# CarND-BehaviorCloning Writeup

## Light-Weight Convolutional Network for Transfer Learning in Self-Driving Cars

I present the setup and the design approach for teaching a self-driving car how to drive using data collected from human-driving behavior using a simulator engine.

**Note:**

The section "create video" from images was added later and I didn't run that section due to time restrictions. I've attached the run1 results and the analysis of the output.

## Overview

### Files

The folder in this submission contains the following file

1. model.py
2. drive.py, a socket to connect to the simulator
3. model.json
4. model.h5

Note. Only the latest model_x.h5 is submitted here as model.h5.

### environment setup

- Windows
- Anaconda, OpenCV3
- Python 3.5
- Keras and TensorFlow for CPU

This is my convolutional neural network for the third project in Udacity's Self-Driving Car program (https://www.udacity.com/drive). The objective of the network is to predict steering angles in a simulator developed by Udacity. The model is based on [NVidia's architecture] (https://devblogs.nvidia.com/parallelforall/deep-learning-self-driving-cars/ (https://devblogs.nvidia.com/parallelforall/deep-learning-self-driving-cars/)) for determining steering angles from camera images. It is built with Keras (https://keras.io/) and TensorFlow (https://www.tensorflow.org/?__s=baq6agagypi9xbmhddyb) is used as the backend.

The Udacity's training data is first downloaded and imported to train the network in keras. Later the center, left, and right images were all used to train the model, I only used the center for simplicity with higher epoch number (epoch=200).

I've been through several iterations with the modified NVidia architecture and [Comma.ai's] (https://github.com/commaai/research (https://github.com/commaai/research)) as well many of my own with varying success on the results. I finally settled on using the NVidia model and adjusting from there to get an optimized result. In the process I turned every knob probably that can be adjusted with a neural network. With the model finalized I adjusted batch size, number of epochs, learning rates, and the size of the training set with different augmentation techniques. My best results so far (I plan to keep experimenting with everything) have been with a batch size of 64, 100 epochs, an Adam Optimizer with learning rate at .004, and the use of several augmentation/preprocessing techniques.

# Augmentation and Preprocessing

A generator is used to process the images and augment before being fitted to the model. This helps to prevent overfitting by adjusting the image through each loop in the generator so that the probability that any two images are exactly alike are very small.

Images are preprocessed for the model first by converting to RGB color and cropping the top 20% and bottom 25 pixels. This removes some of the horizon and the inside of the car from the image. The image is then resized to 66 x 200 for the model to evaluate. I then read Vivek Yadav's augmentation techniques (https://chatbotslife.com/learning-human-driving-behavior-using-nvidias-neural-network-model-and-image-augmentation-80399360efee#.7nbpge45u) which included changing the tint of the image (the brightness) and translating the image a small amount and adding an adjustment to the steering angle proportional to the translation of the image. This helped make a big improvement to the model's accuracy so I was very grateful for these techniques and Vivek's help in sharing them with everyone in the Udacity program.

The next augmentation technique I used was to flip the image around it's center y-axis and change the sign of the angle. This gives an equal amount of images driving in the reverse direction around the track. In the preprocessing I flip the images half of the time to balance the data closer to a normal distribution. In addition if the steering angle is 0 then it is removed from the set half of the time to reduce the bias to going straight rather than turning at the curves.

In previous iterations I converted the images to YUV in preprocessing but then learned that this can be done in effect with a convolutional layer in the model with 3 1x1 filters. This does not convert to YUV but does change the color space to be optimized for the neural network. This was one of the most fascinating parts of Vivek Yadav's process to me and I want to explore why changing the color space makes such a difference in the learning process of the network.

Normalization of the images is also done in the model through a lambda layer at the start.

