In [1]:

```
#running CNN.
#need two input
#data correct-data incorrect___label correct-label incorrect
#added another deep layer to existing CNN
#computed prediction for 90 exercises and finding mean for them
#calculated both angle and position data for predicting movement quality
```

In [ ]:

In [2]:

```
# Import libraries and functions

from __future__ import print_function
import numpy as np
np.random.seed(1337)  # for reproducibility

import csv
import os,random

from keras.models import Sequential
from keras.layers import Dense
from keras.layers.convolutional import Convolution1D
from keras.layers.core import Dropout, Activation, Flatten
from keras.layers.advanced_activations import LeakyReLU
from keras.optimizers import *
from keras.callbacks import EarlyStopping

import matplotlib.pyplot as plt
%matplotlib inline

from sklearn.metrics import mean_squared_error
from math import sqrt

import datetime
now = datetime.datetime.now
```

```
Using TensorFlow backend.
C:\Users\mr1486\.conda\envs\marufi\lib\site-packages\tensorflow\python\framew
ork\dtypes.py:516: FutureWarning: Passing (type, 1) or '1type' as a synonym o
f type is deprecated; in a future version of numpy, it will be understood as
(type, (1,)) / '(1,)type'.
  _np_qint8 = np.dtype([("qint8", np.int8, 1)])
C:\Users\mr1486\.conda\envs\marufi\lib\site-packages\tensorflow\python\framew
ork\dtypes.py:517: FutureWarning: Passing (type, 1) or '1type' as a synonym o
f type is deprecated; in a future version of numpy, it will be understood as
(type, (1,)) / '(1,)type'.
  _np_quint8 = np.dtype([("quint8", np.uint8, 1)])
C:\Users\mr1486\.conda\envs\marufi\lib\site-packages\tensorflow\python\framew
ork\dtypes.py:518: FutureWarning: Passing (type, 1) or '1type' as a synonym o
f type is deprecated; in a future version of numpy, it will be understood as
(type, (1,)) / '(1,)type'.
  _np_qint16 = np.dtype([("qint16", np.int16, 1)])
C:\Users\mr1486\.conda\envs\marufi\lib\site-packages\tensorflow\python\framew
ork\dtypes.py:519: FutureWarning: Passing (type, 1) or '1type' as a synonym o
f type is deprecated; in a future version of numpy, it will be understood as
(type, (1,)) / '(1,)type'.
  _np_quint16 = np.dtype([("quint16", np.uint16, 1)])
C:\Users\mr1486\.conda\envs\marufi\lib\site-packages\tensorflow\python\framew
ork\dtypes.py:520: FutureWarning: Passing (type, 1) or '1type' as a synonym o
f type is deprecated; in a future version of numpy, it will be understood as
(type, (1,)) / '(1,)type'.
  _np_qint32 = np.dtype([("qint32", np.int32, 1)])
C:\Users\mr1486\.conda\envs\marufi\lib\site-packages\tensorflow\python\framew
ork\dtypes.py:525: FutureWarning: Passing (type, 1) or '1type' as a synonym o
f type is deprecated; in a future version of numpy, it will be understood as
(type, (1,)) / '(1,)type'.
  np_resource = np.dtype([("resource", np.ubyte, 1)])
C:\Users\mr1486\.conda\envs\marufi\lib\site-packages\tensorboard\compat\tenso
rflow_stub\dtypes.py:541: FutureWarning: Passing (type, 1) or '1type' as a sy
nonym of type is deprecated; in a future version of numpy, it will be underst
ood as (type, (1,)) / '(1,)type'.
  _np_qint8 = np.dtype([("qint8", np.int8, 1)])
C:\Users\mr1486\.conda\envs\marufi\lib\site-packages\tensorboard\compat\tenso
rflow_stub\dtypes.py:542: FutureWarning: Passing (type, 1) or '1type' as a sy
nonym of type is deprecated; in a future version of numpy, it will be underst
ood as (type, (1,)) / '(1,)type'.
  _np_quint8 = np.dtype([("quint8", np.uint8, 1)])
C:\Users\mr1486\.conda\envs\marufi\lib\site-packages\tensorboard\compat\tenso
rflow_stub\dtypes.py:543: FutureWarning: Passing (type, 1) or '1type' as a sy
nonym of type is deprecated; in a future version of numpy, it will be underst
ood as (type, (1,)) / '(1,)type'.
  _np_qint16 = np.dtype([("qint16", np.int16, 1)])
C:\Users\mr1486\.conda\envs\marufi\lib\site-packages\tensorboard\compat\tenso
rflow_stub\dtypes.py:544: FutureWarning: Passing (type, 1) or '1type' as a sy
nonym of type is deprecated; in a future version of numpy, it will be underst
ood as (type, (1,)) / '(1,)type'.
  _np_quint16 = np.dtype([("quint16", np.uint16, 1)])
C:\Users\mr1486\.conda\envs\marufi\lib\site-packages\tensorboard\compat\tenso
rflow_stub\dtypes.py:545: FutureWarning: Passing (type, 1) or '1type' as a sy
nonym of type is deprecated; in a future version of numpy, it will be underst
ood as (type, (1,)) / '(1,)type'.
  _np_qint32 = np.dtype([("qint32", np.int32, 1)])
C:\Users\mr1486\.conda\envs\marufi\lib\site-packages\tensorboard\compat\tenso
```

```
rflow_stub\dtypes.py:550: FutureWarning: Passing (type, 1) or '1type' as a sy
nonym of type is deprecated; in a future version of numpy, it will be underst
ood as (type, (1,)) / '(1,)type'.
  np_resource = np.dtype([("resource", np.ubyte, 1)])
```

In [3]:

```
timesteps = 240 # Number of timesteps
nr = 90   # Number of repetitions
n_dim = 117  # Dimension of data sequences
dropout_rate = 0.2   # Droput rate
```

In [4]:

```
# Function to load the data
#going through angle data of vicon sensor
def load_data(nr, n_dim, timesteps):
    f = open('C:/Users/mr1486/Downloads/CNN_Rehab_vicon_angle/data_correct/Dat
a_Correct_m02.csv')
    csv_f = csv.reader(f)
    Correct_X = list(csv_f)

    # Convert the input sequences into numpy arrays
    train_input1 = np.asarray(Correct_X, dtype = float)
    correct_input = np.zeros((nr,timesteps,n_dim))
    for i in range(len(train_input1)//n_dim):
           correct_input[i,:,:] = np.transpose(train_input1[n_dim*i:n_dim*(i+1
),:])

    f = open('C:/Users/mr1486/Downloads/CNN_Rehab_vicon_angle/label_correct/La
bels_Correct_m02.csv')
    csv_f = csv.reader(f)
    Correct_Y = list(csv_f)

    # Convert the input labels into numpy arrays
    correct_label = np.asarray(Correct_Y, dtype = float)

    f = open('C:/Users/mr1486/Downloads/CNN_Rehab_vicon_angle/data_incorrect/D
ata_Incorrect_m02.csv')
    csv_f = csv.reader(f)
    Incorrect_X = list(csv_f)

    # Convert the input sequences into numpy arrays
    test_input1 = np.asarray(Incorrect_X)
    n_dim = 117
    incorrect_input = np.zeros((nr,timesteps,n_dim))
    for i in range(len(test_input1)//n_dim):
          incorrect_input[i,:,:] = np.transpose(test_input1[n_dim*i:n_dim*(i+1
),:])
    f = open('C:/Users/mr1486/Downloads/CNN_Rehab_vicon_angle/label_incorrect/
Labels_Incorrect_m02.csv')
    csv_f = csv.reader(f)
    Incorrect_Y = list(csv_f)

    # Convert the input labels into numpy arrays
    incorrect_label = np.asarray(Incorrect_Y, dtype = float)

    return correct_input, correct_label, incorrect_input, incorrect_label
```

In [5]:

```
# Load the data
Correct_data, Correct_label, Incorrect_data, Incorrect_label = load_data(nr, n
_dim, timesteps)

# Print the size of the data
print(Correct_data.shape, 'correct sequences')
print(Correct_label.shape, 'correct labels')
print(Incorrect_data.shape, 'incorrect sequences')
print(Incorrect_label.shape, 'incorrect labels')
```

```
(90, 240, 117) correct sequences
(90, 1) correct labels
(90, 240, 117) incorrect sequences
(90, 1) incorrect labels
```

