the current Open Source (Beta) simulator. The results indicate that both additional training data (from this 2-lane track) as well as additional preprocessing (conversion to HSV color-space e.g. to minimize the effects of shadows on the road).

Dataset preparation

After analysing the data it was found that the model tended to prefer 0-steering because of the over-sized presentation of 0-steering in the training set. To balance the model somewhat, it was decided to reduce the number of 0-steering samples by deleteing any sequences of zeros longer than 8. This number was found

after experimenting. The use of sequences of 0's was chosen by assuming that a sequence of zero steering that close together represents duplicate data in the training set. This choice drastically reduced the 0-steering in the results and allowed the classifier to identify turns correctly from a relatively small training set of just under 10,000 samples. Those 10,000 samples were reduced to just over 5,000 with this method (the downsample_zeros() function at line 101 of TrainSimulator.py).

After cropping, each image was also flipped horizontally with the steering angle adjusted accordingly to double the dataset. This was done dynamically in a python generator using OpenCV cv2.flip() function.

Training and Validation Sets

The resulting data was saved in pickle files for reuse in subsequent training runs. The training set was split into Training and Validation sets with a 90/10 split. Testing was done directly in the simulator.

The data was shuffled before splitting to randomize the order of the samples.

Notes on Training the Model

Execute the TrainSimulator.py script using the following commands

python TrainSimulator.py -h = Runs and displays help on options. This gives the following:

usage: TrainSimulator.py [-h] [--mode MODE] [--batch BATCH] [--epochs EPOCHS] [--test] Remote Driving Trainer optional arguments: -h, --help show this help message and exit --mode MODE Select VGG16 (deprecated) or Simple model: vgg|simple [simple] --batch BATCH Batch Size [512] --epochs EPOCHS Number of Training Epochs [20] --test Run in test mode. This uses a subset of the training set [False]

python TrainSimulator.py = Runs with the default model, whether not test mode, epoch and batch sizes. Default model is the simple classifier (VGG16 is deprecated). Default batch is 512, default epochs is 20, and test mode is turned off by default (test mode will run the model with 10% of the total data available).

Training data (images in 'IMG folder and driving_log.csv) must reside in the 'data' folder.

The TrainSimuator.py is dependent of the ImageProcessing.py script (provided).

Notes on running in autonomous mode.

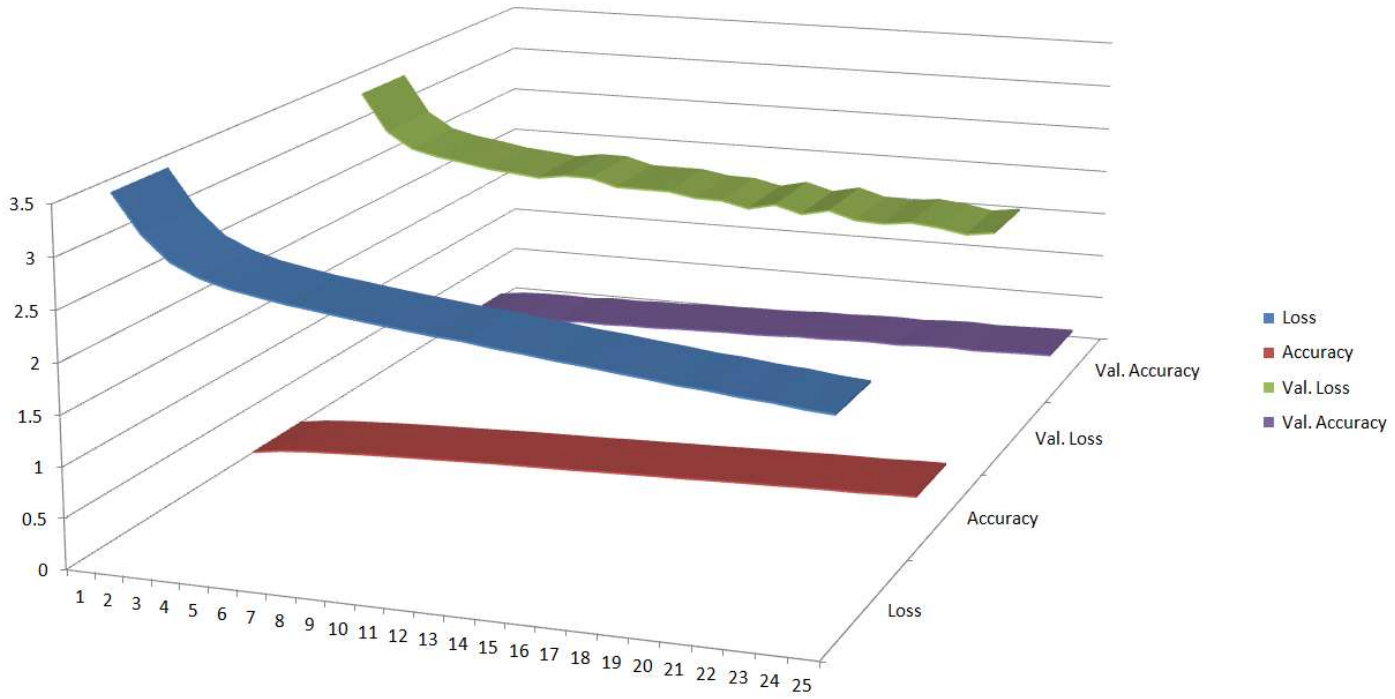
A custom version of drive.py is provided. IT depends on the ImageProcessing.py script (provided). It contains uses two custom functions: the get_steering_angle() that converts the class ID to steering angle, and get_viewport() from ImageProcessing.py, which crops the viewport used for processing.

