Assignment 2 CNN AND ANN USING CROSS FOLD

Name: Aloukik Aditya

student_ID: 1115290

In [1]:
```
import pandas as pd#-------------------------------------LOading all libraries h
from pandas import DataFrame
import numpy as np
import random
import matplotlib.pyplot as plt
import seaborn as sns
from mpl_toolkits.mplot3d import Axes3D
from math import*
import itertools
import re
from sklearn.preprocessing import MinMaxScaler
import os
from scipy.spatial.distance import pdist,squareform
import tensorflow as tf
from sklearn.model_selection import train_test_split
#import tensorflow_docs as tfdocs

from tensorflow import keras
from tensorflow.keras import layers
import numpy as np
import pandas as pd
import pathlib

import matplotlib.pyplot as plt
import os
from sklearn.model_selection import KFold, StratifiedKFold

from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

In [2]:
```
from sklearn.datasets import load_boston
from keras.models import Sequential
from keras.layers import Dense, Conv1D, Flatten
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt
```

In [ ]:
```

```

In [37]:
```
def comparison_graph(predict, orignal):#------------------------function for com
    x_ax = range(len(predict))
    plt.scatter(x_ax, orignal, s=5, color="blue", label="original")
    plt.plot(x_ax, predict, lw=0.8, color="red", label="predicted")
    plt.legend()
    plt.show()
```

In [40]:
```
def comparison_graph_rmse(ann_rmse, cnn_rmse):#------------------------function
    x_ax = range(len(ann_rmse))
    plt.plot(x_ax, ann_rmse, lw=0.8, color="blue", label="ANN_rmse")
    plt.plot(x_ax, cnn_rmse, lw=0.8, color="red", label="CNN_rmse")
    plt.legend()
    plt.show()
```

In [5]:
```
def get_values(dataset):
    df = DataFrame (dataset)
    new_np = np.asarray(df)

    strr1 = new_np###############-------------------removing all the brackets ar
    strr_removved = []
    strr_removved.clear()
    for i in range(strr1.size):
        strr_removved.append(strr1[i][0].replace("[", "").replace("]", "").replac



    b =[]#------------------------------------ creating list of all numbers onl
    b.clear()
    for i in range(strr1.size):
        a = [int(s) for s in strr_removved[i].split() if s.isdigit()]
        b.append(a)

    #------------------merging all the digit into single list

    merged = list(itertools.chain(*b))
    roww = np.asarray(merged)

    for col in df.columns:
        valuee = col

    #-------------------------------------getting optimized value
    j = str(re.findall(r'[\d\.\d]+', valuee))
    l=j.replace("[","").replace("]","").replace("'","").replace("'","")

    target_val = float(l)

    coll = DataFrame(roww)
    df_rows = coll.T

    df_rows['Optimized_value'] = target_val

    #df_target = DataFrame(target_val)

    #print(df_rows)
    #print(type(target_val))

    #print(df_target)

    return(df_rows)
```

In [6]:
```
rmse_list_ann = []
rmse_list_ann.clear()
def ANN_model(X_train,Y_train,X_test,Y_test):#------------------creating ann mode

    def getModel():
        model = Sequential()
        model.add(Dense(200, input_dim = 2500, activation='relu'))
        model.add(Dense(75, activation='relu'))
        model.add(Dense(50, activation='relu'))
        model.add(Dense(25, activation='relu'))
        model.add(Dense(1, activation='linear'))
        model.compile(optimizer = 'RMSprop', loss = 'mean_squared_error')
        return model
    model = getModel()

    verbose, epochs, batch_size = 1, 20, 1
    def showResults(test, pred):
        mse=mean_squared_error(test, pred)
        rmse = sqrt(mse)
        print("RMSE: ", rmse)
        rmse_list_ann.append(rmse)
        return rmse
    model.load_weights('1115290-ANN.h5')
    # Train the model with training data
    history = model.fit(X_train, Y_train, validation_split = 0.3, batch_size = ba

    yPredict = model.predict(X_test)
    showResults(Y_test, yPredict)
    #return showResults(Y_test, yPredict)
    model.save('1115290-ANN.h5')
    print("===========  Printing graph for ANN MODEL  ================")

    comparison_graph(yPredict,Y_test)
```

In [7]:
```
rmse_list_cnn = []
rmse_list_cnn.clear()
def CNN_model(X_train,Y_train,X_test,Y_test):#-------------------------creating

