Machine Learning Engineer Nanodegree

Capstone Project

Luciano Silveira
January, 2020

I. Proposal

Refer to the proposal here.

Domain Background

Machine Learning is starting to touch all aspects of our life, in particular it has a lot of potential for automation in robotic applications.

Being part of the donkeycar2 robocar community, an opensource do it yourself self-driving platform for small scale cars; there are many possibilities for exploration in that space.

In this context, the exploration and experimentation has been done mainly with `supervised learning` and the community is starting to play around with `reinforcement learning` (RL) algorithms.

The project objective is to implement a classification model for the robocar to detect where it is in a specific race track so the model can be used for other high level models. In particular, it would be useful for `reinforcement learning` and to explore how to autonomously navigate a track withing the specified lane. That is, to implement a classification system for a robotic application, building an inference engine extracting information from a race track.

Problem Statement

In the last few year the development of robotics applications using Machine Learning framework has exploded. This project explores the usage of different Machine Learning models to be used as a inference engine for classification; a classifier to detect a robot within a track and the possible actions to do: go `Straight` , turn `Left` or `Right` . That is to say identify 3 possible classes:

- Straight or Center of the track
- Left side of the track
- Right side of the track

Metrics

To judge the performance of the model, the metric selected is `Classification Accuracy` ; which calculates how often predictions matches the labels meaning the percentage of correct preductions

```
Accuracy = number of correct predictions / total number of predictions
* 100
```

Other important metrics detailed in the proposal are `Precision` and `Recall` but were discarded. There are many cases in which `Classification Accuracy` is not a good indicator. One case is when the class distribution is imbalanced (one class is more frequent than others) and other metrics are needed. In this scenario (as detailed in Section II) the dataset is manually built-up and precautions were taken to generate relatively balanced classes.

Some related links:

- Classification & Regression Evaluation Metrics

II. Analysis

Datasets and Inputs

For this phase a robocar was run to follow a standard race track in recording mode. After a few laps, it recorded hundreds of 160x120 RGB images. The sample track is as follows:

The training and test data was manually classified and organized in the following classes: `Straight`, `Left`, `Right`; the folder structure:

```
data/
├── Left/
│   ├── 47_cam-image_array_.jpg
│   ├── ...
│   └── 48_cam-image_array_.jpg
├── Right/
│   ├── 46_cam-image_array_.jpg
│   ├── ...
│   └── 47_cam-image_array_.jpg
└── Straight/
    ├── 25_cam-image_array_.jpg
    ├── ...
    └── 26_cam-image_array_.jpg
```

The images distribution by class is:

- Left: 702 images
- Right: 719 images

- Straight: 841 images

Some examples:

Right

Left

Straight

Classes by definition are not balanced, in general when normally driving there will be more samples from `Straight` than the `Left`, `Right` cases; special emphasis must be done to get more data from the missing classes. In particular, our track is almost an oval so it is important to note that we needed to run the robocar in two ways to get equal number of `right` and `left` turns.

Some examples of the type of images from the simulator (Left, Straight, Right):

All data is within the data folder.

Benchmark Model

It was not possible to benchmark this work against existing ones. The author is aware that this kind of research is being done for reinforcement learning algorithms, as `helper methods` so as an agent can get additional information about the environment; for the case to know the location of the robocar within the track, for example similar to the available parameters in the Deepracer competition where a reward function may be programmed to encourage the agent to stay as close to the center line as possible.

Taking ideas from Evaluate the Performance Of Deep Learning Models in Keras, after training each model the average and standard deviation of the model `Accuracy` and `Loss` are printed and later compared to each other.

Some related links:

- Paper Self-driving scale car trained by Deep reinforcement Learning
- AWS DeepRacer Reward Function
- Semantic segmentation by NVidia

III. Methodology

The project was divided in the following steps:

- Data gathering and manual classification
- Framework exploration
- Training and validation
- Summarization of the results

Data Preprocessing

Manually classify the images in 3 classes: `Straight`, `Left`, `Right` and divide the resultset into two groups: training and valudation.

