Element 5 : Steering Wheel Angle Detection

Abstact :

Over the recent years, the race towards the development of fully autonomous vehicles has accelerated tremendously. Many components make up an autonomous vehicle, and some of its most critical ones are the sensors and AI software that powers it. Moreover, with the increase in computational capabilities, we are now able to train complex and deep neural networks that are able to learn crucial details, visual and beyond, and become the brain of the car, understanding the vehicle's environment and deciding on the next decisions to take.

In this Element, we are going to cover how we can train a deep learning model to predict steering wheel angles and help a virtual car drive itself in a simulator. The model is created using Keras, relying on Tensorflow as the backend.

Phases of the Project

We divided this project into -- phases

1. Loading the Dataset, Serialisation and Marshalling of Labels
2. Training the Model
3. Autopilot

Phase 1 : Loading the Dataset

Dataset Description

Our dataset consist of 2 parts.

1. Images
2. Labels

There are 64000 images and their labels. The image file consist of on feild images and the data log is saved in a data.txt file and contains the path to the images as well as steering wheel angle, throttle and speed. We are only concerned with the steering wheel angle and the images for this project.

Data Collection:

Training data was collected by driving on a wide variety of roads and in a diverse set of lighting and weather conditions. Most road data was collected in central New Jersey, although highway data was also collected from

- Illinois
- Michigan
- Pennsylvania
- New York.

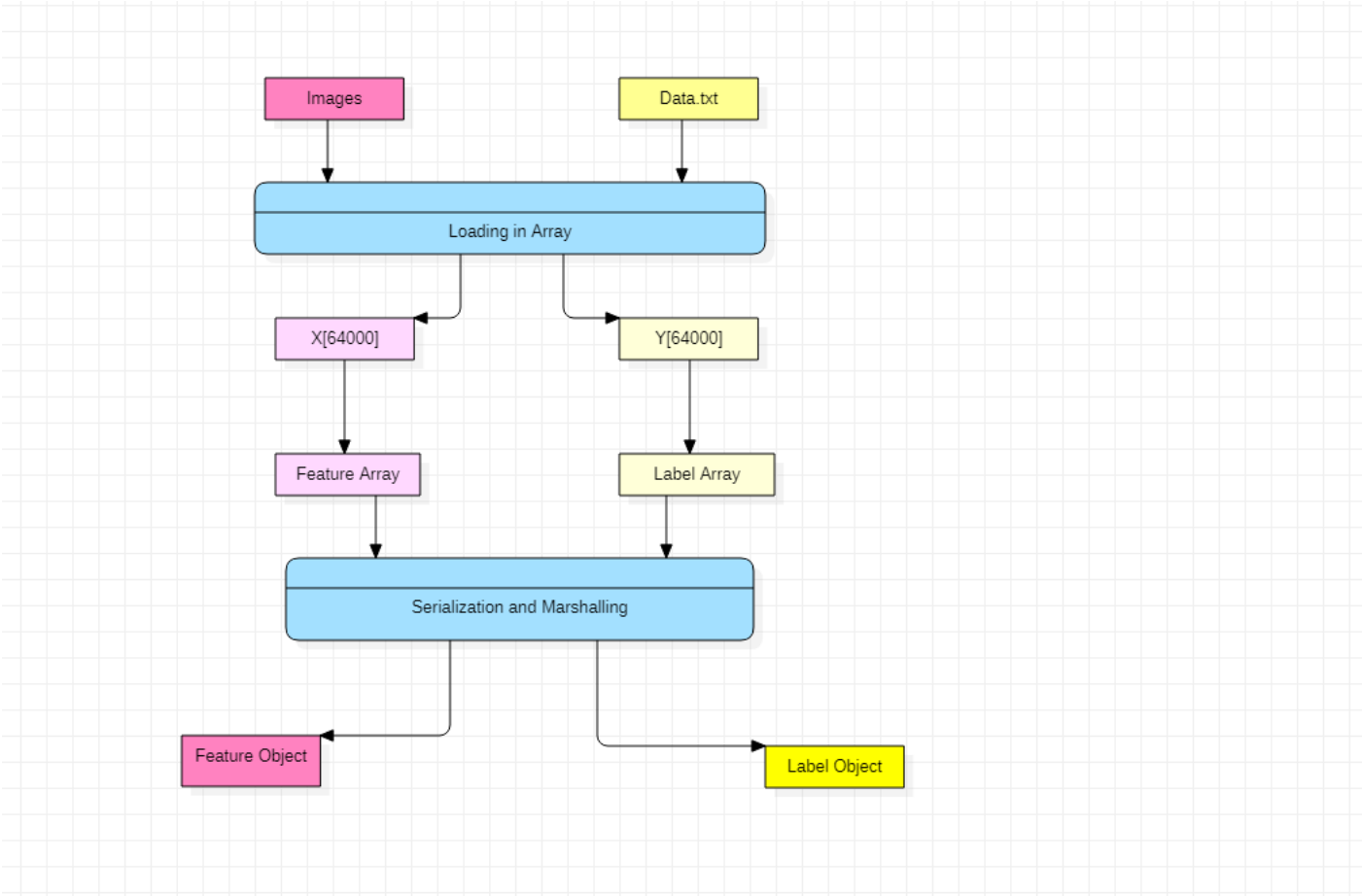
Other road types include two-lane roads (with and without lane markings), residential roads with parked cars, tunnels, and unpaved roads. Data was collected in clear, cloudy, foggy, snowy, and rainy weather, both day and night. In some instances, the sun was low in the sky, resulting in glare reflecting from the road surface and scattering from the windshield. Data was acquired using either our drive-by-wire test vehicle, which is a 2016 Lincoln MKZ, or using a 2013 Ford Focus with cameras placed in similar positions to those in the Lincoln. The system has no dependencies on any particular vehicle make or model. Drivers were encouraged to maintain full attentiveness, but otherwise drive as they usually do. As of March 28, 2016, about 72 hours of driving data was collected.