    X_train = X_train.reshape(X_train.shape[0], X_train.shape[1], 1)
    X_test = X_test.reshape(X_test.shape[0], X_test.shape[1], 1)
    #model.load('1115290-CNN.h5')
    def getModel():
        model = Sequential()
        model.add(Conv1D(32, 2, activation="relu", input_shape=(2500, 1)))
        model.add(Flatten())
        model.add(Dense(64, activation="relu"))
        model.add(Dense(1))
        model.compile(loss="mse", optimizer="adam")
        return model
    model = getModel()

    verbose, epochs, batch_size = 1, 20, 1
    def showResults(test, pred):
        mse=mean_squared_error(test, pred)
        rmse = sqrt(mse)
        rmse_list_cnn.append(rmse)
        print("RMSE: ", rmse)
    model.load_weights('1115290-CNN.h5')
    # Train the model with training data
    history = model.fit(X_train, Y_train, validation_split = 0.3, batch_size = ba

    yPredict = model.predict(X_test)
    showResults(Y_test, yPredict)
    #return showResults(Y_test, yPredict)
    print("===========  Printing graph for CNN MODEL  ================")
    model.save('1115290-CNN.h5')
    comparison_graph(yPredict,Y_test)
```

In [8]:
```
```

In [9]:
```
import glob
path = "DS/data*.csv"#------------------------------ this block will get all fil
count = 0
file = []
for fname in glob.glob(path):

    #print(count)
    file.append(fname)
    count = count+1
    #print(fname)
```

In [10]:
```
dframe = []
dataset_collection = []#---------------------------------reading multiple csv fil
for i in range(len(file)):
    dataset_collection.append(pd.read_csv(file[i]))
    #dframe.append(get_values(dataset[i]))
```

In [ ]:
```
```

In [11]:
```
dframe = []

for i in range(len(dataset_collection)):#------------------------------creating
    dframe.append(get_values(dataset_collection[i]))
```

In [12]:
```
Final_dataset = pd.concat(dframe)#--------will combine all dataframes into single
```

In [13]:
```
Final_dataset
```

Out[13]:

	0	1	2	3	4	5	6	7	8	9	...	2491	2492	2493	2494	2495	2496
0	156	499	284	25	300	40	346	108	190	458	...	348	312	420	295	327	68
0	84	46	272	52	329	217	387	107	337	69	...	22	417	390	66	249	285
0	191	477	275	65	216	340	306	195	146	129	...	324	68	483	50	191	56
0	36	33	283	289	440	117	173	450	235	125	...	352	494	207	127	97	220
0	100	451	272	298	496	351	86	369	122	372	...	432	126	393	242	51	156
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
0	348	315	411	167	254	206	204	431	377	24	...	48	394	419	128	466	41
0	170	195	24	112	219	437	256	114	292	160	...	462	117	390	348	88	429
0	342	401	112	480	22	155	395	393	392	199	...	349	250	43	458	221	127
0	320	497	233	82	286	125	265	282	262	213	...	327	98	45	265	237	360
0	472	172	83	478	113	297	187	387	166	200		478	271	173	476	146	302

In [14]:
```
target = Final_dataset.loc[:,"Optimized_value"]#------------------tarrget values
```

In [15]:
```
Features = Final_dataset.drop(labels='Optimized_value', axis=1)#--------------ext
```

In [16]:
```
target
```

Out[16]:
```
0    1606.0
0    1714.0
0    1785.0
0    1784.0
0    1686.0
      ...
0    1729.0
0    1819.0
0    1721.0
0    1827.0
0    1744.0
Name: Optimized_value, Length: 1000, dtype: float64
```

In [17]:
```
Features
```

Out[17]:

	0	1	2	3	4	5	6	7	8	9	...	2490	2491	2492	2493	2494	2495	2
0	156	499	284	25	300	40	346	108	190	458	...	315	348	312	420	295	327	
0	84	46	272	52	329	217	387	107	337	69	...	444	22	417	390	66	249	
0	191	477	275	65	216	340	306	195	146	129	...	43	324	68	483	50	191	
0	36	33	283	289	440	117	173	450	235	125	...	260	352	494	207	127	97	
0	100	451	272	298	496	351	86	369	122	372	...	210	432	126	393	242	51	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
0	348	315	411	167	254	206	204	431	377	24	...	72	48	394	419	128	466	
0	170	195	24	112	219	437	256	114	292	160	...	107	462	117	390	348	88	
0	342	401	112	480	22	155	395	393	392	199	...	222	349	250	43	458	221	
0	320	497	233	82	286	125	265	282	262	213	...	86	327	98	45	265	237	
0	472	172	83	478	113	297	187	387	166	200	...	478	478	271	173	476	146	

1000 rows × 2500 columns

In [18]:
```
norm = MinMaxScaler().fit(Features)#------------------normalizing data here
Features_norm = norm.transform(Features)
```

In [19]:
```
Features_norm.shape
```

Out[19]:
```
(1000, 2500)
```

In [20]:
```
target.shape
```

Out[20]:
```
(1000,)
```

In [21]:
```
data = np.asarray(Final_dataset)
```

In [22]:
```
data
```

Out[22]:
```
array([[ 156.,  499.,  284., ...,  302.,   47., 1606.],
       [  84.,   46.,  272., ...,  494.,  223., 1714.],
       [ 191.,  477.,  275., ...,   69.,  380., 1785.],
       ...,
       [ 342.,  401.,  112., ...,  316.,  452., 1721.],
       [ 320.,  497.,  233., ...,  321.,  440., 1827.],
       [ 472.,  172.,   83., ...,   89.,  314., 1744.]])
```

main loops starts here, it contains ann and cnn using cross fold , where k = 5

In [23]:
```
from numpy import array
from sklearn.model_selection import KFold#----------------------------MAIN LOOP