Augument data to get more information for the training process to avoiding overfitting. Some ideas to explore are:

- grayscale: convert the image to grayscale.
- Augmentation: Generate random changes on images, such as modification of contranst and coloring to have an effect in the lighting conditions.
    - Horizontal flip of the image
- Cropping: remove not useful parts of the image to concentrate on an specific region of interest (ROI).
- Normalization: rescale the image data from values between 0 and 255 to values between 0 and 1, as neural networks prefer to deal with small input values.

Detail on this exploration can be chacked on the DataManagement jupyter notebook.

Some links on these steps:

- How to prepare/augment images for neural network?
- Introduction to Dataset Augmentation and Expansion
- AutoML for Data Augmentation

Data Exploration

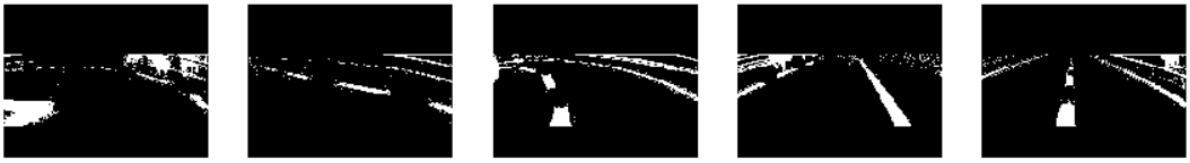
The following analysis was done on the images.

In general, for the problem we are trying to solve only the bottom part of the image is relevant for classification, so we select a region of interest

and cropped the image to get only the neccesary bits.

An alternative analyisis is to change the image to grayscale to remove the 3 RGB channels and only keep one channel:

Plus some analysis on edge detection using Canny Edge detection



and thesholding it to find the continuous relevant edges. Finally the selection is to keep the 3 RGB channels but just center on the selected region of interest.

This section was valiadted with the Data Management jupyter notebook.

Input Image Structure

The image provided by the simulator and in runtime is a RGB encoded height:120px and width:160.

As the camera is in a fixed position and the higher part of it is useless for the classification task, it is cropped the upper part of it and just feed the network with a smaller Image containing only lane information. For example (Straight cropped image):

Framework Exploration

Based on the Build an Image Dataset in TensorFlow article, differnt code sections were developed to evaluate different training alternatives.

A detail analysis can be checked on the Classification And Evaluation jupyter notebook.

Refinement

The following classifiers were created:

- Version 1
- Version 2
- Version 3

Version #1

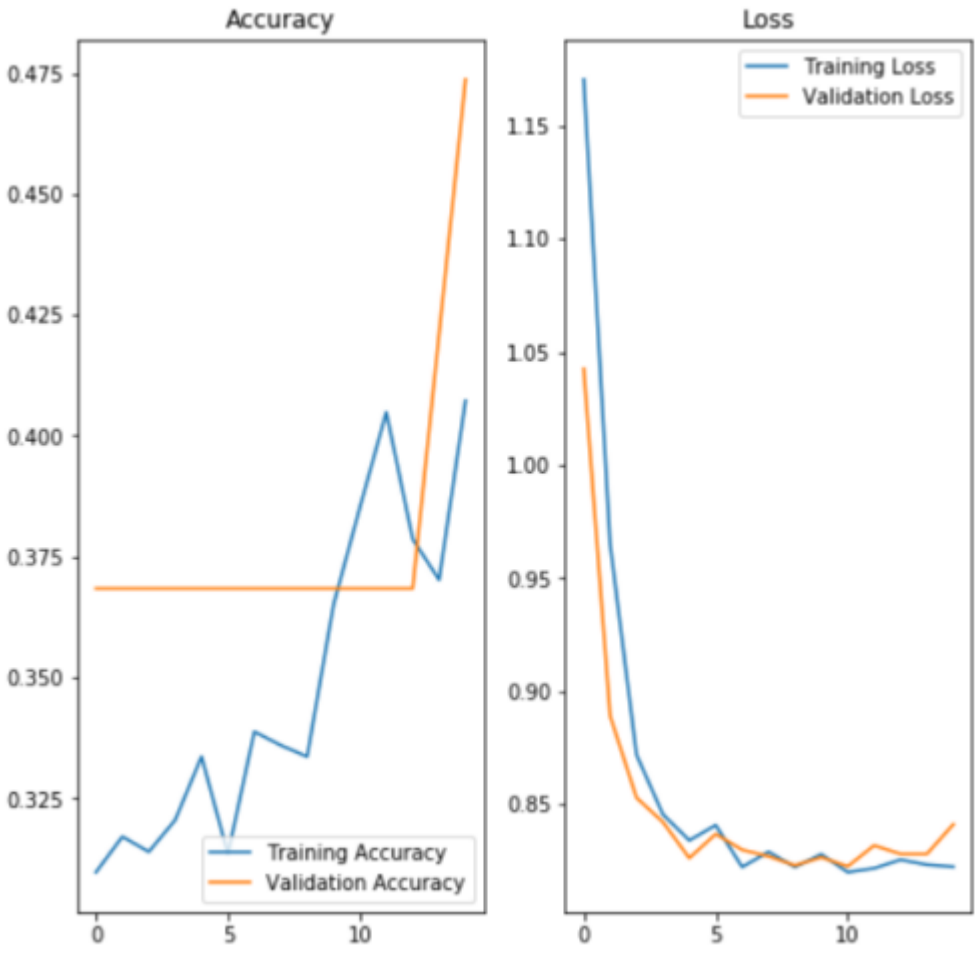
Initial version for a classifier; the objective was to create a working pipeline:

- read the correct images
- separete them for train and validation
- train a classifier
- evaluate results

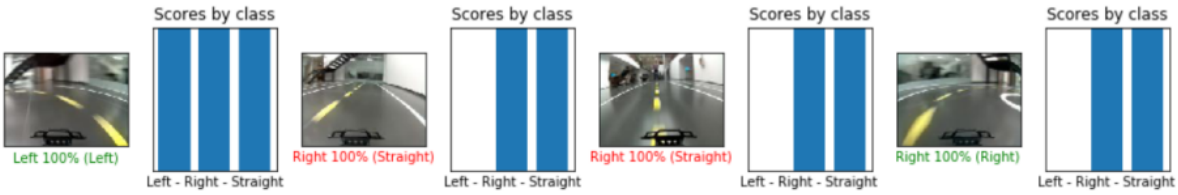
The model consists of a couple of Convolutions and Fully connected layer to output 3 classes using the *ADAM* optimizer and *categorical cross entropy* loss function, taking into account *accuracy* as metric for evaluation.

```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 120, 160, 16)      448
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 60, 80, 16)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 60, 80, 16)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 60, 80, 32)        4640
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 30, 40, 32)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 30, 40, 64)        18496
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 15, 20, 64)        0
_________________________________________________________________
dropout_2 (Dropout)          (None, 15, 20, 64)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 19200)             0
_________________________________________________________________
dense_1 (Dense)              (None, 512)               9830912
_________________________________________________________________
dense_2 (Dense)              (None, 3)                 1539
=================================================================
Total params: 9,856,035
Trainable params: 9,856,035
Non-trainable params: 0
```

The result of the training for 15 epochs is detailed as follows:



And validation of information was created to check the associated result



The image details that out of 4 random samples; 2 were correctly classified.

Version #2

For the following iteration, the objective was to use a similar CNN from the robocars project, change the output to be a classifier.

This CNN is deeper than the previous version; it uses 5 layers of Convolutions interleaving with Dropout and finally a series of Fully connected layer plus randomly dropout 10% of the neurons to prevent overfitting. Finally, assign classification percentages for 3 classes.

```
Layer (type)                 Output Shape              Param #
=================================================================
```