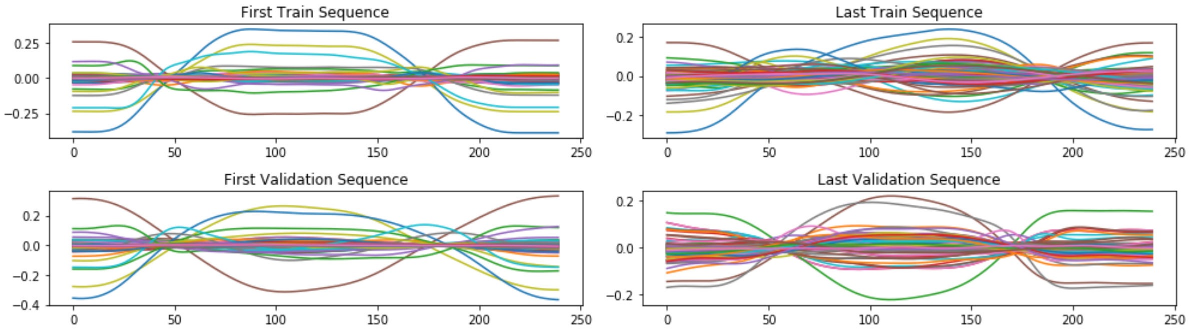
In [6]:

```
# Split the data into training and validation sets
# Training set: 70%
# Validation set: 30%

# Sample random indices
trainidx1 = random.sample(range(0,Correct_data.shape[0]),int(nr*0.7))
trainidx2 = random.sample(range(0,Incorrect_data.shape[0]),int(nr*0.7))
valididx1 = np.setdiff1d(np.arange(0,nr,1),trainidx1)
valididx2 = np.setdiff1d(np.arange(0,nr,1),trainidx2)

# Training set: data and labels
train_x = np.concatenate((Correct_data[trainidx1,:,:],Incorrect_data[trainidx2
,:,:]))
print(train_x.shape, 'training data')
train_y = np.concatenate((np.squeeze(Correct_label[trainidx1]),np.squeeze(Inco
rrect_label[trainidx2])))
print(train_y.shape, 'training labels')

# Validation set: data and labels
valid_x = np.concatenate((Correct_data[valididx1,:,:],Incorrect_data[valididx2
,:,:]))
print(valid_x.shape, 'validation data')
valid_y = np.concatenate((np.squeeze(Correct_label[valididx1]),np.squeeze(Inco
rrect_label[valididx2])))
print(valid_y.shape, 'validation labels')
```

```
(124, 240, 117) training data
(124,) training labels
(56, 240, 117) validation data
(56,) validation labels
```

In [7]:

```
# Plot the first and last sequence in the training and validation sets
plt.figure(figsize = (14,4))
plt.subplot(2,2,1)
plt.plot(train_x[0])
plt.title('First Train Sequence')
plt.subplot(2,2,2)
plt.plot(train_x[-1])
plt.title('Last Train Sequence')
plt.subplot(2,2,3)
plt.plot(valid_x[0])
plt.title('First Validation Sequence')
plt.subplot(2,2,4)
plt.plot(valid_x[-1])
plt.title('Last Validation Sequence')
plt.tight_layout()
plt.show()
```



In [8]:

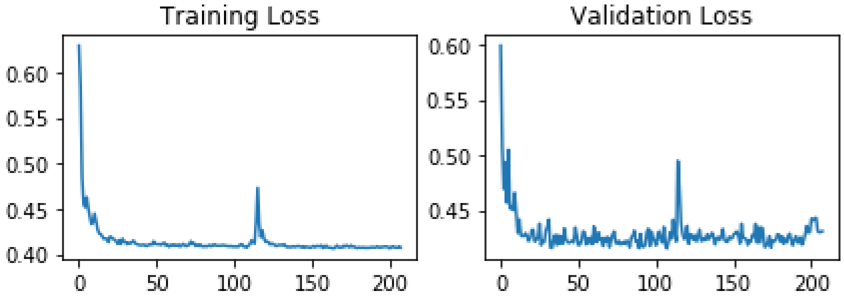
```
# Build CNN model ...
def Network():
    model = Sequential()
    model.add(Convolution1D(60, 5, padding ='same', strides = 2, input_shape =
(timesteps,n_dim)))
    model.add(LeakyReLU())
    model.add(Dropout(dropout_rate))

    model.add(Convolution1D(30, 3, padding ='same', strides = 2))
    model.add(LeakyReLU())
    model.add(Dropout(dropout_rate))

    model.add(Convolution1D(10, 3, padding ='same'))
    model.add(LeakyReLU())
    model.add(Dropout(dropout_rate))

    model.add(Flatten())
    model.add(Dense(300))
    model.add(LeakyReLU())
    model.add(Dropout(dropout_rate))


    model.add(Dense(200))
    model.add(LeakyReLU())
    model.add(Dropout(dropout_rate))

    model.add(Dense(100))
    model.add(LeakyReLU())
    model.add(Dropout(dropout_rate))

    model.add(Dense(1))
    model.add(Activation('sigmoid'))
    model.compile(loss='binary_crossentropy', optimizer=Adam())
    model.summary()

    # Early stopping if the validaton Loss does not decrease for 100 epochs
    early_stopping = EarlyStopping(monitor='val_loss', patience = 100)

    t = now()
    history = model.fit(train_x, train_y, batch_size=5, epochs=1000, verbose=0
,
                     validation_data=(valid_x, valid_y),
                     callbacks = [early_stopping])

    history.history
    print('Training time: %s' % (now() - t))
    model.save('CNN rehab_m02.h5')

    # Plot the results
    plt.figure(1)
    plt.subplot(221)
    plt.plot(history.history['loss'])
    plt.title('Training Loss')
    plt.subplot(222)
    plt.plot(history.history['val_loss'])
    plt.title('Validation Loss')
    plt.tight_layout()
    plt.show()

    # Plot the prediction of the CNN model for the training and validation set
s
    pred_train = model.predict(train_x)
    pred_test = model.predict(valid_x)
    print('prediction on test set:',pred_test)
    print('validation loss: ', history.history['val_loss'],'\n')
    print('training loss: ', history.history['loss'])
    k=len(pred_test)
    totalP=0
    for ele in range(0,k):
        totalP=totalP+pred_test[ele]
    n=float((totalP/k)*100)
    print("mean prediction: " ,n ,"%")

    plt.figure(figsize = (8,8))
    plt.subplot(2,1,1)
    plt.plot(pred_train,'s', color='red', label='Prediction', linestyle='None'
, alpha = 0.5, markersize=6)
    plt.plot(train_y,'o', color='green',label='Quality Score', alpha = 0.4, ma
rkersize=6)
    plt.ylim([-0.1,1.1])
    plt.title('Training Set',fontsize=18)
    plt.xlabel('Sequence Number',fontsize=16)
    plt.ylabel('Quality Scale',fontsize=16)
    plt.legend(loc=3, prop={'size':14}) # loc:position
    plt.subplot(2,1,2)
    plt.plot(pred_test,'s', color='red', label='Prediction', linestyle='None',
alpha = 0.5, markersize=6)
    plt.plot(valid_y,'o', color='green',label='Quality Score', alpha = 0.4, ma
rkersize=6)
    plt.title('Testing Set',fontsize=18)
    plt.ylim([-0.1,1.1])
    plt.xlabel('Sequence Number',fontsize=16)
    plt.ylabel('Quality Scale',fontsize=16)
    plt.legend(loc=3, prop={'size':14}) # loc:position
    plt.tight_layout()
    plt.savefig('C:/Users/mr1486/Downloads/CNN_Vicon_Scores_m02.png', dpi=300)
    plt.show()

    # Calculate the cumulative deviation and rms deviation for the validation
 set
    test_dev = abs(np.squeeze(pred_test)-valid_y)
    # Cumulative deviation
    mean_abs_dev = np.mean(test_dev)
    # RMS deviation
    rms_dev = sqrt(mean_squared_error(pred_test, valid_y))
    print('Mean absolute deviation:', mean_abs_dev)
    print('RMS deviation:', rms_dev)

    return mean_abs_dev, rms_dev
```