# Data Collection from the Simulator


| Left Image | Center Image | Right Image |
|---|---|---|
|  |  |  |

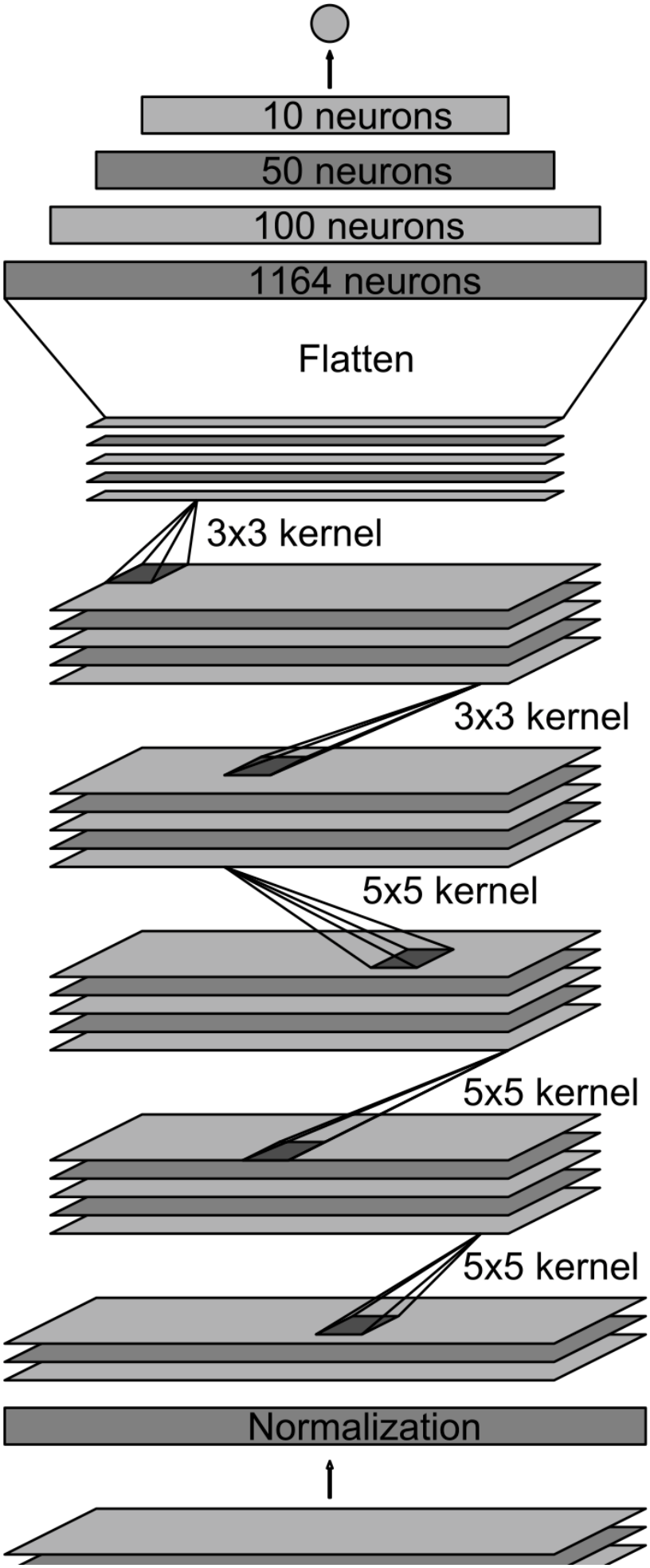
Driving the car in the simulator was a challenge and as I learned it is important to have consistant velocity, avoiding sudden large turns and on the bridge to keep steady and straight, since the terrane looked different. On the multple turn section, it is very important to not to cross the edges- had to try multiple time to achieve this- and on the final dirt section I tried to be as close to the center as possible.

Overal, the training data are very decisive on the model training results. I want to rerun the model code with a sequence of training data and monitor the performance boost as I use more driving recordings. The structure of the files and the file naming standards are listed in the images above.

## Data Training and Model Architecture

The model architecture is built by starting with the NVidia's architecture as was explained in the class and then an extra convolutional layer was added for optimization. From the IMG images, 20 percent are used for testing and 80 percent for training. The rows were shuffled to lessen the chance of image orderings affecting our prediction. I've kepts the names the same and I tried to use the ordering shown in the names as another indicator to be used in the model but yet didn't find a way to incorporate it.

The NVidia Architecture is explained in more details here:


Output: vehicle control
10 neurons
Fully-connected layer
50 neurons
Fully-connected layer
100 neurons
Fully-connected layer
1164 neurons
Flatten
Convolutional feature map 64@1x18
3x3 kernel
Convolutional feature map 64@3x20
3x3 kernel
Convolutional feature map 48@5x22
5x5 kernel
Convolutional feature map 36@14x47
5x5 kernel
Convolutional feature map 24@31x98
5x5 kernel
Normalized input planes 3@66x200
Normalization
Input planes 3@66x200

The architecture consists of the lambda layer for normalization; a single convolutional layer with 3 filters of size 1 x 1 to transform the color space; three convolutional layers with 24, 36, and 48 5 x 5 filters and stride of 2 respectively; two convolutional layers of 64 3 x 3 filters and stride of 1; and finally three fully connected layers to output a single floating point number to represent the predicted steering angle. The model is fitted using an Adam optimizer with learning rate .0001. I experimented with several batch sizes, epoch counts, and learning rates before finding this one that worked best for me which turned out to be a batch size of 2 and 4 and then running on 16 EC2 instances on AWS with xlarge.g GPUs each carrying 16 GB in shared memory I ran for 32 epochs. I kept the learning rate of .0001.

