CNN_Accel_Brake

March 13, 2019

1 Acceleration Prediction using LeNet-inspired Architecture

1.1 Purpose

To predict the acceleration of the vehicle based on the image frames in the video.

We use the **3-layered Convnet Architecture design inspired by LeNet, 1998 paper by Le Cunn.**

1.2 Custom-Defined Functions

```
In [2]: import scipy.misc
        import random
        import numpy as np

        xs = []
        ys = []
        accels = []
        brake = []
        # gear = []
        opticalFlow = []

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

        dataPath = "indian_dataset/"
        corrDataPath = "indian_dataset/corr/"
        fileNamePrefix = "circuit2_x264.mp4 "


        with open(dataPath+"data.txt") as f:
            for line in f:
                xs.append(scipy.misc.imresize(scipy.misc.imread(
                    dataPath + fileNamePrefix + str(int(line.split()[0])).zfill(5)+".jpg")[-150
                #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
                accels.append(float(line.split()[2])* scipy.pi / 180)


        #get number of images
        num_images = len(xs)

        # train_xs, train_accels, val_xs, val_accels
        train_xs = xs[:int(len(xs) * 0.8)]
        train_accels = accels[:int(len(xs) * 0.8)]

        val_xs = xs[-int(len(xs) * 0.2):]
        val_accels = accels[-int(len(xs) * 0.2):]

        # print(train_accels)
        # print(len(train_xs))

        # train_xs, train_accels, val_xs, val_accels
        train_xs = np.array(train_xs)
        train_accels = np.array(train_accels)

        val_xs = np.array(val_xs)
        val_accels = np.array(val_accels)

        input_shape = (112, 112, 3)
```

```
c:\python\lib\site-packages\ipykernel_launcher.py:26: DeprecationWarning: `imread` is deprecate
`imread` is deprecated in SciPy 1.0.0, and will be removed in 1.2.0.
Use ``imageio.imread`` instead.
c:\python\lib\site-packages\ipykernel_launcher.py:26: DeprecationWarning: `imresize` is depreca
`imresize` is deprecated in SciPy 1.0.0, and will be removed in 1.2.0.
Use ``skimage.transform.resize`` instead.
```