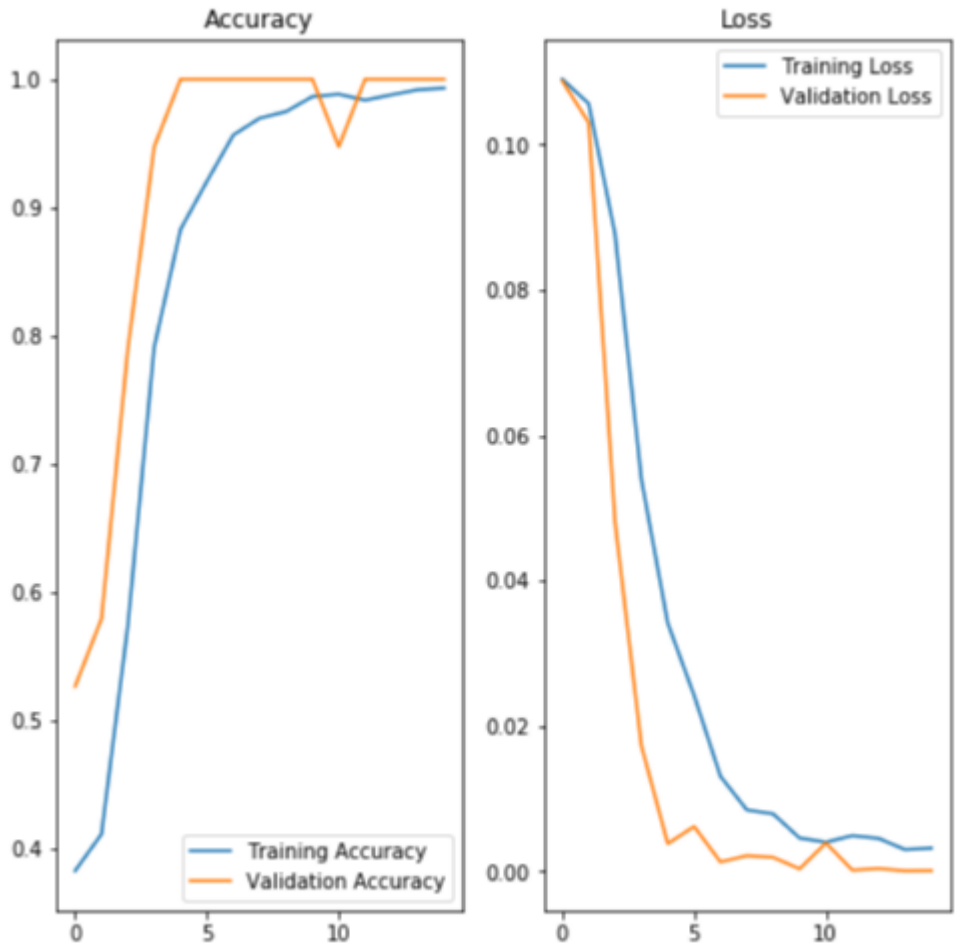
```
img_in (InputLayer)          (None, 120, 160, 3)       0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 58, 78, 24)        1824
_________________________________________________________________
dropout_3 (Dropout)          (None, 58, 78, 24)        0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 27, 37, 32)        19232
_________________________________________________________________
dropout_4 (Dropout)          (None, 27, 37, 32)        0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 12, 17, 64)        51264
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 5, 8, 64)          36928
_________________________________________________________________
dropout_5 (Dropout)          (None, 5, 8, 64)          0
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 3, 6, 64)          36928
_________________________________________________________________
dropout_6 (Dropout)          (None, 3, 6, 64)          0
_________________________________________________________________
flattened (Flatten)          (None, 1152)              0
_________________________________________________________________
dense_3 (Dense)              (None, 100)               115300
_________________________________________________________________
dropout_7 (Dropout)          (None, 100)               0
_________________________________________________________________
dense_4 (Dense)              (None, 50)                5050
_________________________________________________________________
dropout_8 (Dropout)          (None, 50)                0
_________________________________________________________________
class_out (Dense)            (None, 3)                 153
=================================================================
Total params: 266,679
Trainable params: 266,679
Non-trainable params: 0
```

The CNN was modified as follows:

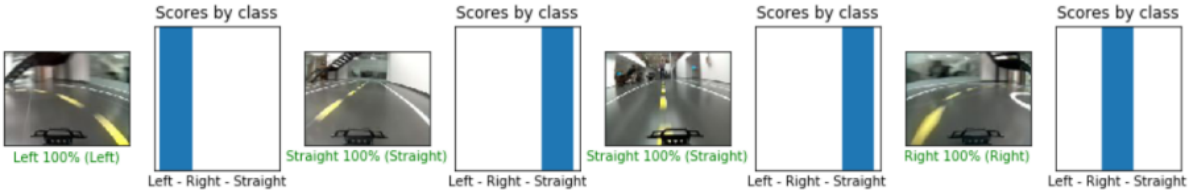
```
#Class output
class_out = Dense(class_count, activation='softmax', name='class_out')(x

model = Model(inputs=[img_in], outputs=[class_out])
model.compile(optimizer='adam',
                        loss={'class_out': 'categorical_crossentropy'}
                        loss_weights={'class_out': 0.1},
                        metrics=['accuracy'])
return model
```

The result of the training is detailed as follows:



And better validation of results:



Version #3

For the next iteration real experimentation took place. The following changes where applied to prevent overfitting the CNN:

- Reduce the capacity of the network
- Add weight regularization
- Add Batch Normalization
- Modify the dropout

Ideas taken from Overfit and Underfit

Dropout

A technique to reduce overfitting is to apply dropout to the CNN. It is a form of regularization that forces the weights in the network to take only small values, which makes the distribution of weight values more regular and the network can reduce overfitting on small training examples. Dropout is one of the regularization technique applied. The following values were tested, 0.3 was selected.

```
drop = 0.3 #0.1 # 0.2 # 0.4
```

When appling 0.1 dropout to a certain layer, it randomly kills 10% of the output units in each training epoch. This means dropping out 10%, 20% or 40% of the output units randomly from the applied layer, at the ends makes a more robust network. The case was applied to different convolutions and fully-connected layers.

References on this section:

- Build an Image Dataset in TensorFlow
- Image classification

Region Of Interest and Salience

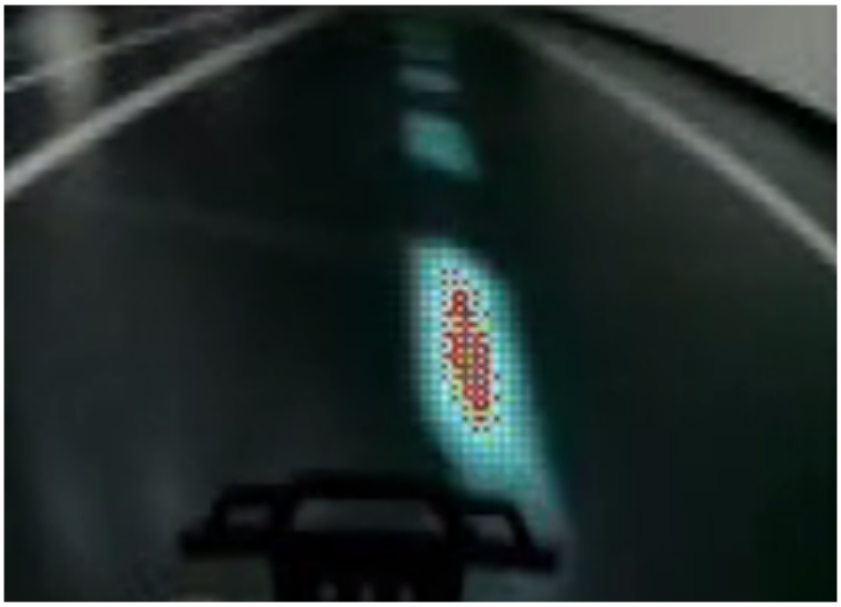
Based on the data exploration done in DataManagement; it was trimmed the higher part of the image:

```
x = Cropping2D(cropping=((40,0), (0,0)))(x)
```

The decision to use this feature is related to the *Salience* analysis for the track. The goal of this kind of visualization is to understand what learns and how a CNN makes its decisions. The central idea in discerning the salient objects is finding parts of the image that correspond to locations where the feature maps of CNN layers have the greatest activations. Based on the keras-salient-object-visualisation project; we tested a couple of images using a saliency heatmap.