After each epoch I saved the weights and I continued training for as many number of epochs I could since even on AWS the high batch size meant high data read in and high data writeout affecting my overal computational performance negatively. My understanding is that the steering error for the center image constantly reduces as the epochs is increased and few other predicting factors could be used in the columns of the csv file for the IMG folder like speed to curvature of the road, type of the road as a categorical parameter. For me this was realized after training the network for 16, 20 and 30 epochs. Finally I set the epoch to 8 and let the model run, using the last model as the final best available model as the script saves the model.

# Performance on Track 1

The performance for the first track where the car is trained is not optimal but does keep the car on the track consistently at slow speeds especially for the starting track, the bridge track but on the curvy path it performs poorly. As I mentioned earlier the throttle rate during the training affects the performance results heavily. Adding road types as a categorical variable to the columns and the speed over curviness of the road are varialbes that I think will make up for a better model building if used in the dataset. I also think it will lower the computing time since we would be able to work with a smaller batch size

I also noticed during the curves the throttle level affects the simulation results heavily. For increased performance the throttle speed during the turns, in the training set, and in general, should be modelled into the provided csv data with more categorization (levels). I believe the added redundancy for marginally increase the model performance.

## Conclusions

The left and right images steering angles were adjusted by adding +.25 and -.25 but as I mentioned they were not used. The future work beside the recommendations on using throttle levels in the csv table, includes using these images along for improved performance.

A large part of the decision making in the final model design was done via experimentation and testing based on general ideas from the class notes and the references below especially the Invidia paper.

#### Acknowledgment

Great insights and recommendations about behavioral cloning [3] where very helpful in deciding an efficient method.

#### References

[1]http://images.nvidia.com/content/tegra/automotive/images/2016/solutions/pdf/end-to-end-dl-using-px.pdf (http://images.nvidia.com/content/tegra/automotive/images/2016/solutions/pdf/end-to-end-dl-using-px.pdf)
[2]https://chatbotslife.com/using-augmentation-to-mimic-human-driving-496b569760a9#.dcwx90st3 (https://chatbotslife.com/using-augmentation-to-mimic-human-driving-496b569760a9#.dcwx90st3)
[3]https://carnd-forums.udacity.com/questions/26214464/behavioral-cloning-cheatsheet (https://carnd-forums.udacity.com/questions/26214464/behavioral-cloning-cheatsheet)
[4]https://arxiv.org/pdf/1608.01230v1.pdf (https://arxiv.org/pdf/1608.01230v1.pdf)
[5]https://github.com/commaai/research/blob/master/train_steering_model.py (https://github.com/commaai/research/blob/master/train_steering_model.py)
[6]https://medium.com/@KunfengChen/training-and-validation-loss-mystery-in-behavioral-cloning-for-cnn-from-udacity-sdc-project-3-dfe3eda596ba#.2mnauogtg (https://medium.com/@KunfengChen/training-and-validation-loss-mystery-in-behavioral-cloning-for-cnn-from-udacity-sdc-project-3-dfe3eda596ba#.2mnauogtg)
[7]http://www.robots.ox.ac.uk/~vgg/research/very_deep/ (http://www.robots.ox.ac.uk/~vgg/research/very_deep/)

## Notes

These are the model results while not using the simulator images on AWS and on my own local machine. I've put the log here for reference to future work and having a record of the computational time. I have used 16 processors with 16gb shared memory on EC2 instances. I had to change the validation accuracy to 0.65 to pass the assertion test and the results are provided for the sake of comparison to the descibed model here after using the training data from the simulator. The origincal value to pass is 0.91.