Importing Libraries (Phase 1)

The following Libraries are used in Phase 1 :

- `OpenCV` : OpenCV provides a real-time optimized Computer Vision library, tools, and hardware. It also supports model execution for Machine Learning and Artificial Intelligence.
- `os` : The OS module in Python provides functions for interacting with the operating system. This module provides a portable way of using operating system-dependent functionality.
- `numpy` : NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays.
- `scipy` : SciPy is a free and open-source Python library used for scientific computing and technical computing. SciPy contains modules for optimization, linear algebra, integration, interpolation, special functions, FFT, signal and image processing, ODE solvers and other tasks common in science and engineering.
- `pickle` : Python pickle module is used for serializing and de-serializing a Python object structure. Any object in Python can be pickled so that it can be saved on disk. What pickle does is that it “serializes” the object first before writing it to file. Pickling is a way to convert a python object (list, dict, etc.) into a character stream. The idea is that this character stream contains all the information necessary to reconstruct the object in another python script.
- `matplotlib` : Matplotlib is a plotting library for the Python programming language and its numerical mathematics extension NumPy.
- `itertools` : Itertools is a module in python, it is used to iterate over data structures that can be stepped over using a for-loop. Such data structures are also known as iterables. This module incorporates functions that utilize computational resources efficiently.

In [1]:
```
from __future__ import division
import cv2
import os
import numpy as np
import scipy
import pickle
import matplotlib.pyplot as plt
from itertools import islice
```


Images
Data.txt
Loading in Array
X[64000]
Y[64000]
Feature Array
Label Array
Serialization and Marshalling
Feature Object
Label Object

In [3]:

```
from __future__ import division
import cv2
import os
import numpy as np
import scipy
import pickle
import matplotlib.pyplot as plt
from itertools import islice
http://localhost:8888/notebooks/Desktop/SDC/Phase1.png
LIMIT = None

DATA_FOLDER = 'driving_dataset'
TRAIN_FILE = os.path.join(DATA_FOLDER, 'data.txt')

def preprocess(img):
    resized = cv2.resize((cv2.cvtColor(img, cv2.COLOR_RGB2HSV))[:, :, 1], (100
, 100))
    return resized

def return_data():

    X = []
    y = []
    features = []

    with open(TRAIN_FILE) as fp:
        for line in islice(fp, LIMIT):
            path, angle, _ = line.strip().split(' ')
            angle = angle.split(',')[0]
            full_path = os.path.join(DATA_FOLDER, 'data\\'+path)
            X.append(full_path)
            # using angles from -pi to pi to avoid rescaling the atan in the n
etwork
            y.append(float(angle) * scipy.pi / 180)

    for i in range(len(X)):
        img = plt.imread(X[i])
        features.append(preprocess(img))

    features = np.array(features).astype('float32')
    labels = np.array(y).astype('float32')

    with open("features", "wb") as f:
        pickle.dump(features, f, protocol=4)
    with open("labels", "wb") as f:
        pickle.dump(labels, f, protocol=4)

return_data()
```

