Libraries

In [1]:

```
import numpy as np
import scipy.misc
import random
import scipy
import matplotlib.pyplot as plt
import os
from cv2 import cv2
from subprocess import call
import math
import keras
import json

import warnings
warnings.filterwarnings("ignore")

```

```
Using TensorFlow backend.
```

Data Preprocessing

In [2]:

```
xs = []
ys = []

#points to the end of the last batch
train_batch_pointer = 0
val_batch_pointer = 0

#read data.txt
with open("driving_dataset/data.txt") as f:
    for line in f:
        xs.append("driving_dataset/" + line.split()[0])
        #the paper by Nvidia uses the inverse of the turning radius,
        #but steering wheel angle is proportional to the inverse of turning radius
        #so the steering wheel angle in radians is used as the output
        ys.append(float(line.split()[1]) * scipy.pi / 180)

#get number of images
num_images = len(xs)

train_xs = xs[:int(len(xs) * 0.7)]
train_ys = ys[:int(len(xs) * 0.7)]

val_xs = xs[-int(len(xs) * 0.3):]
val_ys = ys[-int(len(xs) * 0.3):]

num_train_images = len(train_xs)
num_val_images = len(val_xs)

```

In [3]:

```
xs_t = []

#points to the end of the last batch
test_batch_pointer = 0

#read data.txt
with open("driving_dataset/test/data.txt") as f:
    for line in f:
        xs_t.append("driving_dataset/test" + line)
#get number of images
num_test_images = len(xs_t)
```

Generators

In [4]:

```
def LoadTrainBatch(batch_size):
    while True:
        global train_batch_pointer
        x_train = []
        y_train = []
        for i in range(0, batch_size):
            x_train.append(scipy.misc.imresize(scipy.misc.imread(train_xs[(train_batch_
            y_train.append([train_ys[(train_batch_pointer + i) % num_train_images]])
        train_batch_pointer += batch_size
        yield np.array(x_train), np.array(y_train)


def LoadValBatch(batch_size):
    while True:
        global val_batch_pointer
        x_cv = []
        y_cv = []
        for i in range(0, batch_size):
            x_cv.append(scipy.misc.imresize(scipy.misc.imread(val_xs[(val_batch_pointer
            y_cv.append([val_ys[(val_batch_pointer + i) % num_val_images]])
        val_batch_pointer += batch_size
        yield np.array(x_cv), np.array(y_cv)


def LoadTestBatch(batch_size):
    while True:
        global test_batch_pointer
        x_test = []
        for i in range(0, batch_size):
            x_test.append(scipy.misc.imresize(scipy.misc.imread(xs_t[(test_batch_pointe
        test_batch_pointer += batch_size
        yield np.array(x_test)
```

Model

In [5]:

```
batch_size = 100
epochs = 30
save_freq = 5
```

In [6]:

```
model = keras.models.Sequential()
model.add(layer=keras.layers.Conv2D(filters=24, kernel_size=(5,5), strides=(2, 2), pad
model.add(layer=keras.layers.BatchNormalization())
model.add(layer=keras.layers.Conv2D(filters=36, kernel_size=(5,5), strides=(2, 2), pad
model.add(layer=keras.layers.BatchNormalization())
model.add(layer=keras.layers.Conv2D(filters=48, kernel_size=(5,5), strides=(2, 2), pad
model.add(layer=keras.layers.BatchNormalization())
model.add(layer=keras.layers.Conv2D(filters=64, kernel_size=(3,3), strides=(1, 1), pad
model.add(layer=keras.layers.BatchNormalization())
model.add(layer=keras.layers.Conv2D(filters=64, kernel_size=(3,3), strides=(1, 1), pad
model.add(layer=keras.layers.BatchNormalization())
model.add(layer=keras.layers.Flatten())
model.add(layer=keras.layers.Dense(units=1164,activation='relu',kernel_initializer='gl
model.add(layer=keras.layers.BatchNormalization())
model.add(layer=keras.layers.Dropout(rate=0.5))
model.add(layer=keras.layers.Dense(units=100,activation='relu',kernel_initializer='glo
model.add(layer=keras.layers.BatchNormalization())
model.add(layer=keras.layers.Dropout(rate=0.5))
model.add(layer=keras.layers.Dense(units=50,activation='relu',kernel_initializer='glor
model.add(layer=keras.layers.BatchNormalization())
model.add(layer=keras.layers.Dropout(rate=0.5))
model.add(layer=keras.layers.Dense(units=10,activation='relu',kernel_initializer='glor
model.add(layer=keras.layers.BatchNormalization())
model.add(layer=keras.layers.Dropout(rate=0.5))
model.add(layer=keras.layers.Dense(units=1,activation='linear'))