Salience Heatmap for the whole image

Salience Heatmap ROI image



Notice that the second section has a much stronger focus on the lane lines and everything else is discarded as is not needed as input for the classification task.

References on this section:

- Make Movie from Tub with optional --salient will overlay a visualization of which pixels excited the NN the most.
- Original paper

Batch Normalization

Applied the ideas from Normalization; apply a transformation that maintains the mean activation close to 0 and the activation standard deviation close to 1.

```
x = BatchNormalization(input_shape=default_shape)(x)
```

Keras Improvements

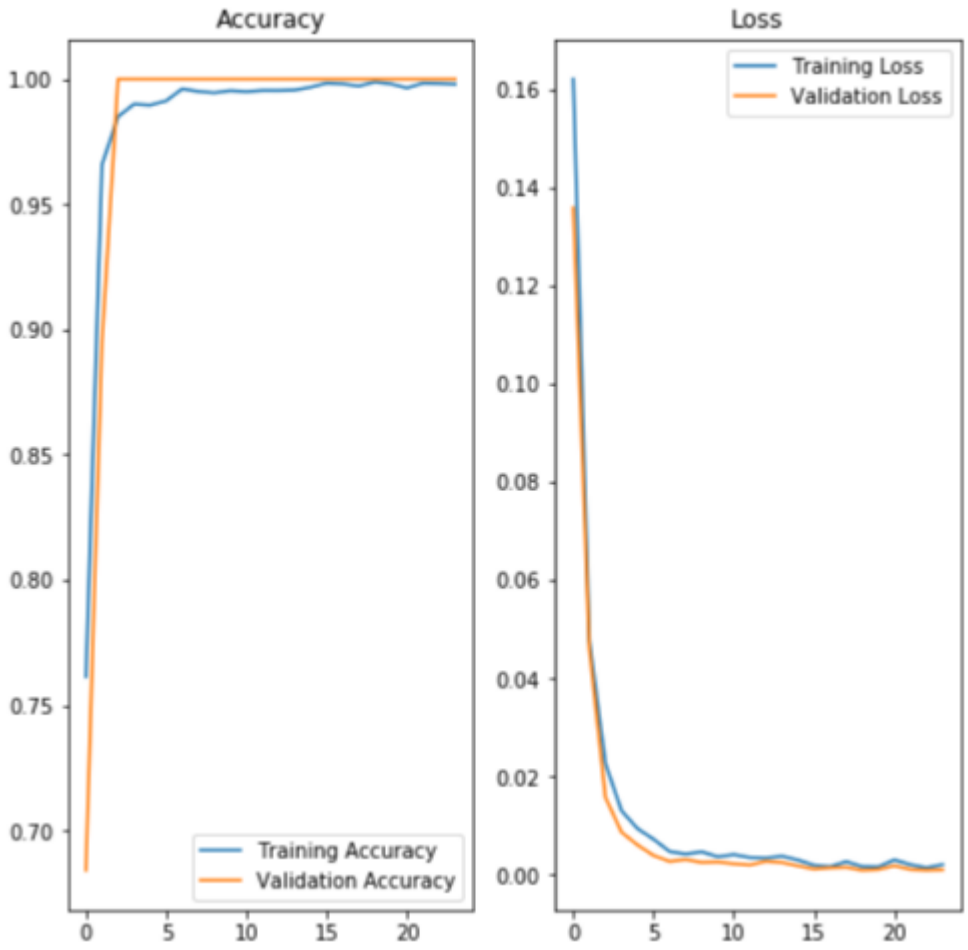
Besides, several improvements related to the Keras framework were added:

- Save a model checkpoint
- Early stopping when the training is not improving after 7 epochs.