In [9]:

```
# Call the CNN model
Mean_abs_dev, RMS_dev  = Network()
```

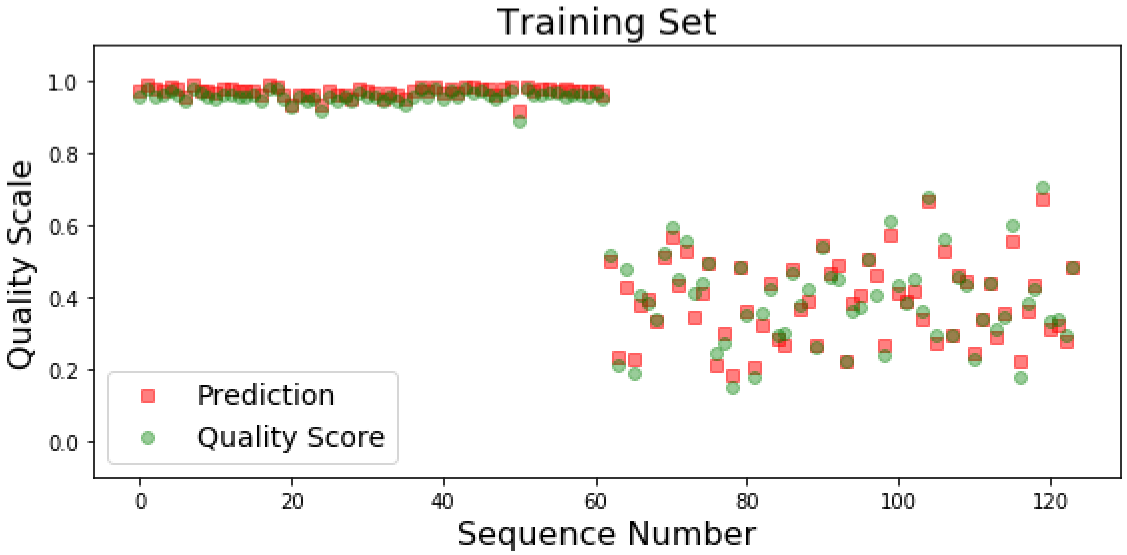
```
WARNING:tensorflow:From C:\Users\mr1486\.conda\envs\marufi\lib\site-packages
\tensorflow\python\ops\nn_impl.py:180: add_dispatch_support.<locals>.wrapper
(from tensorflow.python.ops.array_ops) is deprecated and will be removed in a
future version.
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv1d_1 (Conv1D)            (None, 120, 60)           35160     
_________________________________________________________________
leaky_re_lu_1 (LeakyReLU)    (None, 120, 60)           0         
_________________________________________________________________
dropout_1 (Dropout)          (None, 120, 60)           0         
_________________________________________________________________
conv1d_2 (Conv1D)            (None, 60, 30)            5430      
_________________________________________________________________
leaky_re_lu_2 (LeakyReLU)    (None, 60, 30)            0         
_________________________________________________________________
dropout_2 (Dropout)          (None, 60, 30)            0         
_________________________________________________________________
conv1d_3 (Conv1D)            (None, 60, 10)            910       
_________________________________________________________________
leaky_re_lu_3 (LeakyReLU)    (None, 60, 10)            0         
_________________________________________________________________
dropout_3 (Dropout)          (None, 60, 10)            0         
_________________________________________________________________
flatten_1 (Flatten)          (None, 600)               0         
_________________________________________________________________
dense_1 (Dense)              (None, 300)               180300    
_________________________________________________________________
leaky_re_lu_4 (LeakyReLU)    (None, 300)               0         
_________________________________________________________________
dropout_4 (Dropout)          (None, 300)               0         
_________________________________________________________________
dense_2 (Dense)              (None, 200)               60200     
_________________________________________________________________
leaky_re_lu_5 (LeakyReLU)    (None, 200)               0         
_________________________________________________________________
dropout_5 (Dropout)          (None, 200)               0         
_________________________________________________________________
dense_3 (Dense)              (None, 100)               20100     
_________________________________________________________________
leaky_re_lu_6 (LeakyReLU)    (None, 100)               0         
_________________________________________________________________
dropout_6 (Dropout)          (None, 100)               0         
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 101       
_________________________________________________________________
activation_1 (Activation)    (None, 1)                 0         
=================================================================
Total params: 302,201
Trainable params: 302,201
Non-trainable params: 0
_________________________________________________________________
WARNING:tensorflow:From C:\Users\mr1486\.conda\envs\marufi\lib\site-packages
```