```

```
WARNING:tensorflow:From c:\users\byron\applications\pythonmaster\lib\site-pa
ckages\tensorflow\python\framework\op_def_library.py:263: colocate_with (fro
m tensorflow.python.framework.ops) is deprecated and will be removed in a fu
ture version.
Instructions for updating:
Colocations handled automatically by placer.
WARNING:tensorflow:From c:\users\byron\applications\pythonmaster\lib\site-pa
ckages\keras\backend\tensorflow_backend.py:3445: calling dropout (from tenso
rflow.python.ops.nn_ops) with keep_prob is deprecated and will be removed in
a future version.
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate = 1 -
keep_prob`.
```

In [7]:

```
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 31, 98, 24)        1824
_________________________________________________________________
batch_normalization_1 (Batch (None, 31, 98, 24)        96
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 14, 47, 36)        21636
_________________________________________________________________
batch_normalization_2 (Batch (None, 14, 47, 36)        144
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 5, 22, 48)         43248
_________________________________________________________________
batch_normalization_3 (Batch (None, 5, 22, 48)         192
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 3, 20, 64)         27712
_________________________________________________________________
batch_normalization_4 (Batch (None, 3, 20, 64)         256
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 1, 18, 64)         36928
_________________________________________________________________
batch_normalization_5 (Batch (None, 1, 18, 64)         256
_________________________________________________________________
flatten_1 (Flatten)          (None, 1152)              0
_________________________________________________________________
dense_1 (Dense)              (None, 1164)              1342092
_________________________________________________________________
batch_normalization_6 (Batch (None, 1164)              4656
_________________________________________________________________
dropout_1 (Dropout)          (None, 1164)              0
_________________________________________________________________
dense_2 (Dense)              (None, 100)               116500
_________________________________________________________________
batch_normalization_7 (Batch (None, 100)               400
_________________________________________________________________
dropout_2 (Dropout)          (None, 100)               0
_________________________________________________________________
dense_3 (Dense)              (None, 50)                5050
_________________________________________________________________
batch_normalization_8 (Batch (None, 50)                200
_________________________________________________________________
dropout_3 (Dropout)          (None, 50)                0
_________________________________________________________________
dense_4 (Dense)              (None, 10)                510
_________________________________________________________________
batch_normalization_9 (Batch (None, 10)                40
_________________________________________________________________
dropout_4 (Dropout)          (None, 10)                0
_________________________________________________________________
dense_5 (Dense)              (None, 1)                 11
=================================================================
Total params: 1,601,751
Trainable params: 1,598,631
```

```
Non-trainable params: 3,120
_________________________________________________________________
```

Model - Training

In [8]:

```
# for epoch in range(epochs):
#     print('\nEpoch {}/{}'.format(epoch + 1, epochs))

#     losses, accs = [], []

#     for i, (X_train, Y_train) in enumerate(list(LoadTrainBatch(batch_size))):

#         loss, acc = model.train_on_batch(X_train, Y_train)
#         print('Batch {}: loss = {}, acc = {}'.format(i + 1, loss, acc))
#         losses.append(loss)
#         accs.append(acc)

#         if (epoch + 1) % save_freq == 0:
#             model.save_weights('weight_logs/'+'weights_{}.h5'.format(str(epoch + 1)),
#             print('Saved checkpoint to', 'weights_{}.h5'.format(epoch + 1))
```