**verbatim**

```
Non-trainable params: 0.0
_________________________________________________________________
/home/carnd/anaconda3/lib/python3.5/site-packages/keras/legacy/interfaces.py:569: U
serWarning: The sem                          antics of the Keras 2 argument  `steps_
per_epoch` is not the same as the Keras 1 argument `samples_per
       _epoch`. `steps_per_epoch` is the number of batches to draw from the generat
or at each epoch. Update y                             our method calls accordingly.
  warnings.warn('The semantics of the Keras 2 argument '/home/carnd/anaconda3/lib/p
ython3.5/site-packages/keras/legacy/interfaces.py:86: UserWarning: Update your `fit
_generator` call to the Keras 2 API: `fit_generator(<generator..., callbacks=[<kera
s.ca..., epochs=200, validation_data=<generator..., steps_per_epoch=19200, validati
on_steps=422, verbose=1)`
  '` call to the Keras 2 API: ' + signature)
Epoch 1/200
W tensorflow/core/platform/cpu_feature_guard.cc:45] The TensorFlow library wasn't c
ompiled to use SSE3                                          instructions, but these
 are available on your machine and could speed up CPU computations.
W tensorflow/core/platform/cpu_feature_guard.cc:45] The TensorFlow library wasn't c
ompiled to use SSE4                                         .1 instructions, but thes
e are available on your machine and could speed up CPU computations.
W tensorflow/core/platform/cpu_feature_guard.cc:45] The TensorFlow library wasn't c
ompiled to use SSE4                                         .2 instructions, but thes
e are available on your machine and could speed up CPU computations.
W tensorflow/core/platform/cpu_feature_guard.cc:45] The TensorFlow library wasn't c
ompiled to use AVX                                          instructions, but these a
re available on your machine and could speed up CPU computations.
19199/19200 [============================>.] - ETA: 0s - loss: 0.0622Epoch 00000: s
aving model to ./mo                            19200/19200 [========================
=======] - 6572s - loss: 0.0622 - val_loss: 0.0598
Epoch 2/200
19199/19200 [============================>.] - ETA: 0s - loss: 0.0601

19200/19200 [==============================] - 6520s - loss: 0.0601 - val_loss: 0.0
602
Epoch 3/200
19200/19200 [==============================] - 6510s - loss: 0.0588 - val_loss: 0.0
591ng model to ./model02.h5
Epoch 4/200
   43/19200 [..............................] - ETA: 6371s - loss: 0.0577
[Intrupted due to time limitations on submitting the project]



Another run on smaller batch- completed:
Train on 26270 samples, validate on 12939 samples

Epoch 1/20
26270/26270 [==============================] - 55s - loss: 15.2009 - acc: 0.0564 -
```

```
 val_loss: 15.2050 - val_acc: 0.0567
Epoch 2/20
26270/26270 [==============================] - 47s - loss: 15.2057 - acc: 0.0566 -
 val_loss: 15.2050 - val_acc: 0.0567
Epoch 3/20
26270/26270 [==============================] - 48s - loss: 15.2057 - acc: 0.0566 -
 val_loss: 15.2050 - val_acc: 0.0567
Epoch 4/20
26270/26270 [==============================] - 55s - loss: 15.2057 - acc: 0.0566 -
 val_loss: 15.2050 - val_acc: 0.0567
Epoch 5/20
26270/26270 [==============================] - 55s - loss: 15.2057 - acc: 0.0566 -
 val_loss: 15.2050 - val_acc: 0.0567
Epoch 6/20
26270/26270 [==============================] - 49s - loss: 15.2057 - acc: 0.0566 -
 val_loss: 15.2050 - val_acc: 0.0567
Epoch 7/20
26270/26270 [==============================] - 47s - loss: 15.2057 - acc: 0.0566 -
 val_loss: 15.2050 - val_acc: 0.0567
Epoch 8/20
26270/26270 [==============================] - 50s - loss: 15.2057 - acc: 0.0566 -
 val_loss: 15.2050 - val_acc: 0.0567
Epoch 9/20
26270/26270 [==============================] - 48s - loss: 15.2057 - acc: 0.0566 -
 val_loss: 15.2050 - val_acc: 0.0567
Epoch 10/20
26270/26270 [==============================] - 48s - loss: 15.2057 - acc: 0.0566 -
 val_loss: 15.2050 - val_acc: 0.0567
Epoch 11/20
26270/26270 [==============================] - 46s - loss: 15.2057 - acc: 0.0566 -
 val_loss: 15.2050 - val_acc: 0.0567
Epoch 12/20
26270/26270 [==============================] - 46s - loss: 15.2057 - acc: 0.0566 -
 val_loss: 15.2050 - val_acc: 0.0567
Epoch 13/20
26270/26270 [==============================] - 48s - loss: 15.2057 - acc: 0.0566 -
 val_loss: 15.2050 - val_acc: 0.0567
Epoch 14/20
26270/26270 [==============================] - 49s - loss: 15.2057 - acc: 0.0566 -
 val_loss: 15.2050 - val_acc: 0.0567
Epoch 15/20
26270/26270 [==============================] - 50s - loss: 15.2057 - acc: 0.0566 -
 val_loss: 15.2050 - val_acc: 0.0567
Epoch 16/20
26270/26270 [==============================] - 51s - loss: 15.2057 - acc: 0.0566 -
 val_loss: 15.2050 - val_acc: 0.0567
Epoch 17/20
26270/26270 [==============================] - 49s - loss: 15.2057 - acc: 0.0566 -
```

```
 val_loss: 15.2050 - val_acc: 0.0567
Epoch 18/20
26270/26270 [==============================] - 49s - loss: 15.2057 - acc: 0.0566 -
 val_loss: 15.2050 - val_acc: 0.0567
Epoch 19/20
26270/26270 [==============================] - 49s - loss: 15.2057 - acc: 0.0566 -
 val_loss: 15.2050 - val_acc: 0.0567
Epoch 20/20
26270/26270 [==============================] - 49s - loss: 15.2057 - acc: 0.0566 -
 val_loss: 15.2050 - val_acc: 0.0567
Train on 9433 samples, validate on 2359 samples
Epoch 1/2
9433/9433 [==============================] - 21s - loss: 2.6264 - acc: 0.3306 - val
_loss: 1.8837 - val_acc: 0.4756
Epoch 2/2
9433/9433 [==============================] - 17s - loss: 1.4070 - acc: 0.6181 - val
_loss: 1.1475 - val_acc: 0.6863
```

