

# Competition Overview and Postmortem

Sunday, September 13, 2020 11:28 AM

- [NOTE 9/13/20: To be added after competition winners publish their code and I can review it. At this point I've only seen the one solution from the end of competition webinar from leading teams.]

## Competition Overview

- This was an interesting challenge. The dataset was sets of four images corresponding to left, right, front, and back camera angles from a car where we were tasked with predicting the angle of the steering wheel. <https://www.aicrowd.com/challenges/ai-for-good-ai-blitz-3/problems/autodri>
- I suspect that people came up with a lot of different approaches to fitting this dataset and I'm looking forward to seeing those different approaches, I certainly thought of a ton of different architectures which might work. If there was more time I would have stacked or boosted those different approaches together.
- Due to time constraints I stuck with one approach and then just boosted models with different NN architectures and cv folds together. This worked very well.

## Model Overview

- For this competition my approach was to consider this a video analysis problem and string the four angles together as the frames of a video
  - I ordered the images in a standard sequence (['Left', 'Front', 'Right', 'Rear']) which I'm guessing was necessary for this to work, but I didn't experiment with this. It could be that randomly ordering the images would act as an augmentation or regularizer but I didn't have time to explore this.
- The model architecture relies on Keras's TimeDistributed layer to wrap the base\_model (using efficientnet here) then feeds into an LSTM and and out through a standard MLP
  - NOTE: in my first notebook 200901 AutoDrive keras.ipynb I forgot to add the MLP and basically just had a single Dense layer with one neuron after the LSTM layer and this somehow worked equally well. The only reason I abandoned this approach was because AFAIK everyone sticks an MLP on top their base networks and so I figured I should as well. Something to explore later
- The architecture used was:

```
1 base_model = keras.applications.DenseNet201(include_top = False, weights = 'imagenet')
2 | | input_shape = (224, 224, 3))

1 # create a Sequential model
2 model = keras.models.Sequential()
3
4 # add base_model for 4 input images (keeping the right shape
5 model.add(keras.layers.TimeDistributed(base_model, input_shape=(4, 224, 224, 3)))
6
7 # now, flatten on each output to send 4 outputs with one dimension to LSTM
8 model.add(keras.layers.TimeDistributed(keras.layers.Flatten()))
9 model.add(keras.layers.LSTM(256, activation='relu', return_sequences=False))
10
11 # finalize with standard MLP
12 model.add(keras.layers.Dense(128, activation=None))
13 model.add(keras.layers.BatchNormalization())
14 model.add(keras.layers.Activation('relu'))
15 model.add(keras.layers.Dropout(0.25))
16
17 model.add(keras.layers.Dense(64, activation=None))
18 model.add(keras.layers.BatchNormalization())
```

```

16
17 model.add(keras.layers.Dense(64, activation=None))
18 model.add(keras.layers.BatchNormalization())
19 model.add(keras.layers.Activation('relu'))
20 model.add(keras.layers.Dropout(0.25))
21
22 model.add(keras.layers.Dense(1,activation='linear'))

WARNING:tensorflow:Layer lstm_1 will not use cuDNN kernel since it doesn't meet the cuD

```

---

```

1 # freeze the base model which is inside the first timedistributed layer
2 model.layers[0].trainable=False

```

- Important to note the input shape to the CNN and to the TimeDistributed layers. The CNN takes a single image and the TimeDistributed takes a sequence of 4 images with the same shape
  - I based this approach off of <https://medium.com/smileinnovation/how-to-work-with-time-distributed-data-in-a-neural-network-b8b39aa4ce00>
- One of the big challenges for this approach was that I had to write a custom image generator. In the end this wasn't too complicated, but it took me pretty much a whole day of labor to figure out

```

1 def get_imgs(imlist):
2     imgs=[]
3     for impath in imlist:
4         img = load_img(impath,target_size=(224, 224))
5         imgs.append(img_to_array(img))
6         # Pillow images should be closed after `load_img`,
7         # but not PIL images.
8         if hasattr(img, 'close'):
9             img.close()
10
11     return np.stack(imgs,axis=0)
12
13 def imageseq_generator(df, batch_size = 64):
14     inds=df.index.to_list()
15     while True:
16         # shuffle the indices for the epoch
17         np.random.shuffle(inds)
18
19         # Get index to start each batch: [0, batch_size, 2*batch_size, ...]
20         for offset in range(0, len(inds), batch_size):
21             # Get the samples you'll use in this batch
22             batch_inds = inds[offset:(offset+batch_size)]
23
24             batch_input = []
25             batch_output = []
26             # Read in each input, perform preprocessing and get label
27             for ind in batch_inds:
28                 batch_input.append(get_imgs(df.loc[ind,'img_list']))
29                 batch_output.append(df.loc[ind,'canSteering'])
30
31             # Return a tuple of (input, output) to feed the network
32             yield (np.array(batch_input), np.array(batch_output))