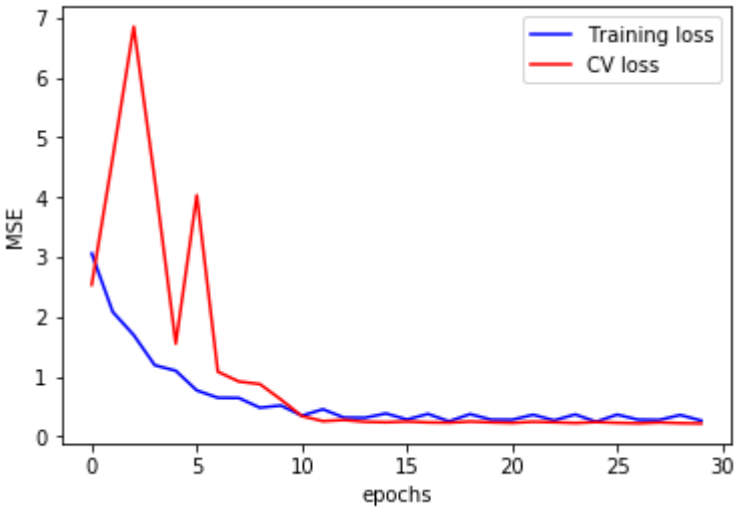
In [9]:

```
# Fitting a model generator:
if not os.path.exists('model_state/model.h5'):
    model.compile(optimizer=keras.optimizers.Adam(lr=0.0001, beta_1=0.9, beta_2=0.99),
    history = model.fit_generator(  generator=LoadTrainBatch(batch_size=batch_size),
                                    validation_data=LoadValBatch(batch_size=batch_size
                                    steps_per_epoch=int(num_images/batch_size),
                                    validation_steps=int(num_images/batch_size),
                                    epochs=epochs )
    model.save_weights('weight_logs/weights.h5')
    model.save('model_state/model.h5')
    with open('training_logs/json.json','w') as output:
        output.write(json.dumps(history.history))
    output = history.history
else:
    model = keras.models.load_model('model_state/model.h5')
    with open('training_logs/json.json','r') as output:
        output = json.loads(output.read())
```

```
WARNING:tensorflow:From c:\users\byron\applications\pythonmaster\lib\site-pa
ckages\tensorflow\python\ops\math_ops.py:3066: to_int32 (from tensorflow.pyt
hon.ops.math_ops) is deprecated and will be removed in a future version.
Instructions for updating:
Use tf.cast instead.
```

In [10]:

```
plt.plot(np.arange(epochs),output['loss'],color='b',label='Training loss')
plt.plot(np.arange(epochs),output['val_loss'],color='r',label='CV loss')
plt.xlabel('epochs')
plt.ylabel('MSE')
plt.legend()
plt.show()
```



Model - Validation

In [11]:

```
#model.load_weights(weight_logs/weights_30.h5)
# for epoch in range(epochs):
#     print('\nEpoch {}/{}'.format(epoch + 1, epochs))

#     losses, accs = [], []

#     for i, (X_cv, Y_cv) in enumerate(list(LoadValBatch(batch_size))):

#         loss, acc = model.test_on_batch(X_cv, Y_cv)
#         print('Batch {}: loss = {}, acc = {}'.format(i + 1, loss, acc))
#         losses.append(loss)
#         accs.append(acc)

#         if (epoch + 1) % save_freq == 0:
#             model.save_weights('weight_logs/'+'weights_{}.h5'.format(str(epoch + 1)),
#             print('Saved checkpoint to', 'weights_{}.h5'.format(epoch + 1))
```

In [12]:

```
1 model.evaluate_generator(generator=LoadValBatch(batch_size=batch_size), steps=int(num_
```

```
454/454 [==============================] - 167s 367ms/step
```