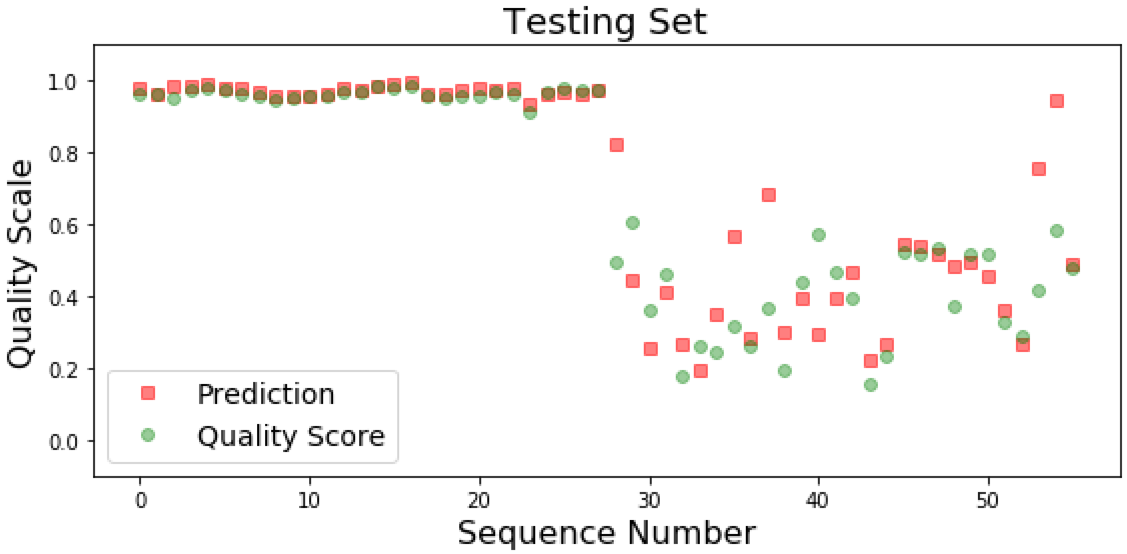
```
\keras\backend\tensorflow_backend.py:422: The name tf.global_variables is dep
recated. Please use tf.compat.v1.global_variables instead.

Training time: 0:00:39.228557
```



```
prediction on test set: [[0.97960854]
 [0.9634347 ]
 [0.9823331 ]
 [0.9850878 ]
 [0.9865984 ]
 [0.9796942 ]
 [0.9788323 ]
 [0.9683928 ]
 [0.9527361 ]
 [0.95398045]
 [0.95677847]
 [0.95989335]
 [0.97503215]
 [0.96974206]
 [0.98124254]
 [0.9861437 ]
 [0.9923247 ]
 [0.9613165 ]
 [0.9596605 ]
 [0.9697833 ]
 [0.9793377 ]
 [0.97445184]
 [0.97561824]
 [0.93295527]
 [0.9586754 ]
 [0.96624094]
 [0.96035653]
 [0.9713228 ]
 [0.82316387]
 [0.4417275 ]
 [0.25706995]
 [0.41315296]
 [0.26375663]
 [0.19465935]
 [0.3498835 ]
 [0.5636507 ]
 [0.28197938]
 [0.6803019 ]
 [0.2968827 ]
 [0.39198714]
 [0.2940114 ]
 [0.39242   ]
 [0.46531028]
 [0.22142193]
 [0.2634883 ]
 [0.5422661 ]
 [0.5359448 ]
 [0.5165394 ]
 [0.4818194 ]
 [0.49259788]
 [0.4557098 ]
 [0.35777843]
 [0.26670694]
 [0.75302505]
 [0.9458446 ]
 [0.48767444]]
validation loss:  [0.5988662519625255, 0.5085729991218874, 0.469269080353634
```

```
7, 0.49374019194926533, 0.45667015495044844, 0.504876514098474, 0.45170201680
489946, 0.45362283369260176, 0.4501050407333033, 0.4658664355852774, 0.447397
6186875786, 0.42943220159837175, 0.44165477475949694, 0.4272352156362 3634, 0.
4264369886368513, 0.42690989109022276, 0.429279998210924, 0.4265746123024395
4, 0.4220585980053459, 0.4247656626892941, 0.4321544718529497, 0.432288232658
59057, 0.4216455026928867 5, 0.4235558560384171, 0.42442613786884714, 0.437924
0891763142, 0.4183720126748085, 0.4217705178473677, 0.4209801017173699, 0.431
4705338329077, 0.435046377990927, 0.44168728510183947, 0.4161722143845899, 0.
4158410424632 6853, 0.4230881623391594, 0.4275822096637317, 0.418499176789607
3, 0.4275503690753664, 0.4195721157427345, 0.4263070284255913, 0.419838418651
90435, 0.4335504637232849, 0.42339059204927515, 0.42123889683612753, 0.420424
05530810356, 0.4216378283287798, 0.4223938456603459, 0.4207711975489344, 0.43
43682700502021, 0.4270436407199928, 0.4183318724057504, 0.41880814810948713,
0.42201059285019127, 0.4309400290782 9626, 0.4301911954368864, 0.4190197201179
607, 0.42490545340946745, 0.42045968904026915, 0.4252835961857012, 0.41925513
74435425, 0.4352966187787907, 0.4215707076447351, 0.4210917824613197, 0.42999
606100576265, 0.42296971061400007, 0.42396277827875956, 0.4175243390990155,
0.4251728970557451, 0.4218258455927883, 0.42579892703465055, 0.4164654676403 3
18, 0.4226870752338852, 0.427835916568126, 0.43078164836125715, 0.42042056870
247635, 0.4198474053825651, 0.4223660427544798, 0.41903506884617464, 0.419850
3347912005, 0.421496989737664, 0.41932562259691103, 0.4224772405411516, 0.421
1604855954647, 0.41634661571255754, 0.4254342573029654, 0.4265083795679467,
0.4315497593155929, 0.42128864782197134, 0.42281417122909, 0.415621202705161
9, 0.4286893215030432, 0.41714654623397757, 0.41676095740071367, 0.4187337051
012686, 0.4300809984228441, 0.4333045056888035, 0.4175139561827694, 0.4306800
314890487, 0.4249217552798135, 0.42205328973276274, 0.41814887816352503, 0.42
721181070165976, 0.43074627965688705, 0.4227392455296857, 0.4208524243107864,
0.4349218551069498, 0.4184667406869786, 0.41535347593682154, 0.42173313163220
88, 0.4299081691673824, 0.424190525497709, 0.4414608563695635, 0.429342942046
26766, 0.4217221832701138, 0.4948525849197592, 0.4711214594010796, 0.43873653
96001509, 0.42910372598895, 0.42394340597093105, 0.4247811308928898, 0.435867
41254798006, 0.4196907960410629, 0.4281889077808176, 0.42740232470844475, 0.4
232489650270769, 0.4274712625358786, 0.42176950962415766, 0.427217944658228 3,
0.4324919818235295, 0.4221982844173908, 0.4224806201777288, 0.42514756827482 1
53, 0.4230840653181076, 0.4283119583768504, 0.42377808557025026, 0.4262217299
2144314, 0.4295726072575365, 0.4258255884051323, 0.4190321753599814, 0.424355
86106564316, 0.4256500745458262, 0.423034724646381, 0.42671845241316725, 0.43
3123654020 6841, 0.4230865439666169, 0.4215804923857961 5, 0.42136854731610845,
0.42599376583737986, 0.4243146537670067, 0.42469825808491024, 0.4281788218234
2665, 0.4292456076613494 5, 0.4308635148086718, 0.42669702427727835, 0.4199798
317360027, 0.43774409778416157, 0.4247937976781811, 0.4247029386460781, 0.423
6810563930443, 0.41855679026671816, 0.4227936281157391, 0.42146710466 06132,
0.4276957671557154, 0.4262104723602533 3, 0.43767682037183214, 0.4205584310527
359, 0.4347015132329294, 0.4206062366387674, 0.4345055352896452, 0.4283150849
597795, 0.4164728093892336, 0.41792015571679386, 0.4212751878159387, 0.426536
9647847755, 0.41617199645510744, 0.4226615165493318, 0.42210965789854527, 0.4
2263685406318735, 0.42822946501629694, 0.4278905926538365, 0.421637062515531
3, 0.4265530298330954, 0.4196464243744101, 0.42388214543461 8, 0.4208378395331
757, 0.42683300828295095, 0.42875705286860466, 0.42230491765907835, 0.4181580
71115613, 0.42356195646737305, 0.42655049450695515, 0.42096832155116964, 0.42
04941391944885, 0.4257088242364781, 0.4188204141599791, 0.4228108831282173,
0.42852434010377954, 0.43581525262977394, 0.42969913807298454, 0.430293614310
87765, 0.4424824517752443, 0.4406927443508591, 0.4425142290336745, 0.44328898
75024557, 0.43114541843533516, 0.4299903315092836, 0.4303088848079954, 0.4310
8356185257435]

training loss:  [0.6296616823923203, 0.583814267189272, 0.48437310178433696,
0.4556544456751116, 0.4520017970954218, 0.4629801565841321, 0.450335435088603
74, 0.4418119855465428, 0.4334124297864975, 0.4356079313062852, 0.44489554063
44706, 0.4359523515787817, 0.4276035035089139, 0.4229821781476207, 0.4223333
712547056, 0.4194944154831670 7, 0.4170475794423011, 0.4176455149727483, 0.416
80040054263606, 0.4134883088690619, 0.4196889674471271, 0.418468951938 6138,
0.4173231954055448, 0.4156693001427958, 0.4143186773984663, 0.411541627659913
03, 0.4157153074779818, 0.41147806423325695, 0.4171301941237142, 0.4141069132
714502, 0.413512586946449, 0.4113434130145657 3, 0.41268908905406154, 0.412020
07330233054, 0.41329335349221386, 0.41511380408079396, 0.41244497258336316,
0.4113725743466808, 0.41072462979824315, 0.4101844244906979, 0.41008547909798
16, 0.4104268608554717, 0.4105867510361056, 0.4092702591611493, 0.40936863927
11055, 0.4107635948927172, 0.4110626861933739, 0.41050445024044285, 0.4143148
742375835, 0.411602693099168, 0.4112361323448919, 0.4109690089619929, 0.41126
812057149026, 0.4103985402372576, 0.4121199176917153, 0.4126151430030023, 0.4
097643957263039, 0.4104918123733613, 0.40842049172328365, 0.4103083042127470
5, 0.4094210142089474 4, 0.410874730156314, 0.409278996769459, 0.4097382517591
5995, 0.4107029039052225, 0.40846367036142656, 0.4097062719445075, 0.41144090
458270044, 0.41013419507972654, 0.40887133789158636, 0.41013971571960756, 0.4
103236103490476, 0.41428059844240067, 0.411651938192306, 0.4122704791445886,
0.4091878987608417, 0.4103583141921028, 0.4092250141885973, 0.410851225136749
2, 0.4090047186661151, 0.4085623725287376, 0.40912987712410187, 0.40867768444
361224, 0.4091889086750246, 0.4101690645179441, 0.40885574791219925, 0.409479
59463923206, 0.4110746267101457, 0.4097253852073223, 0.40958777526693957, 0.4
095570866859728 5, 0.4094571127526222, 0.4098892513542406, 0.410055948962127,
0.40955525985167873, 0.4093207799859585, 0.4091993667665989, 0.40881607585376
306, 0.4091428530312354, 0.40864808977611605, 0.4084334969520569, 0.408847401
21902957, 0.40825357828890124, 0.4089925402114468, 0.411067551302333, 0.41099
741297864145, 0.4084976940385757, 0.40826286507710335, 0.407480622251187 6, 0.
4097227669290958, 0.4110774950635049, 0.4153712902578615, 0.4137461038847123
4, 0.4118927150003372, 0.43541090586973774, 0.4734107132159894 3, 0.4333285847
9792075, 0.41906934471861007, 0.426244224151296, 0.417661232092688 2, 0.413844
19077827084, 0.4141598858179585, 0.41319053055297944, 0.4107816130403549 3, 0.
4116866600609595, 0.41159889222152773, 0.4100838440560525, 0.409075980345087
7, 0.40997192972610075, 0.4103417836370007, 0.4101811695483423 3, 0.4101916487
899519, 0.4110929063491283, 0.40906175201939, 0.40794056617925245, 0.40845496
95253372, 0.4078402510814128 6, 0.40845110651946837, 0.4087449331437388, 0.408
144632414464, 0.4087692747914022, 0.4091022307834318, 0.40880764564198835, 0.
40783518084114595, 0.40883151417778385, 0.4073619888194145 4, 0.40843605406342
015, 0.4080382590332339, 0.4074526728401261, 0.4073307175309427 4, 0.408310129
5175091, 0.40918319227714695, 0.40780955589106005, 0.4092678451730359, 0.4089
624780801035, 0.4087425555673338, 0.40984602068220416, 0.4071135929515285, 0.
4076000550581563, 0.4097070315432164, 0.4076302147680713, 0.407425944122575 9
3, 0.4072805228492906, 0.40672452098900275, 0.40642103264408724, 0.4078456563
574652, 0.4097887495112035, 0.407711703810961, 0.4075790602833994, 0.40752063
042694525, 0.4093480543984520 7, 0.4084313801459728, 0.40987397562111577, 0.40
897233075191897, 0.40791959387640797, 0.4097784892205269, 0.408505161202722 9
6, 0.40873797454180255, 0.40742792633752667, 0.40684878754038967, 0.407658293
60292805, 0.4073118230988902 5, 0.40725389279184804, 0.4075834611731191, 0.407
1780206455338, 0.4073104658915150 4, 0.407089953100489, 0.4073291711749569, 0.
4076916492514072, 0.4078292733719272, 0.40735933352862636, 0.407646876189016
5, 0.4087433858263877, 0.4076945099138444, 0.40746373370770483, 0.40831868446
642355, 0.4080996118005245, 0.4085631616894276, 0.40849901519475446, 0.407924
3477073408, 0.4065371214622451 3, 0.4080733194226218 7, 0.408802869819825 7, 0.4
0772995472915713, 0.4074244052171707, 0.40722193472808405, 0.408657674587542,
0.4075719821116617]
mean prediction:  70.70063018798828 %
```





```
Mean absolute deviation: 0.0613880545530244485
RMS deviation: 0.11184939590475484
```