```

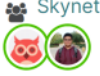







- This worked great and is also where I think I left most on the table. I didn't have time to implement any image augmentations with this generator! IMO you should always use augmentations when training on images (and on any other kind of data where it makes sense too!). **This is the first place I would focus**

### on to improve this

- One thing that was problematic with this approach was transfer learning. I used pretrained CNN models but was having issues with the standard freeze-train-unfreeze-train approach.
  - I think I froze the base model correctly where I tried both code snippets below and got identical results (suggesting either they both worked or didn't work)
  - ```
model.layers[0].trainable=False
```
  - ```
for layer in base_model.layers:  
    layer.trainable = False
```
  - Then I trained until the validation loss stopped improving
  - The problem arose when I tried to unfreeze the base layer and resume training.
  - My expectation was for the loss to start approximately where it had left off in the previous training round and then train as normal (improve at least a bit)
  - However what always happened was that the loss would start off basically at the same levels as at the very beginning of training and then would converge to loss level much higher than in the frozen training round.
  - I don't understand what was happening here. I assume that I was making a mistake somehow, but it could also be that I'm misunderstanding the TimeDistributed wrapping a CNN approach and that this approach to transfer learning doesn't work for some reason...
- I used a standard boosting approach. Trained this architecture on a bunch of different CNN models and different cv folds then trained a gradient boosting model on the predictions on the training set and used that model to make predictions on the test set

### Postmortem

- NOTE: I was having some confusion over the units. Firstly, clearly there's a big difference between the angle of the steering wheel and the angle of the tires, but I'm not quite sure if it makes sense for the steering wheel to rotate through  $[-720, 720]$  which is what the target variable here ranged from. Also XGBoost only had RMSE instead of MSE but there still seemed to be an order of magnitude difference in the squared RMSE and the MSE on the test set
- The boosting regressor reached RMSE=30 on a validation set but only achieved an MSE=10,441 on the test set which put me in 6th place. The top of the leaderboard was (I'm Benai)  
<https://www.aicrowd.com/challenges/ai-for-good-ai-blitz-3/problems/snake/leaderboards>

	Δ	#	Participants	MSE
	▲	01	 Skynet	6894.098
	▲	02	 jayaramanjay97	9212.303
	▼	03	 AmaurySu	9550.211
○	▼	04	 roboty_zbunto...	10094.31
	▼	05	 TODO	10221.597
	▼	06	 Benai	10441.386
	●	07	 BayesianMech...	11811.142
	●	08	 Zac	13115.616

- Where you can see that Skynet did much better than everyone else, but besides them the other top solutions were fairly closely clustered around 10k
  - I'm not sure if it was Skynet, but the presenter in the webinar pointed out that they realized that the different sets of images were all different frames (from each angle) of a continuous movie of a car driving and that they somehow took advantage of this. If so, this definitely sounds like it goes against the spirit of the challenge though I'll let the competition organizers decide if it counts as cheating. Either way it is solving a different problem than the one I was solving
- The boosting classifier didn't have a true validation set, because the individual NNs had been trained on the data. To really measure the performance of the boosting classifier a true test set would have been needed, but I didn't have time to do that experiment AND train the models on as much data as possible.
  - Just means that the RMSE=30 isn't the whole story
- The biggest thing I would do to improve this is to implement image augmentations and tta
  - Would have to carefully explore whether geometric image distortions (warp, rotate, flip, zoom, etc) messed up the angle predictions. If so, would just use augmentations which did not geometrically distort the image (lighting, colors, etc)
- I didn't cv optimize pretty much any hyperparameters, I just found ones that worked very well and stuck with them. This may have helped slightly, but I think it was right to not spend time on this
  - The learning curves from training showed convergence and no overfitting, so the only thing I might have gotten was slight better accuracy
- This would have been a good use case for stacking or boosting together truly different models instead of just different CNN architectures