Phase 2 : Training the driver

Method :

We trained a convolutional neural network (CNN) to map raw pixels from a single front-facing camera directly to steering commands. This end-to-end approach proved surprisingly powerful. With training data from humans the system learns to drive in traffic on local roads with or without lane markings and on highways. It also operates in areas with unclear visual guidance such as in parking lots and on unpaved roads.

The system automatically learns internal representations of the necessary processing steps such as detecting useful road features with only the human steering angle as the training signal. We never explicitly trained it to detect, for example, the outline of roads. Compared to explicit decomposition of the problem, such as lane marking detection, path planning, and control, our end-to-end system optimizes all processing steps simultaneously. We argue that this will eventually lead to better performance and smaller systems. Better performance will result because the internal components self-optimize to maximize overall system performance, instead of optimizing human-selected intermediate criteria, e.g., lane detection. Such criteria understandably are selected for ease of human interpretation which doesn’t automatically guarantee maximum system performance. Smaller networks are possible because the system learns to solve the problem with the minimal number of processing steps.

Importing Libraries : Phase 2

We are using the following Libraries in this phase :

1. Tensorflow :

TensorFlow is a free and open-source software library for machine learning. It can be used across a range of tasks but has a particular focus on training and inference of deep neural networks. Tensorflow is a symbolic math library based on dataflow and differentiable programming.

2. Keras :

Keras is an open-source software library that provides a Python interface for artificial neural networks. Keras acts as an interface for the TensorFlow library.

3. Numpy :

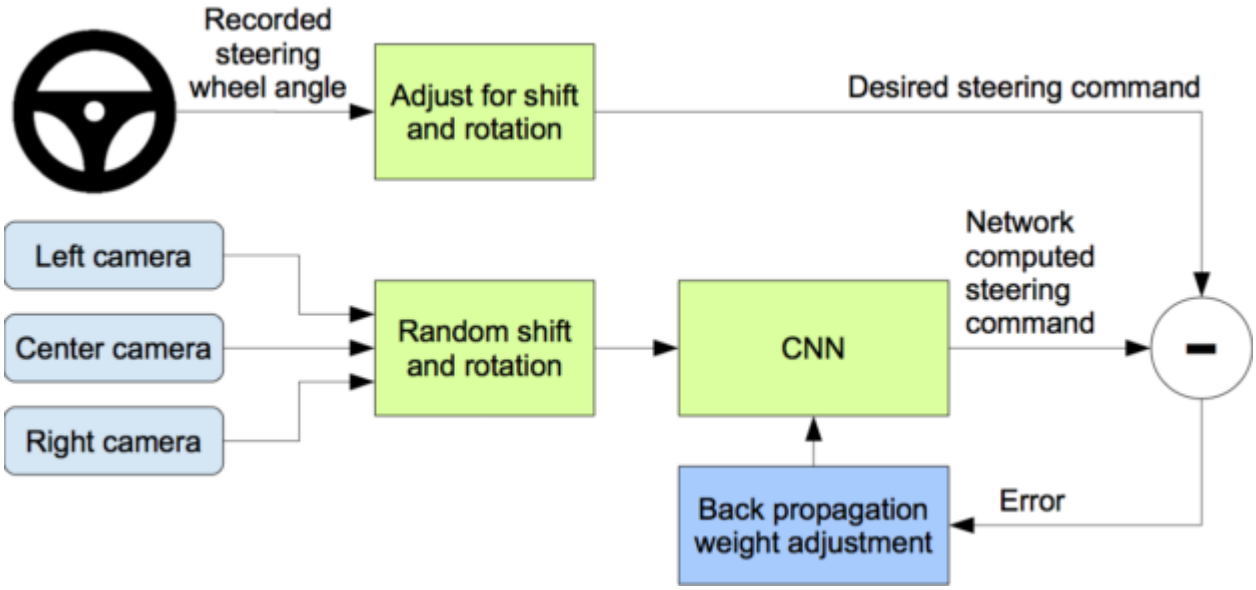
Keras is an open-source software library that provides a Python interface for artificial neural networks. Keras acts as an interface for the TensorFlow library.