Test Results

The final model trained to a validation accuracy of 30% after 25 epochs of 512 batch size and was able to successfully navigate the track continuously for over 15 laps at upto 30 mph. This testing was done with GPU enabled for drive.py and a 800x600 resolution on the simulator. The same was true without the GPU but never tested past 15 mph.

The model stays on the track though in some sections it veers close to or on the side lines but never on the sidewalk. It also zig-zags somewhat in 2 other sections.

The training progress report for accuracy and error are show below:



Data Analysis

The following two sections outline the data visualization showing the results of using the side cameras with steering adjustment as well as flipping the images and steering to balance the data set. The idea was to find ways to remove bias towards a particular direction and help the simulator focus on the most appropriate steering to make.

All simulator recordings were also reviewed as video sequences to identify sections of "bad" behaviour (poor driving on my part) and edit them out of the training data set. This was the first attempt to fix poor behaviour in the model.

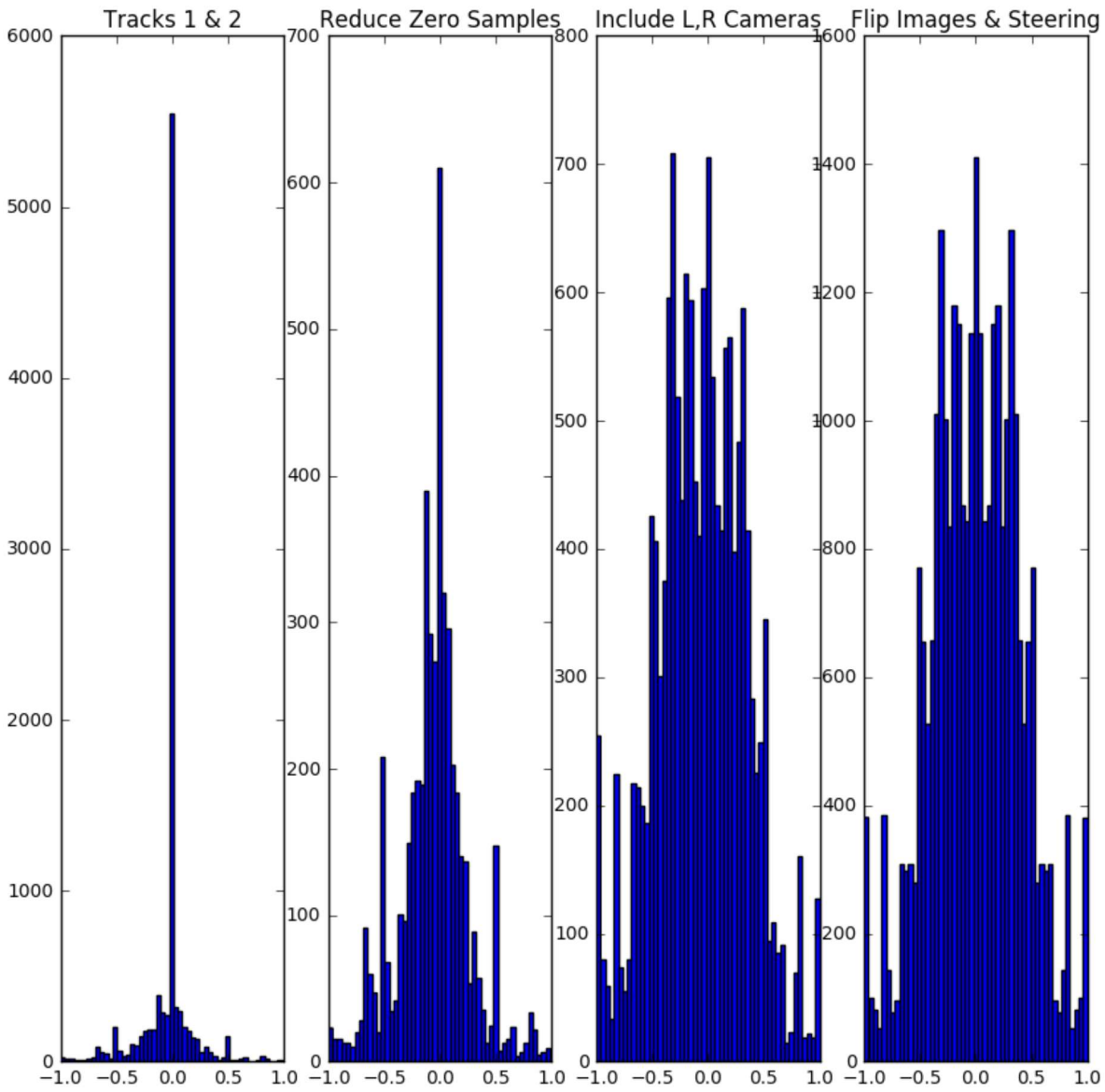
In [2]:

```
Using TensorFlow backend.

Length: original 9989 cleansed 5051
Std Deviation: original 0.225 cleansed 0.313
```

In [3]:

```
b'IMG/center_2017_02_09_23_52_50_038.jpg' 0.06110085 30306
```



Data Analysis conclusions

- Downsampling zeros to prevent zero-steering bias is effect in promoting other steering angles
- The original data which consisted on driving 2 laps each in opposite directions on the track as well as a lap of corrections, is stll not completely balanced.
- Flipping images and steering angles doubles the dataset for free and balances the distribution nicely