1.3 Model 1: LeNet Inspired 3-Convolution Layer Architecture

This 3-layered is different but inspired from the LeNet, 1998 paper by Le Cunn.
http://yann.lecun.com/exdb/publis/pdf/lecun-01a.pdf

```
In [2]: # The model is inspired from the LeNet, 1998 paper by Le Cunn

        # Credits: https://github.com/keras-team/keras/blob/master/examples/mnist_cnn.py

        from __future__ import print_function
        import keras
        from keras.datasets import mnist
        from keras.models import Sequential
        from keras.layers import Dense, Dropout, Flatten
```

```python
from keras.layers import Conv2D, MaxPooling2D
from keras import backend as K

from keras.layers.normalization import BatchNormalization
# import seaborn as sns

batch_size = 128
epochs = 30

model = Sequential()
model.add(Conv2D(256, kernel_size=(3, 3),
                 activation='relu',
                 input_shape=input_shape)) #Convolution
model.add(MaxPooling2D(pool_size=(2, 2)))  #Subsampling
model.add(Dropout(0.25))
# model.add(BatchNormalization())

model.add(Conv2D(128, (3, 3), activation='relu')) #Convolution
model.add(MaxPooling2D(pool_size=(2, 2)))        #Subsampling
model.add(Dropout(0.25))
# model.add(BatchNormalization())

# model.add(Conv2D(256, (3, 3), activation='relu'))
# model.add(Dropout(0.5))
model.add(Flatten())
model.add(Dense(128, activation='relu')) # Full Connection
model.add(Dropout(0.5))
model.add(Dense(64, activation='relu')) # Full Connection
model.add(Dropout(0.5))
model.add(Dense(1))

model.compile(loss='mse',#loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.Adadelta(),
              metrics=['mean_squared_error', 'mean_absolute_error', 'mean_absolute_perc


# train_xs, train_accels, val_xs, val_accels
history=model.fit(train_xs, train_accels,
          batch_size=batch_size,
          epochs=epochs,
          verbose=1,
          validation_data=(val_xs, val_accels))

score = model.evaluate(val_xs, val_accels, verbose=0)
# print('Test loss:', score[0])
# print('Test accuracy:', score[1])

# plotGraph(history=history)
```

```
        # plotWeightM1(model=model)
```

```
Using TensorFlow backend.


Train on 23760 samples, validate on 5940 samples
Epoch 1/30
23760/23760 [==============================] - 72s 3ms/step - loss: 0.0027 - mean_squared_erro
Epoch 2/30
23760/23760 [==============================] - 67s 3ms/step - loss: 9.8084e-04 - mean_squared_e
Epoch 3/30
23760/23760 [==============================] - 66s 3ms/step - loss: 8.6685e-04 - mean_squared_e
Epoch 4/30
23760/23760 [==============================] - 66s 3ms/step - loss: 7.9394e-04 - mean_squared_e
Epoch 5/30
23760/23760 [==============================] - 66s 3ms/step - loss: 7.2962e-04 - mean_squared_e
Epoch 6/30
23760/23760 [==============================] - 66s 3ms/step - loss: 6.8767e-04 - mean_squared_e
Epoch 7/30
23760/23760 [==============================] - 66s 3ms/step - loss: 6.4344e-04 - mean_squared_e
Epoch 8/30
23760/23760 [==============================] - 66s 3ms/step - loss: 5.9781e-04 - mean_squared_e
Epoch 9/30
23760/23760 [==============================] - 66s 3ms/step - loss: 5.7865e-04 - mean_squared_e
Epoch 10/30
23760/23760 [==============================] - 66s 3ms/step - loss: 5.4817e-04 - mean_squared_e
Epoch 11/30
23760/23760 [==============================] - 66s 3ms/step - loss: 5.1795e-04 - mean_squared_e
Epoch 12/30
23760/23760 [==============================] - 66s 3ms/step - loss: 4.8637e-04 - mean_squared_e
Epoch 13/30
23760/23760 [==============================] - 66s 3ms/step - loss: 4.7477e-04 - mean_squared_e
Epoch 14/30
23760/23760 [==============================] - 66s 3ms/step - loss: 4.4863e-04 - mean_squared_e
Epoch 15/30
23760/23760 [==============================] - 66s 3ms/step - loss: 4.2700e-04 - mean_squared_e
Epoch 16/30
23760/23760 [==============================] - 66s 3ms/step - loss: 4.1772e-04 - mean_squared_e
Epoch 17/30
