User manual of the CNN

Problem Statement

Implement a CNN to work on the outputs (the graphical images) of the test results generated from Thales Alenia Space España test benches during equipment acceptance campaigns by outputting 3 different values namely current_stabilised_value (mA), current_max/min_value (mA), current_rise/fall_time_spec (mS).

Methodology

- Two neural networks are built one to work with the ON power state and other on the OFF power state.
- The problem is dealt as a regression problem.
- The neural network takes the bmp images (outputs of the test benches) as the raw input and outputs the 4 values mentioned above.
- These values are then compared with their Spec values to obtain Compliant/non - Compliant.
- We have around 400 samples in ON and OFF dataset each.
- When compared to the previous implementation using perceptron the present implementation does not require any reduction of the feature space as inhere we take images as inputs instead of values. This is an added advantage as the loss of data due to feature space reduction was one of the main reasons of low accuracy.

Data Pre-processing and Code Analysis

❖ Inorder to normalise the data the following technique was used:
- We know that max values of current we measure in ON = 800 and min is -100 [from the graphs] , max and min of time we measure in ON are 100 and 0 [from the graphs] .
- We know that max values of current we measure in OFF = 200 and min is -200 [from the graphs] , max and min of time we measurein OFF are 10 and 0 [from the graphs] .
- So we performing min max scaling using the above data.
- We also resize the image to 150 pixels.

- We first read the data, normalize and split the data into test and train.

```
image_size = 150
# OFF - 'current_stabilised_value (mA)'
min1 = -200
max1 = 200

def load_train():
   X_train = []
   y_train = []
   heights = pd.read_csv('drive/My Drive/INTERN/socis/CNN/train_off.csv')
   print('Read train images')
   for index, row in heights.iterrows():
       try:

           image_path = os.path.join('drive/My Drive/INTERN/socis/CNN/Final_Dataset',
str((row['Path'])) )
           print(image_path,"imagepath")
           img = cv2.imread(image_path, cv2.IMREAD_COLOR)
           img =  img[83:441, 30:510]
           img = cv2.resize(img, (image_size, image_size) ).astype('uint8')

           X_train.append(img)
           y_train.append( [ float(row['current_stabilised_value (mA)']) ])


def read_and_normalize_train_data():
   train_data, train_target = load_train()
   train_data = np.array(train_data, dtype=np.float64)
   train_target = np.array(train_target, dtype=np.float64)
   m = train_data.mean()
   s = train_data.std()

   print ('Train mean, sd:', m, s )
   train_data -= m
   train_data /= s
   print('Train shape:', train_data.shape)
   print(train_data.shape[0], 'train samples')
   return train_data, train_target
# DATA PRE PROCESSING
train_data, train_target = read_and_normalize_train_data()
   train_data = train_data[0:num_samples,:,:,:]
   train_target = train_target[0:num_samples]
   train_target =  (train_target - min1)/(max1 - min1)

 X_train, X_valid, y_train, y_valid = train_test_split(train_data, train_target,
test_size=cv_size, random_state=56741)
```

- The base model of the CNN (inspired form RESNET - 50 and VGG) :

```
def create_model():
   nb_filters = 8
   nb_conv = 5

   ##model building
   model = Sequential()

--------------------------------------------------------------------------------
------------------------------------------------------------


   model.add(Convolution2D(nb_filters, nb_conv, nb_conv,
                           border_mode='valid',
                           input_shape=(image_size, image_size, 3) ) )
   model.add(Activation('relu'))

   model.add(Convolution2D(nb_filters, nb_conv, nb_conv))
   model.add(Activation('relu'))

   model.add(Convolution2D(nb_filters, nb_conv, nb_conv))
   model.add(Activation('relu'))

   model.add(Convolution2D(nb_filters, nb_conv, nb_conv))
   model.add(Activation('relu'))
   model.add(Dropout(0.25))

   model.add(Convolution2D(nb_filters*2, nb_conv, nb_conv))
   model.add(Activation('relu'))

   model.add(Convolution2D(nb_filters*2, nb_conv, nb_conv))
   model.add(Activation('relu'))

   model.add(Convolution2D(nb_filters*2, nb_conv, nb_conv))
   model.add(Activation('relu'))

   model.add(Convolution2D(nb_filters*2, nb_conv, nb_conv))
   model.add(Activation('relu'))
   model.add(Dropout(0.5))

   model.add(Flatten())
   model.add(Dense(256))
   model.add(Activation('relu'))
```

```
    model.add(Dropout(0.5))

    model.add(Dense(128))
    model.add(Activation('relu'))
    model.add(Dropout(0.5))

    model.add(Dense(1))
    model.add(Activation('linear'))

    model.compile(loss='mean_squared_error', optimizer=Adadelta())
    return model
```

- We then train the model for 50 epochs each of batch size 8 and use r2 score for evaluation of the model.

```
def train_model(batch_size = 8, nb_epoch = 2):
    from sklearn.preprocessing import RobustScaler
    num_samples = 370
    cv_size = 0.2

    train_data, train_target = read_and_normalize_train_data()
    train_data = train_data[0:num_samples,:,:,:]
    train_target = train_target[0:num_samples]
    train_target =  (train_target + 200 - min1)/(max1 - min1)
    X_train, X_valid, y_train, y_valid = train_test_split(train_data, train_target,
test_size=cv_size, random_state=56741)


    model = create_model()
    history = model.fit(X_train, y_train, batch_size=batch_size, nb_epoch=nb_epoch,
verbose=1, validation_data=(X_valid, y_valid) )

    predictions_valid = (model.predict(X_valid, batch_size=8, verbose=1))


    y1 =  (( y_valid.flatten() ) * ( max1 - min1)) + min1
    y2 = ((predictions_valid.flatten() ) * ( max1 - min1)) + min1
    print(y1,"y1")
    print(y2,"y2")
    from sklearn.metrics import r2_score
    r2 = r2_score(y_valid, predictions_valid)
    print(r2,"r2")
```

```
   return model, history

# CALL THE FUNCTION
model1, history = train_model(nb_epoch = 50)
```

Results

- As we are solving the problem using Regression analysis we use R2 score to measure the efficiency of the model.

R2_SCORES	ON	OFF
current_rise/fall_time_spec (mS).	0.4-0.5	0.3
current_stabilised_value (mA)	0.3 - 0.4	0.8-0.9
current_max/min_value (mA)	0.8 -0.9	0.6-0.7

Deploy the model

- Make sure to follow the instruction on the cleaning images section of repository for data pre processing.
- Note the path for the csv files and change the file path in the reg_cnn (code).
- You can either run the code on the google colab or any jupyter notebook with all the dependencies involved.
- Inorder to run the code for ON (power parameter):
    - current [current_stabilised_value (mA),current_max/min_value (mA)] ----- min = -100 and max = 800
    - current_rise/fall_time_value (mS) : min = 0 max = 100

- Inorder to run the code for OFF(power parameter):

    - current [current_stabilised_value (mA),current_max/min_value (mA)] ----- min = -200 and max = 200
    - current_rise/fall_time_value (mS) : min = 0 max = 10
- Alter the values accorningly while running the code
- Inorder to run it on ON dataset - use train.csv
- Inorder to run it on OFF dataset - use train_off.csv
- Change the value of row array accordingly in the line depending on the parameter to be evaluated.

```
y_train.append( [ float(row['current_stabilised_value (mA)']) ])
```