In [10]:
```
#going through angle data of vicon sensor
```

In [11]:

```
# Function to load the data
def load_data(nr, n_dim, timesteps):
    f = open('C:/Users/mr1486/Downloads/CNN_Rehab_vicon_pos/data_correct/Data_
Correct_m02.csv')
    csv_f = csv.reader(f)
    Correct_X = list(csv_f)

    # Convert the input sequences into numpy arrays
    train_input1 = np.asarray(Correct_X, dtype = float)
    correct_input = np.zeros((nr,timesteps,n_dim))
    for i in range(len(train_input1)//n_dim):
           correct_input[i,:,:] = np.transpose(train_input1[n_dim*i:n_dim*(i+1
),:])

    f = open('C:/Users/mr1486/Downloads/CNN_Rehab_vicon_pos/label_correct/Labe
ls_Correct_pos_m02.csv')
    csv_f = csv.reader(f)
    Correct_Y = list(csv_f)

    # Convert the input labels into numpy arrays
    correct_label = np.asarray(Correct_Y, dtype = float)

    f = open('C:/Users/mr1486/Downloads/CNN_Rehab_vicon_pos/data_incorrect/Dat
a_Incorrect_m02.csv')
    csv_f = csv.reader(f)
    Incorrect_X = list(csv_f)

    # Convert the input sequences into numpy arrays
    test_input1 = np.asarray(Incorrect_X)
    n_dim = 117
    incorrect_input = np.zeros((nr,timesteps,n_dim))
    for i in range(len(test_input1)//n_dim):
          incorrect_input[i,:,:] = np.transpose(test_input1[n_dim*i:n_dim*(i+1
),:])
    f = open('C:/Users/mr1486/Downloads/CNN_Rehab_vicon_pos/label_incorrect/La
bels_Incorrect_pos_m02.csv')
    csv_f = csv.reader(f)
    Incorrect_Y = list(csv_f)

    # Convert the input labels into numpy arrays
    incorrect_label = np.asarray(Incorrect_Y, dtype = float)

    return correct_input, correct_label, incorrect_input, incorrect_label
```

In [12]:

```
# Load the data
Correct_data, Correct_label, Incorrect_data, Incorrect_label = load_data(nr, n
_dim, timesteps)

# Print the size of the data
print(Correct_data.shape, 'correct sequences')
print(Correct_label.shape, 'correct labels')
print(Incorrect_data.shape, 'incorrect sequences')
print(Incorrect_label.shape, 'incorrect labels')
```

```
(90, 240, 117) correct sequences
(90, 1) correct labels
(90, 240, 117) incorrect sequences
(90, 1) incorrect labels
```

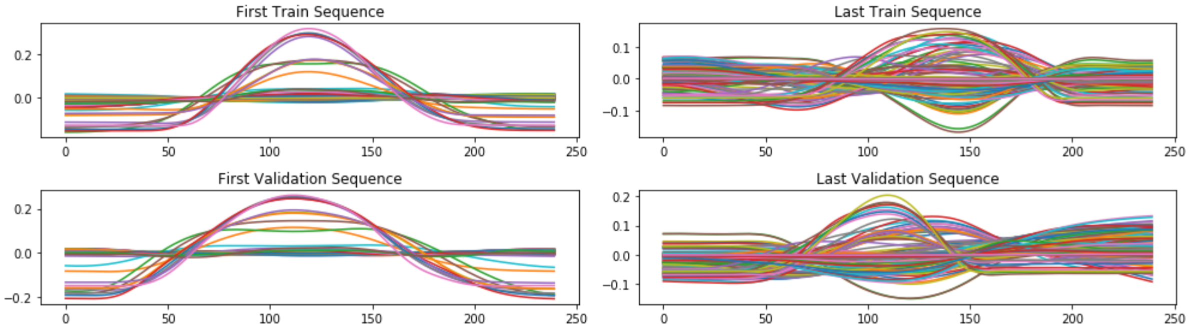
In [13]:

```
# Split the data into training and validation sets
# Training set: 70%
# Validation set: 30%

# Sample random indices
trainidx1 = random.sample(range(0,Correct_data.shape[0]),int(nr*0.7))
trainidx2 = random.sample(range(0,Incorrect_data.shape[0]),int(nr*0.7))
valididx1 = np.setdiff1d(np.arange(0,nr,1),trainidx1)
valididx2 = np.setdiff1d(np.arange(0,nr,1),trainidx2)

# Training set: data and labels
train_x = np.concatenate((Correct_data[trainidx1,:,:],Incorrect_data[trainidx2
,:,:]))
print(train_x.shape, 'training data')
train_y = np.concatenate((np.squeeze(Correct_label[trainidx1]),np.squeeze(Inco
rrect_label[trainidx2])))
print(train_y.shape, 'training labels')

# Validation set: data and labels
valid_x = np.concatenate((Correct_data[valididx1,:,:],Incorrect_data[valididx2
,:,:]))
print(valid_x.shape, 'validation data')
valid_y = np.concatenate((np.squeeze(Correct_label[valididx1]),np.squeeze(Inco
rrect_label[valididx2])))
print(valid_y.shape, 'validation labels')
```

```
(124, 240, 117) training data
(124,) training labels
(56, 240, 117) validation data
(56,) validation labels
```