4. Scikit :

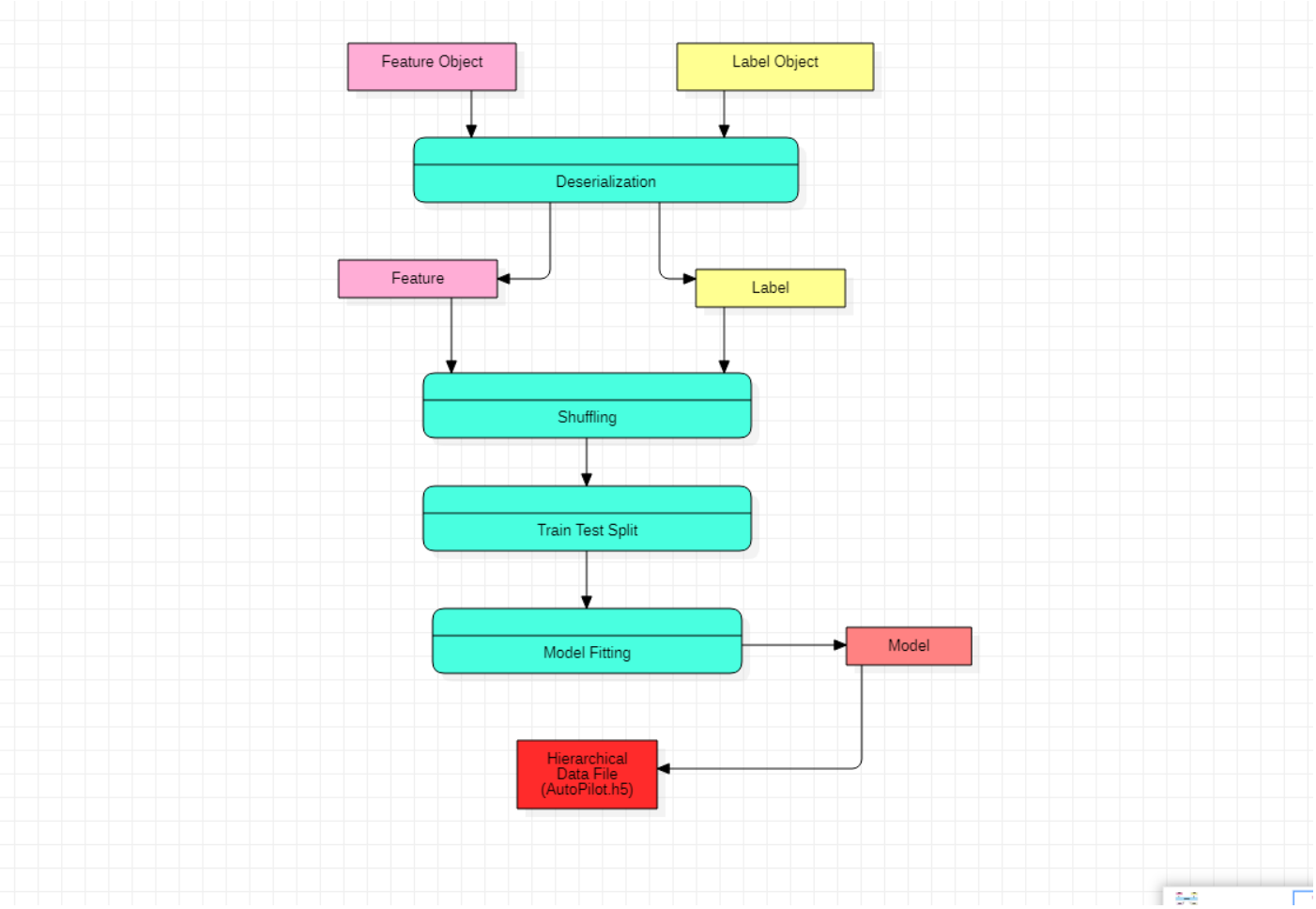
Scikit-learn is a free software machine learning library for the Python programming language. It features various classification, regression and clustering algorithms including support vector machines, random forests, gradient boosting, k-means and DBSCAN, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy.