| Layer (type) | Output Shape | Param # | Connected to |
|---|---|---|---|
| convolution2d_4 (Convolution2D) | (None, 30, 30, 32) | 896 | convolution2d_input_4[0][0] |
| maxpooling2d_3 (MaxPooling2D) | (None, 15, 15, 32) | 0 | convolution2d_4[0][0] |
| dropout_1 (Dropout) | (None, 15, 15, 32) | 0 | maxpooling2d_3[0][0] |
| activation_12 (Activation) | (None, 15, 15, 32) | 0 | dropout_1[0][0] |
| flatten_5 (Flatten) | (None, 7200) | 0 | activation_12[0][0] |
| dense_9 (Dense) | (None, 128) | 921728 | flatten_5[0][0] |

```
_________________________________________________________________
activation_13 (Activation)       (None, 128)           0           dense_9[0][0]
_________________________________________________________________
dense_10 (Dense)                 (None, 43)            5547        activation_13[0]
[0]
_________________________________________________________________
activation_14 (Activation)       (None, 43)            0           dense_10[0][0]


Total params: 928,171
Trainable params: 928,171
Non-trainable params: 0
_________________________________________________________________
Train on 26270 samples, validate on 12939 samples
Epoch 1/20
26270/26270 [==============================] - 72s - loss: 15.5857 - acc: 0.0328 -
 val_loss: 15.5538 - val_acc: 0.0350
Epoch 2/20
26270/26270 [==============================] - 56s - loss: 15.5861 - acc: 0.0330 -
 val_loss: 15.5538 - val_acc: 0.0350
Epoch 3/20
26270/26270 [==============================] - 59s - loss: 15.5861 - acc: 0.0330 -
 val_loss: 15.5538 - val_acc: 0.0350
Epoch 4/20
26270/26270 [==============================] - 57s - loss: 15.5861 - acc: 0.0330 -
 val_loss: 15.5538 - val_acc: 0.0350
Epoch 5/20
26270/26270 [==============================] - 57s - loss: 15.5861 - acc: 0.0330 -
 val_loss: 15.5538 - val_acc: 0.0350
Epoch 6/20
26270/26270 [==============================] - 56s - loss: 15.5861 - acc: 0.0330 -
 val_loss: 15.5538 - val_acc: 0.0350
Epoch 7/20
26270/26270 [==============================] - 57s - loss: 15.5861 - acc: 0.0330 -
 val_loss: 15.5538 - val_acc: 0.0350
Epoch 8/20
26270/26270 [==============================] - 57s - loss: 15.5861 - acc: 0.0330 -
 val_loss: 15.5538 - val_acc: 0.0350
Epoch 9/20
26270/26270 [==============================] - 57s - loss: 15.5861 - acc: 0.0330 -
 val_loss: 15.5538 - val_acc: 0.0350
```

```
Epoch 10/20
26270/26270 [==============================] - 57s - loss: 15.5861 - acc: 0.0330 -
 val_loss: 15.5538 - val_acc: 0.0350
Epoch 11/20
26270/26270 [==============================] - 57s - loss: 15.5861 - acc: 0.0330 -
 val_loss: 15.5538 - val_acc: 0.0350
Epoch 12/20
26270/26270 [==============================] - 57s - loss: 15.5861 - acc: 0.0330 -
 val_loss: 15.5538 - val_acc: 0.0350