In [14]:

```
# Plot the first and last sequence in the training and validation sets
plt.figure(figsize = (14,4))
plt.subplot(2,2,1)
plt.plot(train_x[0])
plt.title('First Train Sequence')
plt.subplot(2,2,2)
plt.plot(train_x[-1])
plt.title('Last Train Sequence')
plt.subplot(2,2,3)
plt.plot(valid_x[0])
plt.title('First Validation Sequence')
plt.subplot(2,2,4)
plt.plot(valid_x[-1])
plt.title('Last Validation Sequence')
plt.tight_layout()
plt.show()
```



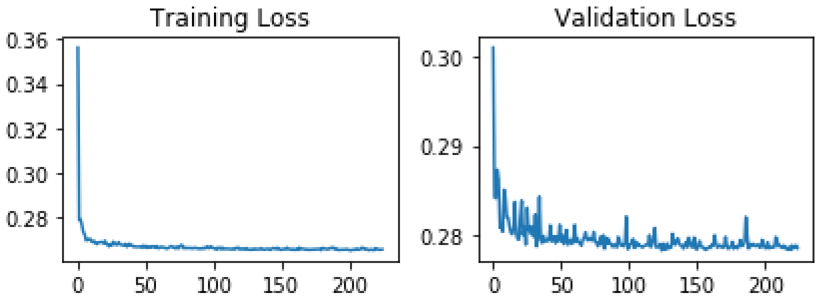
In [15]:

```
# Build CNN model ...
def Network():
    model = Sequential()
    model.add(Convolution1D(60, 5, padding ='same', strides = 2, input_shape =
(timesteps,n_dim)))
    model.add(LeakyReLU())
    model.add(Dropout(dropout_rate))

    model.add(Convolution1D(30, 3, padding ='same', strides = 2))
    model.add(LeakyReLU())
    model.add(Dropout(dropout_rate))

    model.add(Convolution1D(10, 3, padding ='same'))
    model.add(LeakyReLU())
    model.add(Dropout(dropout_rate))

    model.add(Flatten())
    model.add(Dense(300))
    model.add(LeakyReLU())
    model.add(Dropout(dropout_rate))


    model.add(Dense(200))
    model.add(LeakyReLU())
    model.add(Dropout(dropout_rate))

    model.add(Dense(100))
    model.add(LeakyReLU())
    model.add(Dropout(dropout_rate))

    model.add(Dense(1))
    model.add(Activation('sigmoid'))
    model.compile(loss='binary_crossentropy', optimizer=Adam())
    model.summary()

    # Early stopping if the validaton Loss does not decrease for 100 epochs
    early_stopping = EarlyStopping(monitor='val_loss', patience = 100)

    t = now()
    history = model.fit(train_x, train_y, batch_size=5, epochs=1000, verbose=0
,
                    validation_data=(valid_x, valid_y),
                    callbacks = [early_stopping])

    history.history
    print('Training time: %s' % (now() - t))
    model.save('CNN rehab_pos_m02.h5')

    # Plot the results
    plt.figure(1)
    plt.subplot(221)
    plt.plot(history.history['loss'])
    plt.title('Training Loss')
    plt.subplot(222)
    plt.plot(history.history['val_loss'])
    plt.title('Validation Loss')
    plt.tight_layout()
    plt.show()

    # Plot the prediction of the CNN model for the training and validation set
s
    pred_train = model.predict(train_x)
    pred_test = model.predict(valid_x)
    print('prediction on test set:',pred_test)
    print('validation loss: ', history.history['val_loss'])
    print('training loss: ', history.history['loss'])
    k=len(pred_test)
    totalP=0
    for ele in range(0,k):
        totalP=totalP+pred_test[ele]
    n=float((totalP/k)*100)
    print("mean prediction: " ,n ,"%")


    plt.figure(figsize = (8,8))
    plt.subplot(2,1,1)
    plt.plot(pred_train,'s', color='red', label='Prediction', linestyle='None'
, alpha = 0.5, markersize=6)
    plt.plot(train_y,'o', color='green',label='Quality Score', alpha = 0.4, ma
rkersize=6)
    plt.ylim([-0.1,1.1])
    plt.title('Training Set',fontsize=18)
    plt.xlabel('Sequence Number',fontsize=16)
    plt.ylabel('Quality Scale',fontsize=16)
    plt.legend(loc=3, prop={'size':14}) # loc:position
    plt.subplot(2,1,2)
    plt.plot(pred_test,'s', color='red', label='Prediction', linestyle='None',
alpha = 0.5, markersize=6)
    plt.plot(valid_y,'o', color='green',label='Quality Score', alpha = 0.4, ma
rkersize=6)
    plt.title('Testing Set',fontsize=18)
    plt.ylim([-0.1,1.1])
    plt.xlabel('Sequence Number',fontsize=16)
    plt.ylabel('Quality Scale',fontsize=16)
    plt.legend(loc=3, prop={'size':14}) # loc:position
    plt.tight_layout()
    plt.savefig('C:/Users/mr1486/Downloads/CNN_Vicon_Scores_pos_m02.png', dpi=
300)
    plt.show()

    # Calculate the cumulative deviation and rms deviation for the validation
 set
    test_dev = abs(np.squeeze(pred_test)-valid_y)
    # Cumulative deviation
    mean_abs_dev = np.mean(test_dev)
    # RMS deviation
    rms_dev = sqrt(mean_squared_error(pred_test, valid_y))
    print('Mean absolute deviation:', mean_abs_dev)
    print('RMS deviation:', rms_dev)

    return mean_abs_dev, rms_dev
```

In [16]:

```
# Call the CNN model
Mean_abs_dev, RMS_dev  = Network()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv1d_4 (Conv1D)            (None, 120, 60)           35160     
_________________________________________________________________
leaky_re_lu_7 (LeakyReLU)    (None, 120, 60)           0         
_________________________________________________________________
dropout_7 (Dropout)          (None, 120, 60)           0         
_________________________________________________________________
conv1d_5 (Conv1D)            (None, 60, 30)            5430      
_________________________________________________________________
leaky_re_lu_8 (LeakyReLU)    (None, 60, 30)            0         
_________________________________________________________________
dropout_8 (Dropout)          (None, 60, 30)            0         
_________________________________________________________________
conv1d_6 (Conv1D)            (None, 60, 10)            910       
_________________________________________________________________
leaky_re_lu_9 (LeakyReLU)    (None, 60, 10)            0         
_________________________________________________________________
dropout_9 (Dropout)          (None, 60, 10)            0         
_________________________________________________________________
flatten_2 (Flatten)          (None, 600)               0         
_________________________________________________________________
dense_5 (Dense)              (None, 300)               180300    
_________________________________________________________________
leaky_re_lu_10 (LeakyReLU)   (None, 300)               0         
_________________________________________________________________
dropout_10 (Dropout)         (None, 300)               0         
_________________________________________________________________
dense_6 (Dense)              (None, 200)               60200     
_________________________________________________________________
leaky_re_lu_11 (LeakyReLU)   (None, 200)               0         
_________________________________________________________________
dropout_11 (Dropout)         (None, 200)               0         
_________________________________________________________________
dense_7 (Dense)              (None, 100)               20100     
_________________________________________________________________
leaky_re_lu_12 (LeakyReLU)   (None, 100)               0         
_________________________________________________________________
dropout_12 (Dropout)         (None, 100)               0         
_________________________________________________________________
dense_8 (Dense)              (None, 1)                 101       
_________________________________________________________________
activation_2 (Activation)    (None, 1)                 0         
=================================================================
Total params: 302,201
Trainable params: 302,201
Non-trainable params: 0
_________________________________________________________________
Training time: 0:00:42.149119
```