23760/23760 [==============================] - 66s 3ms/step - loss: 4.0288e-04 - mean_squared_e
Epoch 18/30
23760/23760 [==============================] - 66s 3ms/step - loss: 3.8548e-04 - mean_squared_e
Epoch 19/30
23760/23760 [==============================] - 67s 3ms/step - loss: 3.7357e-04 - mean_squared_e
Epoch 20/30
23760/23760 [==============================] - 66s 3ms/step - loss: 3.6180e-04 - mean_squared_e
Epoch 21/30
23760/23760 [==============================] - 66s 3ms/step - loss: 3.5360e-04 - mean_squared_e
```

```
Epoch 22/30
23760/23760 [==============================] - 66s 3ms/step - loss: 3.4575e-04 - mean_squared_e
Epoch 23/30
23760/23760 [==============================] - 66s 3ms/step - loss: 3.3386e-04 - mean_squared_e
Epoch 24/30
23760/23760 [==============================] - 66s 3ms/step - loss: 3.2564e-04 - mean_squared_e
Epoch 25/30
23760/23760 [==============================] - 66s 3ms/step - loss: 3.1210e-04 - mean_squared_e
Epoch 26/30
23760/23760 [==============================] - 66s 3ms/step - loss: 3.0707e-04 - mean_squared_e
Epoch 27/30
23760/23760 [==============================] - 66s 3ms/step - loss: 3.0236e-04 - mean_squared_e
Epoch 28/30
23760/23760 [==============================] - 66s 3ms/step - loss: 2.9985e-04 - mean_squared_e
Epoch 29/30
23760/23760 [==============================] - 66s 3ms/step - loss: 2.8882e-04 - mean_squared_e
Epoch 30/30
23760/23760 [==============================] - 66s 3ms/step - loss: 2.9487e-04 - mean_squared_e
```

```
In [16]: # To try predict acceleration using train data
         pred = model.predict(train_xs)

         #Print acceleration first n frames
         for i in pred[0:500]:
             print(i*180/scipy.pi)
```

```
[0.92357355]
[0.92357355]
[0.9310391]
[0.9301475]
[0.9338954]
[0.9341978]
[0.94233406]
[0.93697715]
[0.9277656]
[0.931919]
[0.9339071]
[0.9301183]
[0.9293895]
[0.9270301]
[0.9313618]
[0.94097805]
[0.9448145]
[0.9453828]
[0.9402479]
[0.94632584]
[0.94994295]
```

```
[0.9340492]
[0.9422594]
[0.93306893]
[0.94362503]
[0.9397941]
[0.9537025]
[0.94687253]
[0.9523851]
[0.9354172]
[0.9383295]
[0.9428991]
[0.94260913]
[0.94029]
[0.9449467]
[0.9489774]
[0.942226]
[0.93854314]
[0.9425991]
[0.95988804]
[0.94689965]
[0.948667]
[0.9394076]
[0.9394108]
[0.94459707]
[0.93810683]
[0.9388357]
[0.9477207]
[0.9370355]
[0.9421201]
[0.9484173]
[0.94396645]
[0.948088]
[0.9414139]
[0.9583285]
[0.9663747]
[0.95048684]
[0.95292795]
[0.9477265]
[0.9556924]
[0.9527644]
[0.9528844]
[0.95880514]
[0.9665853]
[0.9427091]
[0.94912565]
[0.9515617]
[0.96681327]
[0.96333563]
```

```
[0.9631466]
[0.95660615]
[0.95560145]
[0.96558595]
[0.97763824]
[0.97463447]
[0.98999953]
[0.98700166]
[0.98382944]
[0.9916441]
[0.9790558]
[0.9685356]
[0.9627041]
[0.96793133]
[0.943886]
[0.95342344]
[0.9442144]
[0.95509684]
[0.95879275]
[0.9559156]
[0.9542325]
[0.9513349]
[0.9522948]
[0.95300186]
[0.94813055]
[0.9415414]
[0.93940866]
[0.93264735]
[0.93566215]
[0.92251396]
[0.9200164]
[0.915039]
[0.9238741]
[0.90476394]
[0.91993165]
[0.9099057]
[0.91145986]
[0.92022145]
[0.9114425]
[0.91618896]
[0.9241743]
[0.9106088]
[0.92057216]
[0.9206505]
[0.9085712]
[0.9204438]
[0.925014]
[0.9395769]
```

```
[0.9590481]
[0.96299845]
[0.9686124]
[0.9589119]
[0.9487808]
[0.94530976]
[0.93471396]
[0.9303346]
[0.9274143]
[0.9238589]
[0.91802675]
[0.9126113]
[0.9129979]
[0.91688985]
[0.9166581]
[0.9108903]
[0.90156496]
[0.8984708]
[0.8999014]
[0.90529615]
[0.91165096]
[0.9138574]
[0.9254932]
[0.92828864]
[0.9282427]
[0.9110569]
[0.9005762]
[0.88092494]
[0.8791093]
[0.87473285]
[0.86107963]
[0.86474204]
[0.8650788]
[0.8623865]
[0.85104436]
[0.86458415]
[0.8672497]
[0.8708364]
[0.8623261]
[0.86947644]
[0.87745875]
[0.90166086]
[0.91387105]
[0.8822031]
[0.89150643]
[0.88545954]
[0.91030884]
[0.89821005]
```

```
[0.8872493]
[0.88205874]
[0.86367106]
[0.8707646]
[0.850026]
[0.84275156]
[0.8319595]
[0.8219217]
[0.8204057]
[0.8229427]
[0.79350805]
[0.7840077]
[0.7704222]
[0.7479197]
[0.72855395]