Out[12]:

```
0.23676606487264315
```

Model - Testing

In [13]:

```
i = math.ceil(num_images*0.7)
smoothed_angle = 0
img = cv2.imread('steering_wheel_image.jpg',0)
rows,cols = img.shape

while(cv2.waitKey(10) != ord('q')):

    full_image = scipy.misc.imread("driving_dataset/" + str(i) + ".jpg", mode="RGB")
    image = scipy.misc.imresize(full_image[-150:], [66, 200]) / 255.0


    degrees =  model.predict_on_batch(x=np.array([image])) * 180.0 / scipy.pi


    print("Steering angle: " + str(degrees[0][0]) + " (pred)\t" + str(ys[i]*180/scipy.
    cv2.imshow("frame", cv2.cvtColor(full_image, cv2.COLOR_RGB2BGR))

    smoothed_angle += 0.2 * pow(abs((degrees[0][0] - smoothed_angle)), 2.0 / 3.0) * (d

    M = cv2.getRotationMatrix2D((cols/2,rows/2),-smoothed_angle,1)
    dst = cv2.warpAffine(img,M,(cols,rows))
    cv2.imshow("steering wheel", dst)
    i += 1

cv2.destroyAllWindows()
```

```
Steering angle: -0.12940411 (pred)      -40.03000000000001 (actual)
Steering angle: 0.47187644 (pred)       -40.03000000000001 (actual)
Steering angle: 0.89203244 (pred)       -40.03000000000001 (actual)
Steering angle: 1.512366 (pred) -40.03000000000001 (actual)
Steering angle: 1.9969054 (pred)        -40.24 (actual)
Steering angle: 2.4896917 (pred)        -40.93999999999999 (actual)
Steering angle: 2.3975863 (pred)        -41.14 (actual)
Steering angle: 2.308415 (pred) -41.14 (actual)
Steering angle: 2.9398663 (pred)        -41.14 (actual)
Steering angle: 2.5729916 (pred)        -41.34 (actual)
Steering angle: 2.4226675 (pred)        -41.45 (actual)
Steering angle: 2.4810672 (pred)        -41.45 (actual)

Steering angle: 2.216194 (pred) -41.45 (actual)
Steering angle: 2.6775236 (pred)        -41.45 (actual)
Steering angle: 3.7102058 (pred)        -41.45 (actual)
Steering angle: 3.388995 (pred) -41.45 (actual)
Steering angle: 2.8680608 (pred)        -41.45 (actual)
Steering angle: 3.499905 (pred) -41.45 (actual)
Steering angle: 3.56835 (pred)  -41.45 (actual)
```

In [14]:

```
i = 45406
smoothed_angle = 0
img = cv2.imread('steering_wheel_image.jpg',0)
rows,cols = img.shape

while(cv2.waitKey(10) != ord('q')):

    full_image = scipy.misc.imread("driving_dataset/test/" + str(i) + ".jpg", mode="RGB
    image = scipy.misc.imresize(full_image[-150:], [66, 200]) / 255.0


    degrees =  model.predict_on_batch(x=np.array([image])) * 180.0 / scipy.pi


    print("Steering angle: " + str(degrees[0][0]) + " (pred)\t")
    cv2.imshow("frame", cv2.cvtColor(full_image, cv2.COLOR_RGB2BGR))

    smoothed_angle += 0.2 * pow(abs((degrees[0][0] - smoothed_angle)), 2.0 / 3.0) * (de

    M = cv2.getRotationMatrix2D((cols/2,rows/2),-smoothed_angle,1)
    dst = cv2.warpAffine(img,M,(cols,rows))
    cv2.imshow("steering wheel", dst)
    i += 1
    if i == 45567:
        break

cv2.destroyAllWindows()
```

```
Steering angle: 7.4521613 (pred)
Steering angle: 7.2200375 (pred)
Steering angle: 6.88315 (pred)
Steering angle: 6.8863583 (pred)
Steering angle: 7.2005343 (pred)
Steering angle: 7.059284 (pred)
Steering angle: 7.1794486 (pred)
Steering angle: 6.964485 (pred)
Steering angle: 7.056954 (pred)
Steering angle: 6.948978 (pred)
Steering angle: 7.0628633 (pred)
Steering angle: 6.677731 (pred)
Steering angle: 6.6757264 (pred)
Steering angle: 6.2726126 (pred)
Steering angle: 5.876835 (pred)
Steering angle: 5.89881 (pred)
Steering angle: 5.456636 (pred)
Steering angle: 5.5041337 (pred)
Steering angle: 5.8003535 (pred)
```

In [ ]:

```
1
```