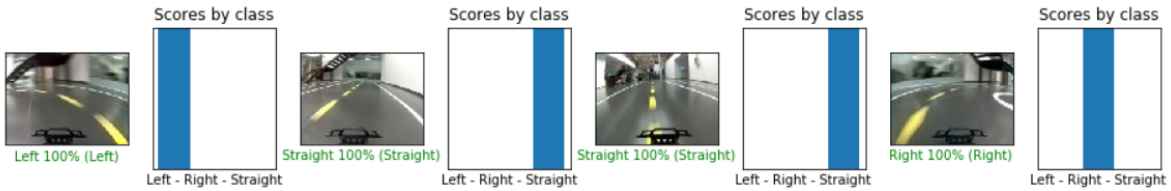
```
#checkpoint to save model after each epoch
save_best = keras.callbacks.ModelCheckpoint(saved_model_path,
                                            monitor=monitor,
                                            verbose=verbose,
                                            save_best_only=True,
                                            mode='min')

#stop training if the validation error stops improving.
early_stop = keras.callbacks.EarlyStopping(monitor=monitor,
                                           min_delta=min_delta,
                                           patience=patience,
                                           verbose=verbose,
                                           mode='auto')
```

The result of the training is detailed as follows:



And better validation of results were obtained:



IV. Results

Model Evaluation and Validation

Comparing the 3 CNNs created, the Accuracy and Loss streadily improved in every iteration.

Measure	#1	#2	#3
Train accuracy	0.32%	0.89%	0.98%
Test accuracy	.45%	0.95%	0.98%
Train loss	0.59%	0.02%	0.02%
Test loss	0.58%	0.01%	0.01%

To verify the robustness of the final model, a test was conducted using a video taken from the same track. For the pipeline evaluation, we executed the video taking frame by frame and checked the class assigned; some samples are:

The complete video:

video track result

the following observations are based on this test:

- Classification errors: in several sections of the track a wrong classification class was assigned.

Justification

The final architecture and hyperparameters are those related to the `Version #3` because they performed the best among the tried combinations. Some comments on the important parts:

- Cropping resulted to be very important once the Salience analysis was done.
- Dropout worked better in reducing overfitting. The procedure to interleave it in the architecture was basically try and error and check the training accuracy and loss graphs.
- Keras Callbacks were very useful not to continue training a model if it does not improve after a couple of epochs.

V. Conclusion

The model satisfactory met the given requirements. This model will only work on the specified track. A test was done to use the classifier on the simulator data but the result was very discorauging:

The simulation video result.

Reflection

The process used for this project can be summarized using the following steps:

- Research and Frame a relevant problem based on the knowledge acquired.
- Generate a dataset for the problem
- Manually segment the information
- Create a benchmark for the classifier
- Train the classifier using the sample data multiple times until a good set of parameters were found.
- Reviewed code from differente sources: Tensorflow, Keras, Donkeycar repositories and create a specific pipeline.

- Create own metrics and visualization for evaluation.

Manually tuning the model needs carefully taking note of the parameters, evaluate, measure and readjust.

The use of Salience analysis was important to improve the classifier and remove parts of the image that were not useful for the selected task.

Initially, I got encouraging results but not enough to be used as a classification system. The dataset should be increased and polished and trained again. Some extra preprocessing can be done to the images; so as to improve it's classification:

- change the image color scheme to HSV, Gray or a combination.
- change the image field of view with a birds view transformation to better aply the classification.

Improvement

- Get more training data and classify it, mix simulation data to create a more generalized model.
- data-augmentation; use horizontal flip to auto-generate more `Left` and `Right` samples.
- Evaluate an alternative training pipeline with `Transfer Learning` (also known as `Behavioral Cloning`). Select and existing models for image classification with weights already trained: (for example ImageNet Model Zoo for TensorFlow or Keras applications) and retrain the classifier for our purposes.

Links

- Donkeycar Github repository
- Train autopilot docs
- Reinforcement Learning
- watermark
- Save and Restore Models
- Techniques to Tackle Overfitting and Achieve Robustness for Donkey Car Neural Network Self-Driving Agent
- Keras
- Training sandbox
- Project Rubric

How to reproduce the results

Tested with:

```
Tensorfow 1.8.0
Keras 2.0.8
Numpy 1.14.3
Matplotlib 2.0.2
Moviepy 0.2.3.2
IPython 6.4.0
import 3.2.0
```

Unzip the following files:

```
data/Training.zip
data/Validation.zip
```

Use the following Jupyter notebooks:

- Data Management
- Classification And Evaluation