In [7]:
```
import numpy as np
from keras.layers import Dense, Activation, Flatten, Conv2D, Lambda
from keras.layers import MaxPooling2D, Dropout
from keras.utils import print_summary
from keras.models import Sequential
from keras.callbacks import ModelCheckpoint
import keras.backend as K
import pickle
```

We will be training a convolutional neural network to predict steering wheel angles based on steering angle data and images captured by three cameras (left, center, right) mounted in front of the car. The trained model is able to accurately steer the car using only the center camera. The diagram below shows the process used to create such an efficient model.



Images are fed into a CNN which then computes a proposed steering command. The proposed command is compared to the desired command for that image and the weights of the CNN are adjusted to bring the CNN output closer to the desired output. The weight adjustment is accomplished using back propagation


Feature Object
Label Object
Deserialization
Feature
Label
Shuffling
Train Test Split
Model Fitting
Model
Hierarchical Data File (AutoPilot.h5)

In [8]:

```
def keras_model(image_x, image_y):
    model = Sequential()
    model.add(Lambda(lambda x: x / 127.5 - 1., input_shape=(image_x, image_y,
1)))
    model.add(Conv2D(32, (3, 3), padding='same'))
    model.add(Activation('relu'))
    model.add(MaxPooling2D((2, 2), padding='valid'))
    model.add(Conv2D(32, (3, 3), padding='same'))
    model.add(Activation('relu'))
    model.add(MaxPooling2D((2, 2), padding='valid'))

    model.add(Conv2D(64, (3, 3), padding='same'))
    model.add(Activation('relu'))
    model.add(MaxPooling2D((2, 2), padding='valid'))
    model.add(Conv2D(64, (3, 3), padding='same'))
    model.add(Activation('relu'))
    model.add(MaxPooling2D((2, 2), padding='valid'))

    model.add(Conv2D(128, (3, 3), padding='same'))
    model.add(Activation('relu'))
    model.add(MaxPooling2D((2, 2), padding='valid'))
    model.add(Conv2D(128, (3, 3), padding='same'))
    model.add(Activation('relu'))
    model.add(MaxPooling2D((2, 2), padding='valid'))

    model.add(Flatten())
    model.add(Dropout(0.5))
    model.add(Dense(1024))
    model.add(Dense(256))
    model.add(Dense(64))
    model.add(Dense(1))

    model.compile(optimizer='adam', loss="mse")
    filepath = "Autopilot_10.h5"
    checkpoint = ModelCheckpoint(filepath, verbose=1, save_best_only=True)
    callbacks_list = [checkpoint]

    return model, callbacks_list


def loadFromPickle():
    with open("features", "rb") as f:
        features = np.array(pickle.load(f))
    with open("labels", "rb") as f:
        labels = np.array(pickle.load(f))

    return features, labels


def main():
    features, labels = loadFromPickle()
    features, labels = shuffle(features, labels)
    train_x, test_x, train_y, test_y = train_test_split(features, labels, rand
om_state=0,
                                                        test_size=0.3)
    train_x = train_x.reshape(train_x.shape[0], 100, 100, 1)
```