[0.7314589]
[0.7406634]
[0.73661315]
[0.7195626]
[0.7203924]
[0.71316713]
[0.72594583]
[0.73706394]
[0.74367577]
[0.74674004]
[0.7457792]
[0.75394696]
[0.7480418]
[0.74752545]
[0.7257994]
[0.73646235]
[0.75342077]
[0.77841955]
[0.8143443]
[0.83158535]
[0.8549231]
[0.868442]
[0.85895497]
[0.8365425]
[0.8095643]
[0.81338644]
[0.8365186]
[0.8482044]
[0.87796515]
[0.88283074]
[0.8600529]
[0.8161016]
[0.8269331]
```

```
[0.822992]
[0.8294791]
[0.82440877]
[0.8439457]
[0.8682041]
[0.8441302]
[0.859312]
[0.84792715]
[0.8761638]
[0.86791134]
[0.84269667]
[0.8435793]
[0.8634543]
[0.89717454]
[0.9356946]
[0.99599713]
[1.0167785]
[1.0339339]
[1.0642612]
[1.0840813]
[1.1420026]
[1.1634926]
[1.2290145]
[1.1966968]
[1.1835644]
[1.0982788]
[1.0384785]
[1.01977]
[0.959578]
[0.97629225]
[1.0454319]
[1.0531812]
[1.0263866]
[0.99493665]
[1.0072173]
[1.0026139]
[1.0118704]
[0.9863667]
[0.9837937]
[0.97154677]
[0.9626454]
[0.9596074]
[0.95364636]
[0.9561902]
[0.93159264]
[0.94842505]
[0.93144876]
[0.93686867]
```

```
[0.9241271]
[0.9118338]
[0.90270805]
[0.9056711]
[0.9051896]
[0.89212143]
[0.87769675]
[0.8811056]
[0.8629304]
[0.8605791]
[0.85038817]
[0.8503183]
[0.83899415]
[0.84407043]
[0.84660786]
[0.83424664]
[0.8290462]
[0.8138602]
[0.8335827]
[0.82630485]
[0.82564825]
[0.8310502]
[0.84167564]
[0.8475244]
[0.86636657]
[0.85709214]
[0.8758698]
[0.8664634]
[0.8893201]
[0.88245296]
[0.9067545]
[0.916322]
[0.9202887]
[0.90955347]
[0.92365944]
[0.9192844]
[0.91451967]
[0.9344861]
[0.9293363]
[0.95008814]
[0.93010783]
[0.9547901]
[0.94410104]
[0.9443989]
[0.95232236]
[0.9644196]
[0.9682893]
[0.9781855]
```

```
[0.96252066]
[0.9830696]
[0.98204786]
[1.0007803]
[1.0048856]
[1.0234174]
[1.048122]
[1.0252248]
[1.0299866]
[1.0306106]
[1.0453371]
[1.0488528]
[1.0647415]
[1.0784332]
[1.1112446]
[1.1344757]
[1.1273502]
[1.1241584]
[1.1441549]
[1.1431059]
[1.1465023]
[1.1531982]
[1.1678956]
[1.1774986]
[1.1893216]
[1.2158031]
[1.2449013]
[1.2680755]
[1.2864035]
[1.3209008]
[1.3313009]
[1.3562062]
[1.3692762]
[1.3714597]
[1.370057]
[1.3641974]
[1.3645078]
[1.375003]
[1.3705657]
[1.3830924]
[1.3837837]
[1.3706061]
[1.3627698]
[1.34656]
[1.3507036]
[1.3533473]
[1.3483729]
[1.3463407]
```

```
[1.3468617]
[1.3425999]
[1.3328763]
[1.3277127]
[1.3588235]
[1.3306398]
[1.33572]
[1.359622]
[1.3321513]
[1.3348776]
[1.31162]
[1.286075]
[1.3127716]
[1.3437306]
[1.3520048]
[1.4002119]
[1.4235718]
[1.4195621]
[1.3825723]
[1.381219]
[1.3690385]
[1.3706627]
[1.3709453]
[1.3364332]
[1.338435]
[1.3284469]
[1.3391124]
[1.3389394]
[1.296155]
[1.227619]
[1.268974]
[1.2880924]
[1.2478769]
[1.2383322]
[1.2607045]
[1.2887259]
[1.3293341]
[1.3731195]
[1.3611574]
[1.3551658]
[1.3309369]
[1.3600179]
[1.3449199]
[1.3116074]
[1.3144517]
[1.3136162]
[1.3415769]
[1.3164779]
```

```
[1.2834789]
[1.2527516]
[1.2733141]
[1.2763101]
[1.2891037]
[1.2677023]
[1.3220471]
[1.3370636]
[1.319204]
[1.2914317]
[1.2886823]
[1.3080208]
[1.3143456]
[1.3255308]
[1.3459607]
[1.2954897]
[1.3518258]
[1.3438071]
[1.3301889]
[1.3843293]
[1.3553054]
[1.2707816]
[1.2966286]
[1.3336034]
[1.4060547]
[1.4745098]
[1.5178976]
[1.5731158]
[1.576893]
[1.5992246]
[1.6033586]
[1.6190901]
[1.6013478]
[1.591466]
[1.5948329]
[1.6320802]
[1.6587751]
[1.6477685]
[1.7029836]
[1.7431957]
[1.8917855]
[1.9616673]
[2.0167127]
[2.02814]
[2.0636706]
[2.110657]
[2.2280586]
[2.3639114]
```

```
[2.4639022]
[2.685708]
[2.76365]
[2.8592691]
[2.9558606]
[2.983342]
[3.2166684]
[3.3073065]
[3.270742]
[3.7331123]
[3.9293516]
[4.0053616]
[4.06205]
[4.2649136]
[4.337888]
[4.3596253]
[4.4200063]
[4.5125837]
[4.5460014]
[4.5233364]
[4.4941955]
[4.4982486]
[4.5160336]
[4.5245953]
[4.417109]
[4.42547]
[4.4219527]
[4.4052186]
[4.3705196]
[4.3673573]
[4.3751907]
[4.441688]
[4.5124846]
[4.539051]
[4.6165376]
[4.620568]
[4.664722]
[4.6845293]
[4.6924257]
[4.654179]
[4.616378]
[4.646823]
[4.682429]
[4.6911473]
[4.6978917]
[4.701402]
[4.70392]
```

```
In [4]: # model.save("save/model_accel.ckpt")
```