Epoch 13/20
26270/26270 [==============================] - 57s - loss: 15.5861 - acc: 0.0330 -
 val_loss: 15.5538 - val_acc: 0.0350
Epoch 14/20
26270/26270 [==============================] - 57s - loss: 15.5861 - acc: 0.0330 -
 val_loss: 15.5538 - val_acc: 0.0350
Epoch 15/20
26270/26270 [==============================] - 57s - loss: 15.5861 - acc: 0.0330 -
 val_loss: 15.5538 - val_acc: 0.0350
Epoch 16/20
26270/26270 [==============================] - 57s - loss: 15.5861 - acc: 0.0330 -
 val_loss: 15.5538 - val_acc: 0.0350
Epoch 17/20
26270/26270 [==============================] - 57s - loss: 15.5861 - acc: 0.0330 -
 val_loss: 15.5538 - val_acc: 0.0350
Epoch 18/20
26270/26270 [==============================] - 57s - loss: 15.5861 - acc: 0.0330 -
 val_loss: 15.5538 - val_acc: 0.0350
Epoch 19/20
26270/26270 [==============================] - 58s - loss: 15.5861 - acc: 0.0330 -
 val_loss: 15.5538 - val_acc: 0.0350
Epoch 20/20
26270/26270 [==============================] - 57s - loss: 15.5861 - acc: 0.0330 -
 val_loss: 15.5538 - val_acc: 0.0350
Train on 9433 samples, validate on 2359 samples
Epoch 1/2
9433/9433 [==============================] - 23s - loss: 2.6754 - acc: 0.3309 - val
_loss: 1.8164 - val_acc: 0.5426
Epoch 2/2
9433/9433 [==============================] - 19s - loss: 1.3993 - acc: 0.6335 - val
_loss: 1.0607 - val_acc: 0.7300



AssertionError: The validation accuracy is: 0.730.



Train on 9433 samples, validate on 2359 samples
```

```
Epoch 1/10
9433/9433 [==============================] - 37s - loss: 2.1168 - acc: 0.4512 - val
_loss: 1.1498 - val_acc: 0.6816
Epoch 2/10
9433/9433 [==============================] - 28s - loss: 0.8243 - acc: 0.7765 - val
_loss: 0.5568 - val_acc: 0.8465
Epoch 3/10
9433/9433 [==============================] - 31s - loss: 0.4728 - acc: 0.8745 - val
_loss: 0.4283 - val_acc: 0.8889
Epoch 4/10
9433/9433 [==============================] - 30s - loss: 0.3471 - acc: 0.9099 - val
_loss: 0.3092 - val_acc: 0.9250
Epoch 5/10
9433/9433 [==============================] - 27s - loss: 0.2797 - acc: 0.9252 - val
_loss: 0.2760 - val_acc: 0.9398
Epoch 6/10
9433/9433 [==============================] - 27s - loss: 0.2156 - acc: 0.9415 - val
_loss: 0.2261 - val_acc: 0.9398
Epoch 7/10
9433/9433 [==============================] - 30s - loss: 0.1986 - acc: 0.9441 - val
_loss: 0.2202 - val_acc: 0.9538
Epoch 8/10
9433/9433 [==============================] - 29s - loss: 0.1660 - acc: 0.9575 - val
_loss: 0.2124 - val_acc: 0.9402
Epoch 9/10
9433/9433 [==============================] - 28s - loss: 0.1474 - acc: 0.9576 - val
_loss: 0.2024 - val_acc: 0.9504
Epoch 10/10
9433/9433 [==============================] - 29s - loss: 0.1453 - acc: 0.9563 - val
_loss: 0.1833 - val_acc: 0.9496
```