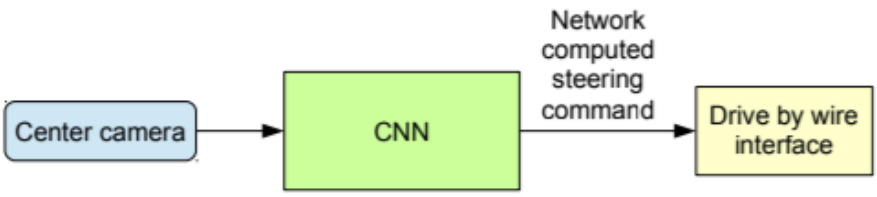
```
    test_x = test_x.reshape(test_x.shape[0], 100, 100, 1)
    model, callbacks_list = keras_model(100, 100)
    model.fit(train_x, train_y, validation_data=(test_x, test_y), epochs=3, ba
tch_size=32,
              callbacks=callbacks_list)
    print_summary(model)

    model.save('Autopilot_10.h5')


main()
K.clear_session();
```

Phase 3 : Autopilot Testing

Once trained, the network can generate steering from the video images of a single center camera. This configuration is shown in the figure.



We use this model to evaluate on the input video.

In [10]:

```
import numpy as np
from keras.layers import Dense, Activation, Flatten, Conv2D, Lambda
from keras.layers import MaxPooling2D, Dropout
from keras.utils import print_summary
from keras.models import Sequential
from keras.callbacks import ModelCheckpoint
import keras.backend as K
import pickle

from sklearn.model_selection import train_test_split
from sklearn.utils import shuffle


def keras_model(image_x, image_y):
    model = Sequential()
    model.add(Lambda(lambda x: x / 127.5 - 1., input_shape=(image_x, image_y,
1)))
    model.add(Conv2D(32, (3, 3), padding='same'))
    model.add(Activation('relu'))
    model.add(MaxPooling2D((2, 2), padding='valid'))
    model.add(Conv2D(32, (3, 3), padding='same'))
    model.add(Activation('relu'))
    model.add(MaxPooling2D((2, 2), padding='valid'))

    model.add(Conv2D(64, (3, 3), padding='same'))
    model.add(Activation('relu'))
    model.add(MaxPooling2D((2, 2), padding='valid'))
    model.add(Conv2D(64, (3, 3), padding='same'))
    model.add(Activation('relu'))
    model.add(MaxPooling2D((2, 2), padding='valid'))

    model.add(Conv2D(128, (3, 3), padding='same'))
    model.add(Activation('relu'))
    model.add(MaxPooling2D((2, 2), padding='valid'))
    model.add(Conv2D(128, (3, 3), padding='same'))
    model.add(Activation('relu'))
    model.add(MaxPooling2D((2, 2), padding='valid'))

    model.add(Flatten())
    model.add(Dropout(0.5))
    model.add(Dense(1024))
    model.add(Dense(256))
    model.add(Dense(64))
    model.add(Dense(1))

    model.compile(optimizer='adam', loss="mse")
    filepath = "Autopilot_10.h5"
    checkpoint = ModelCheckpoint(filepath, verbose=1, save_best_only=True)
    callbacks_list = [checkpoint]

    return model, callbacks_list


def loadFromPickle():
    with open("features", "rb") as f:
        features = np.array(pickle.load(f))
```