# data sample
#data = array([0.1, 0.2, 0.3, 0.4, 0.5, 0.6])
#---------------------------------- prepare cross validation
kfold = KFold(5, True, 1)
k_train = []
k_test = []
k_train.clear()
k_test.clear()
# enumerate splits
count = 1
for train, test in kfold.split(data):

    data_train = data[train]#-------------splitting data here
    X_train = data_train[:,0:2500]
    norm = MinMaxScaler().fit(X_train)#-------------------normalizing training da
    X_train = norm.transform(X_train)
    Y_train = data_train[:,2500]

    data_test = data[test]
    X_test = data_test[:,0:2500]
    norm = MinMaxScaler().fit(X_test)#-------------------normalizing testin data
    X_test = norm.transform(X_test)

    Y_test = data_test[:,2500]



    #--------------------------after here we are ready with training and testing
    #----time to feed into CNN and ANN




    #----function for neural network
    print("-----------------------position k-",+ count,  'train: %s, test: %s' %
    print("++++++++++++++++++++++++++  ANN STARTING HHERE  +++++++++++++++")
    ANN_model(X_train,Y_train,X_test,Y_test)#--------------calling Ann function

    print("++++++++++++++++++++++++++  CNN STARTING HHERE  +++++++++++++++")
    CNN_model(X_train,Y_train,X_test,Y_test)#---------------calling cnn function

    count = count+1
```

```
560/560 [==============================] - 3s 5ms/step - loss: 626.7328 - val
_loss: 9505.5654
Epoch 8/20
560/560 [==============================] - 3s 5ms/step - loss: 624.0784 - val
_loss: 9011.7881
Epoch 9/20
560/560 [==============================] - 3s 5ms/step - loss: 587.5747 - val
_loss: 9202.7168
Epoch 10/20
560/560 [==============================] - 3s 5ms/step - loss: 590.0748 - val
_loss: 9002.0850
```

```
Epoch 11/20
560/560 [==============================] - 3s 5ms/step - loss: 625.0007 - val
_loss: 9712.6260
Epoch 12/20
560/560 [==============================] - 3s 5ms/step - loss: 602.3136 - val
_loss: 9548.2471
Epoch 13/20
560/560 [==============================] - 3s 5ms/step - loss: 571.9328 - val
 loss: 10597.3545
```

In [24]:
```
Average_rmse_ann = (sum(rmse_list_ann)/5)#------------------calcilate average rm
Average_rmse_cnn = (sum(rmse_list_cnn)/5)
```

In [ ]:
```
```

From the following result CNN is better than ANN

In [26]:
```
rmse_list_ann
```

Out[26]:
```
[73.09410040043167,
 89.02007497501361,
 59.641254099416095,
 65.26476829066189,
 53.30310055496208]
```

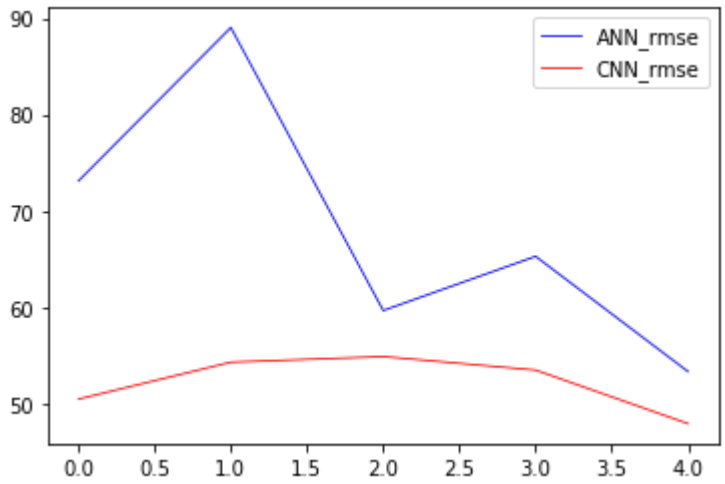
In [27]:
```
rmse_list_cnn
```

Out[27]:
```
[50.4435751561129,
 54.27775676952652,
 54.85924666837283,
 53.45600570667547,
 47.894752686474554]
```

In [42]:
```
print("AVERAGE RMSE OF ANN IS", + Average_rmse_ann)#---------------comapring mod
print("AVERAGE RMSE OF CNN IS", + Average_rmse_cnn)
```

```
AVERAGE RMSE OF ANN IS 68.06465966409708
AVERAGE RMSE OF CNN IS 52.186267397432445
```

In [41]:
```
comparison_graph_rmse(rmse_list_ann,rmse_list_cnn)#----------------performace met
```



In [ ]:
```

```