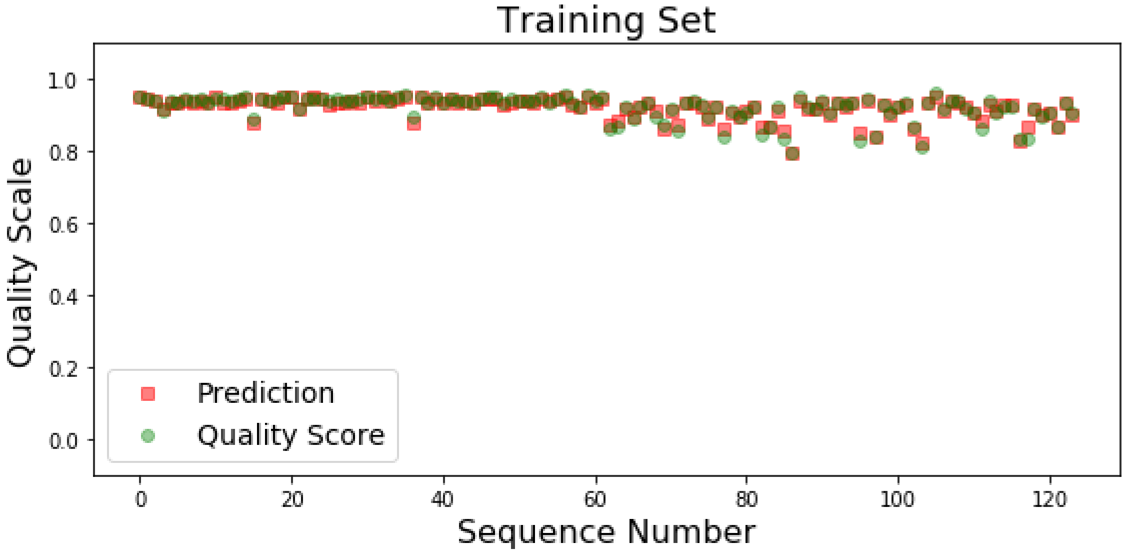

Training Loss
0.36
0.34
0.32
0.30
0.28
0
50
100
150
200
Validation Loss
0.30
0.29
0.28
0
50
100
150
200