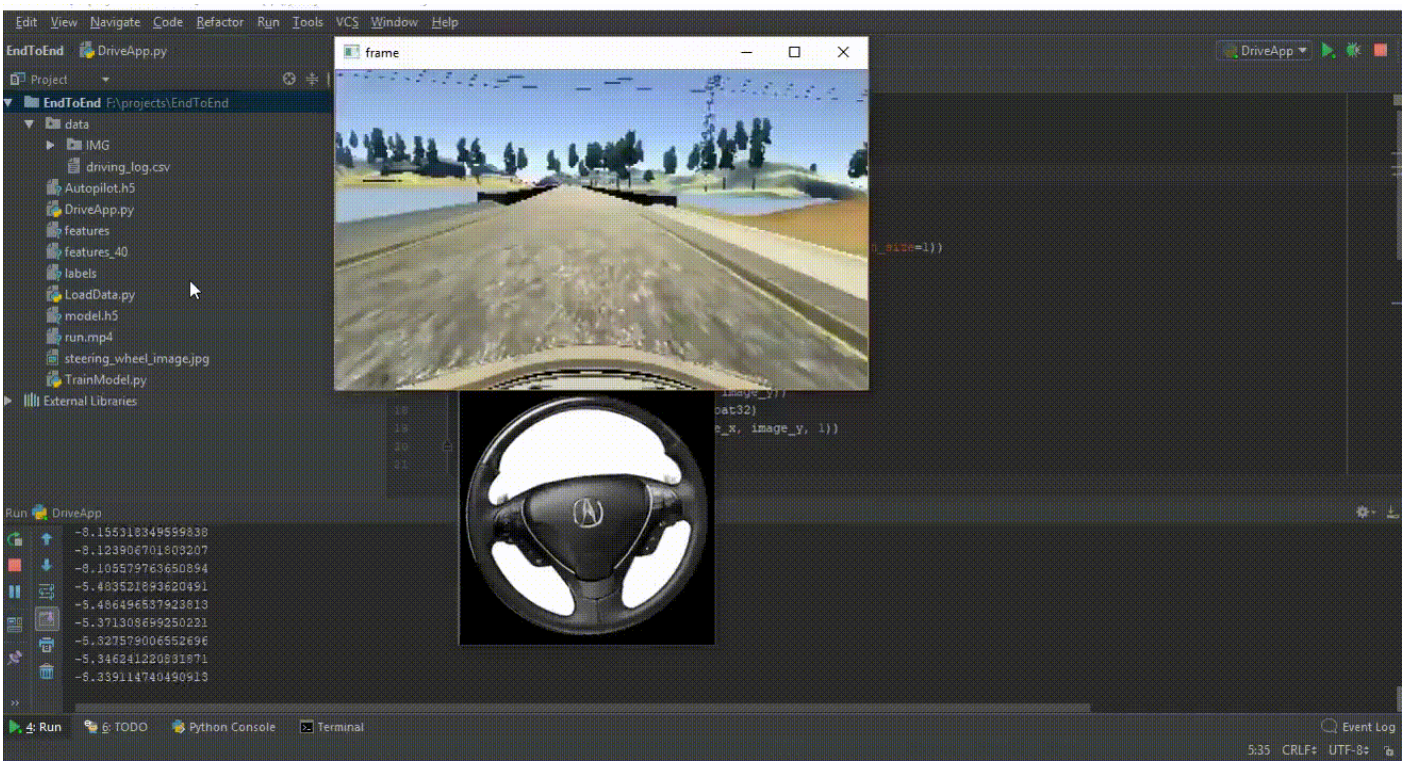
```
    with open("labels", "rb") as f:
        labels = np.array(pickle.load(f))

    return features, labels


def main():
    features, labels = loadFromPickle()
    features, labels = shuffle(features, labels)
    train_x, test_x, train_y, test_y = train_test_split(features, labels, rand
om_state=0,
                                                        test_size=0.3)
    train_x = train_x.reshape(train_x.shape[0], 100, 100, 1)
    test_x = test_x.reshape(test_x.shape[0], 100, 100, 1)
    model, callbacks_list = keras_model(100, 100)
    model.fit(train_x, train_y, validation_data=(test_x, test_y), epochs=3, ba
tch_size=32,
              callbacks=callbacks_list)
    print_summary(model)

    model.save('Autopilot_10.h5')


main()
K.clear_session();
```



Conclusion

We have shown that it is possible to create a model that reliably predicts steering wheel angles for a vehicle using a deep neural network and a plethora of data augmentation techniques. While we have obtained encouraging results, we would like in the future to explore the following:

1. Take into account speed and throttle in the model
2. Get the car to drive faster than 15–20MPH
3. Experiment with models based VGG/ResNets/Inception via transfer learning
4. Use Recurrent Neural Networks like in this paper from people using the Udacity dataset
5. Read the Learning A Driving Simulator paper by comma.ai and attempt to implement their model
6. Experiment with Reinforcement Learning

As can be seen, there are many areas we could explore to push this project further and obtain even more convincing results. One of the most important learnings from this project is that DATA IS POWERFUL: without all those images and steering angles, along with their potentially infinite augmentations, we would not have been able to build a robust enough model. From a personal perspective, We have tremendously enjoyed this project, the hardest so far, as it enabled me to gain more practical experience of hyper-parameter tweaking, data augmentation, and dataset balancing among other important concepts. We feel our intuition of neural network architectures has deepened as well.