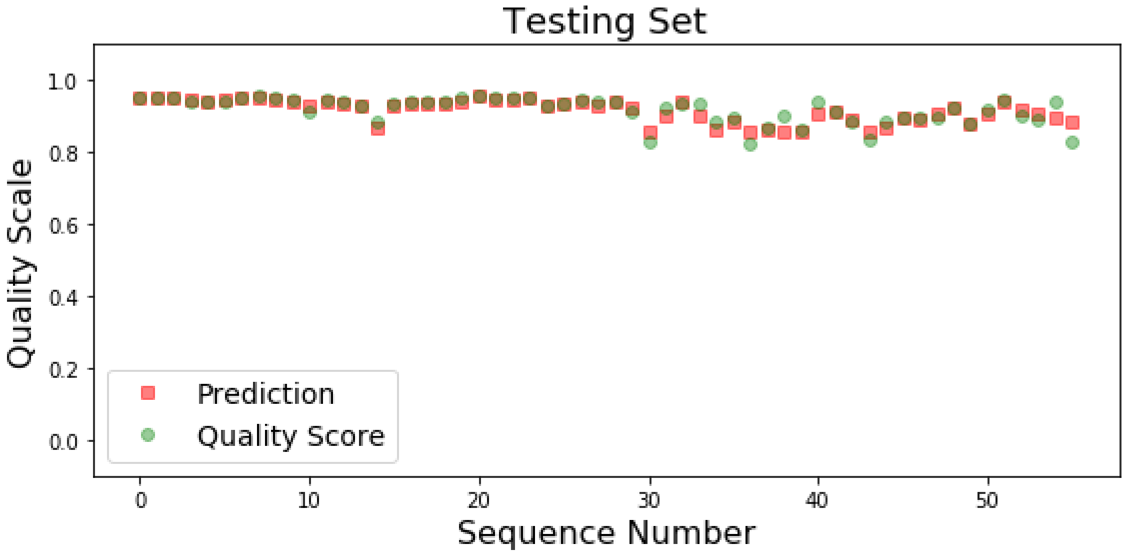
```
prediction on test set: [[0.9483851 ]
 [0.9494146 ]
 [0.94807756]
 [0.94205177]
 [0.9414059 ]
 [0.94219506]
 [0.9476739 ]
 [0.94929224]
 [0.9453542 ]
 [0.9411559 ]
 [0.9274621 ]
 [0.9379338 ]
 [0.93287647]
 [0.9295051 ]
 [0.86471236]
 [0.92698634]
 [0.93290025]
 [0.9308237 ]
 [0.9326515 ]
 [0.93734676]
 [0.9539026 ]
 [0.9468336 ]
 [0.94260585]
 [0.94978946]
 [0.9263718 ]
 [0.93051296]
 [0.9369719 ]
 [0.9302882 ]
 [0.9410773 ]
 [0.92480505]
 [0.8567771 ]
 [0.9009541 ]
 [0.93743193]
 [0.9013039 ]
 [0.85879594]
 [0.88235766]
 [0.85807574]
 [0.85927194]
 [0.8548188 ]
 [0.85797334]
 [0.9080708 ]
 [0.9093233 ]
 [0.8865714 ]
 [0.8549402 ]
 [0.8651041 ]
 [0.8957837 ]
 [0.8894851 ]
 [0.9051309 ]
 [0.9215753 ]
 [0.87611926]
 [0.9045202 ]
 [0.94047767]
 [0.9142202 ]
 [0.9034374 ]
 [0.89675015]
 [0.88381827]]
validation loss:  [0.3011096284857818, 0.2842544689774513, 0.2841449095202343
```

```
6, 0.28738024910645826, 0.2862808302577053, 0.2808246524738414, 0.28115084474
640234, 0.2803732257868625, 0.2851096865321909, 0.284249380763088, 0.2819478
309580258, 0.2819612066128424, 0.2809421167309795, 0.2802504942353283, 0.2800
874734031303, 0.280851879290172, 0.2837378201740129, 0.2800972613372973, 0.27
952355997903006, 0.2794865713055645, 0.2821322245020525, 0.2838808910122939
7, 0.28001648134418894, 0.28083288616367746, 0.27889508408095154, 0.283086099
79493277, 0.2805340276764972, 0.28095752666039125, 0.2800445660416569, 0.2809
448130428791, 0.27978208661079407, 0.2823105191013643, 0.2787140780793769, 0.
2798935206873076, 0.28433202925537315, 0.27951110766402315, 0.279089959338307
4, 0.27990743783967836, 0.27945371876869884, 0.2792371285281011, 0.2794852224
843843, 0.27939796580799986, 0.28109430202416014, 0.27927827036806513, 0.2793
894024299724, 0.2798635669584785, 0.2790655477770737, 0.27983780072203707, 0.
27936889151377337, 0.28115316161087583, 0.27926720465932575, 0.27997964700417
86, 0.2789608453001295, 0.2799099763589246, 0.2805754922862564, 0.27888905044
11289, 0.2791651790695531, 0.2794783716755254, 0.2789913447839873, 0.27945914
891149315, 0.28114694223872255, 0.2792098607335772, 0.2796412892639637, 0.279
14110012352467, 0.2790920774319342, 0.2787855107869421, 0.2794926291597741,
0.2797159845275538, 0.2803086896560022, 0.27945179279361454, 0.27939146357987
61, 0.27966777102223467, 0.27964101385857376, 0.2793596014380455, 0.280345961
9837148, 0.2793058098426887, 0.27913380281201433, 0.27871578452842577, 0.2792
012124721493, 0.27979787598763195, 0.2789269166865519, 0.2799670533942325, 0.
27844663549746784, 0.27972135932317804, 0.27898393385112286, 0.27952836028167
18, 0.27903671003878117, 0.2789412284536021, 0.2786354036735637, 0.2788862229
4579235, 0.27881372906267643, 0.2785978743008205, 0.2797890103289059, 0.27946
087904274464, 0.27878405446452753, 0.27887909007923944, 0.27871441734688623,
0.27916501462459564, 0.28208149251128944, 0.27839517939303604, 0.278532096583
92836, 0.2789601587823459, 0.27903394401073456, 0.2795509570943458, 0.2785434
411572559, 0.27924863940903116, 0.2787298373878002, 0.2790102096540587, 0.278
8007916616542, 0.2786805408873728, 0.2785975267844541, 0.2787611207791737, 0.
2788208017924002, 0.27895877164389404, 0.2788721879145929, 0.279987489804625
5, 0.2786230921213116, 0.2788824960589409, 0.2796990844820227, 0.280795748744
6921, 0.27858696452208925, 0.2786500815834318, 0.2786999001566853, 0.27898622
70333937, 0.2782490836190326, 0.2790295433785234, 0.27844296608652386, 0.2783
2774525242193, 0.2790453386093889, 0.27845150684671743, 0.27860031010849134,
0.2785832927163158, 0.280148252046534, 0.2791847943195275, 0.2788938890610422
3, 0.2793943623879126, 0.2796272419925247, 0.27896314593298094, 0.27883620826
261385, 0.27883285842835903, 0.27846737844603403, 0.27891811249511583, 0.2792
0084313622545, 0.27861629985272884, 0.2797025003071342, 0.27926431064094814,
0.27864559553563595, 0.27863391142870697, 0.2797219106661422, 0.2785015382937
023, 0.2783469139997448, 0.27892862340169294, 0.2793224682765348, 0.278651107
900909, 0.27866001932748724, 0.2785724393491234, 0.278307407828314, 0.2785099
470721824, 0.2784665195005281, 0.2786496765911579, 0.27853174933365415, 0.278
88051979243755, 0.27878353611699175, 0.2799809495253222, 0.27904533807720455,
0.2785692542259182, 0.27871275453695227, 0.27887350214379175, 0.2786707867469
1065, 0.27855590198721203, 0.27864494547247887, 0.27979205389107975, 0.278811
70782659737, 0.27842921178255764, 0.27857962250709534, 0.27868514747491907,
0.27923649416438173, 0.27845410417233196, 0.2787106779537031, 0.2786407558513
539, 0.27847957983613014, 0.2789176854171923, 0.27944827558738844, 0.27873096
93438666, 0.2788825748222215, 0.27913286217621397, 0.2820402498223952, 0.2799
7678704559803, 0.27846743459148066, 0.278971739379423, 0.2787184909518276, 0.
2790026741900614, 0.2789398335984775, 0.27873221784830093, 0.2788500200424875
5, 0.27893860292221817, 0.27858448480921133, 0.27861600794962477, 0.279452845
18812386, 0.2788788431457111, 0.2785396783479622, 0.278503238090447, 0.279179
2447013514, 0.27849926905972616, 0.2785704505762884, 0.27857189119926523, 0.2
7873263508081436, 0.27862190880945753, 0.2797589427126305, 0.2794339505157300
6, 0.27869714078094276, 0.27880765683948994, 0.27869834857327597, 0.278521461
67840275, 0.278809950287853, 0.27847703972033094, 0.2785722584064518, 0.27831
```

```
88998167004, 0.2786449694207736, 0.2783168059374605, 0.278829162408198, 0.2
7848201830472263, 0.278605601085083, 0.278830107301473, 0.27850271255842]
training loss:  [0.3561545955317636, 0.27894964561827723, 0.2791650858857939,
0.2767649100672814, 0.2735569509527376, 0.2726351509411489, 0.2698687132327
87, 0.2699756039486777, 0.2703610655040510, 0.2705007640104 14, 0.2694357471
360314, 0.2690433683174272, 0.2695247238922503, 0.2686072669682964, 0.268195
0142066325, 0.2688470475615994, 0.2689881115671127, 0.2690849417159634, 0.269
0446279462307, 0.2684907946855791, 0.2696336009089023, 0.2681928480104092,
0.2679633242228338, 0.2671393654759853, 0.2681189440190792, 0.2676280041135
27, 0.269220708478843, 0.268100455042816, 0.2682639887977031, 0.267780994215
1926, 0.2690519840726929, 0.2681858252373434, 0.2678507407826762, 0.26765179
92250381, 0.2676668256040542, 0.2682861388931351, 0.2670284287343102, 0.267
54536611899254, 0.2683344394449264, 0.2676122967273958, 0.267570847705487,
0.2669887738602777, 0.2671048504210287, 0.2669325963624062, 0.2669723305490
094, 0.2669423455672879, 0.2670207625675586, 0.2666415832936764, 0.267483302
18176686, 0.2668853599217630, 0.2673783338358325, 0.266461160394453, 0.266
71735293442206, 0.267127446710102, 0.2666338126986257, 0.2671680887860636,
0.267113036085521, 0.2665737122297287, 0.2672980281133805, 0.26634501389438
1, 0.266406635843938, 0.2669249670880456, 0.2667928186635817, 0.2668054467
9684026, 0.2664558188088478, 0.2663186678001957, 0.266143554761525, 0.2666
650823047084, 0.2669481579815188, 0.2672621780826199, 0.2666662229645636,
0.2667013505293477, 0.2661972532589589, 0.2669956079654155, 0.266191256863
556, 0.266889490427509, 0.2677457204028483, 0.267425104374847, 0.266258223
52305534, 0.266511789493022, 0.2663949774878640, 0.2664242761750375, 0.2664
010480286614, 0.2664517429807493, 0.266228260772843, 0.2662842726755526,
0.2665687005125707, 0.266370653264945, 0.2662628407199536, 0.2660862628010
117, 0.2662580143059453, 0.2660735789085588, 0.266440624671597, 0.266179760
4974239, 0.266455601540304, 0.2671338280122126, 0.266319244979850, 0.266562
1741644798, 0.2671631346306493, 0.2665297254439323, 0.2663993447298003, 0.
2661766028932987, 0.2656958686007607, 0.2666336319619609, 0.266430126803536
6, 0.2659468840687505, 0.2667285276036109, 0.2659431298173243, 0.2661517004
918934, 0.2660167890210305, 0.2660919759542711, 0.2660113456508806, 0.26598
890870809555, 0.2658897041072768, 0.266030402193146, 0.2657494697599641,
0.2663852547205264, 0.266332505691436, 0.2666246762439128, 0.265857266682770
3, 0.2658981773401460, 0.2662386889419248, 0.2658974467025649, 0.265769842
5158254, 0.2663567363014144, 0.2658785550344374, 0.265728504066505, 0.2655
9301850295836, 0.2658942471588811, 0.265680548524664, 0.2660835018802073,
0.2657600360051278, 0.265917460404096, 0.2658561834644887, 0.26595299751047
63, 0.2663154086518672, 0.265793976644354, 0.2657375139815192, 0.2657686982
643695, 0.2659869226476838, 0.2661149162919290, 0.265649205374140, 0.26566
35592781728, 0.2661873152659785, 0.266403104028394, 0.2662455753212975, 0.2
658959493762062, 0.266530354897822, 0.2661992727268126, 0.265601035808363
5, 0.2657571336434733, 0.2658712379634380, 0.2659148975245414, 0.265821218
85111254, 0.2666071885295452, 0.2660480248591592, 0.2658502627764978, 0.26
565774518155283, 0.265960251131365, 0.265533052505024, 0.2657369211796791,
0.265919292886411, 0.2655394536352927, 0.2666171773547126, 0.266008989344681
5, 0.2659778724755010, 0.2660542682053581, 0.2658090174438492, 0.265638453
0480366, 0.265857527813603, 0.2656433695266323, 0.2655886767372008, 0.26575
74829795668, 0.2658156804259746, 0.265634009314160, 0.2658992559919434, 0.2
658273094604092, 0.2659610688926712, 0.2659368397247407, 0.265475474778682
4, 0.2663621275175002, 0.2663612214307631, 0.2660219783504163, 0.266005824
96343, 0.2657431116507899, 0.2661322122139315, 0.2665028646588325, 0.266371
6002097053, 0.266614753392434, 0.2659541912857563, 0.2663670495873497, 0.26
5664002707889, 0.265552778277666, 0.2655810910847879, 0.2657085848191092,
0.2657123754822438, 0.265720597077761, 0.2657679745987538, 0.265824707285
504, 0.2656835762243117, 0.265483354608858, 0.265301378023239, 0.265645855
776506, 0.265583598926182, 0.265607451359110, 0.2661184084511572, 0.265774
```

```
89883188277, 0.2656565117499521, 0.2659086575431208, 0.2665723421640934, 0.26
64132608521369, 0.2657769657671451б, 0.26604802137421024, 0.2658606762847593,
0.26542885421264556, 0.2656459016424994, 0.26578685461032775, 0.2656366366051
858, 0.26541415998531925, 0.2663016122195028, 0.2661463812954964, 0.265683001
5682405, 0.2656783439940022, 0.26565864273617346, 0.26582267075296373]
mean prediction:  91.45439147949219 %
```





```
Mean absolute deviation: 0.011117720295180017
RMS deviation: 0.016513364540712887
```