National College of Ireland

Configuration Manual

MSc Research Project

MSc in Data Analytics

Vikas Chhikara

x18194095

National college of Ireland

Supervisor: Dr. Catherine Mulwa

National College of Ireland

Project Submission Sheet – 2019/2020

Student Name:	Vikas chhikara		
Student ID:	X18194095		
Programme:	MSc in data Analytics	**Year:**	2019/20
Module:	MSc Research Project		
Lecturer:	Dr Catherine Mulwa		
Submission Due Date:	17 August 2020		
Project Title:	Aircraft Predictive Maintenance		
Word Count:	1300		

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the references section. Students are encouraged to use the Harvard Referencing Standard supplied by the Library. To use other author's written or electronic work is illegal (plagiarism) and may result in

disciplinary action. Students may be required to undergo a viva (oral examination) if there is suspicion about the validity of their submitted work.

Signature: Vikas

Date: 17th August 2020

PLEASE READ THE FOLLOWING INSTRUCTIONS:

1. Please attach a completed copy of this sheet to each project (including multiple copies).
2. Projects should be submitted to your Programme Coordinator.
3. **You must ensure that you retain a HARD COPY of ALL projects**, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer. Please do not bind projects or place in covers unless specifically requested.
4. You must ensure that all projects are submitted to your Programme Coordinator on or before the required submission date. **Late submissions will incur penalties.**
5. All projects must be submitted and passed in order to successfully complete the year. **Any project/assignment not submitted will be marked as a fail.**

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Vikas Chhikara(x18194095)

1. Introduction:

In this section system setup which include hardware and software requirement and implementation and evaluation of technique is explained in this configuration manual

2. System requirement:

2.1 Hardware

System hardware requirement is intel core i7 – 4150 CPU @ 2.5 ghz,AMD rayzen R 7; 12 GB RAM , 1 TeraByte hard disk; and Window 8 64 bit

2.2 Software

spyder IDE for running all deep learning models on NASA aircraft predictive maintenance dataset on IDE of python that is accessed by any browser.

Various python library to plot and understand data and extract and select the feature of the time series aircraft dataset.

3. Data:

Data is prepared includes all the data cleaning , transforming and then select the important feature in the dataset between all the attributes , for this project open portal NASA dataset has been used which is download from NASA data portal website, the dataset gives information about the number of cycle aircraft engine going to run, operational setting and sensor settings which include , then dataset is checked whether there is any null values and not needed columns or have any special characters then it is replaced.

3.1 Data understanding:

As the data contain 24 features or attribute so few plot have been done to better understand the data so in figure 1 based on ID number the various plot have been generated in which all attribute value and their occurrence frequency has been shown.

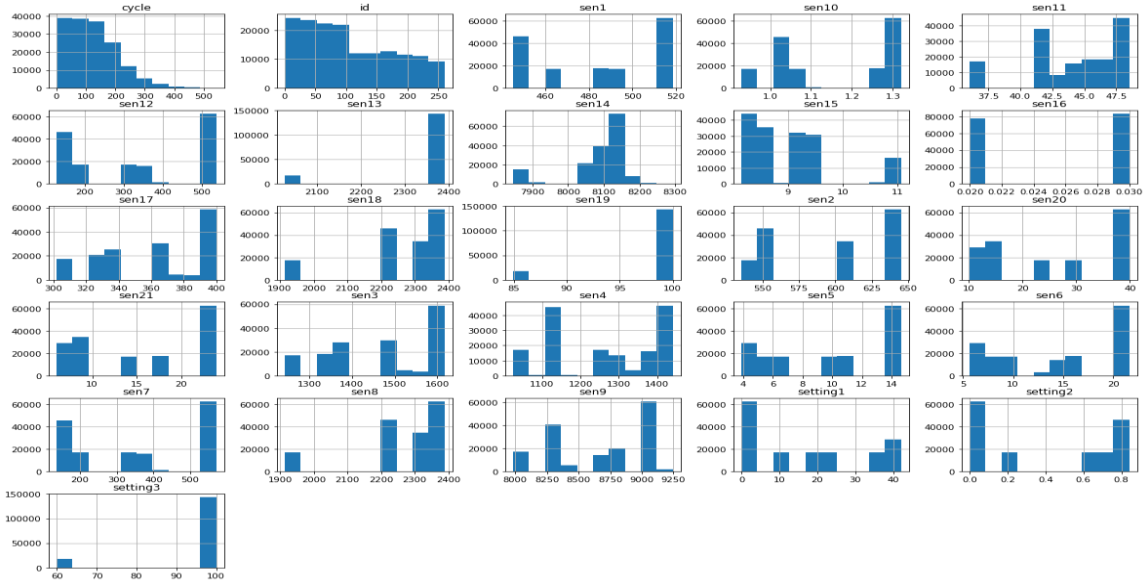

Figure 1: Based on cycle number values of all the setting

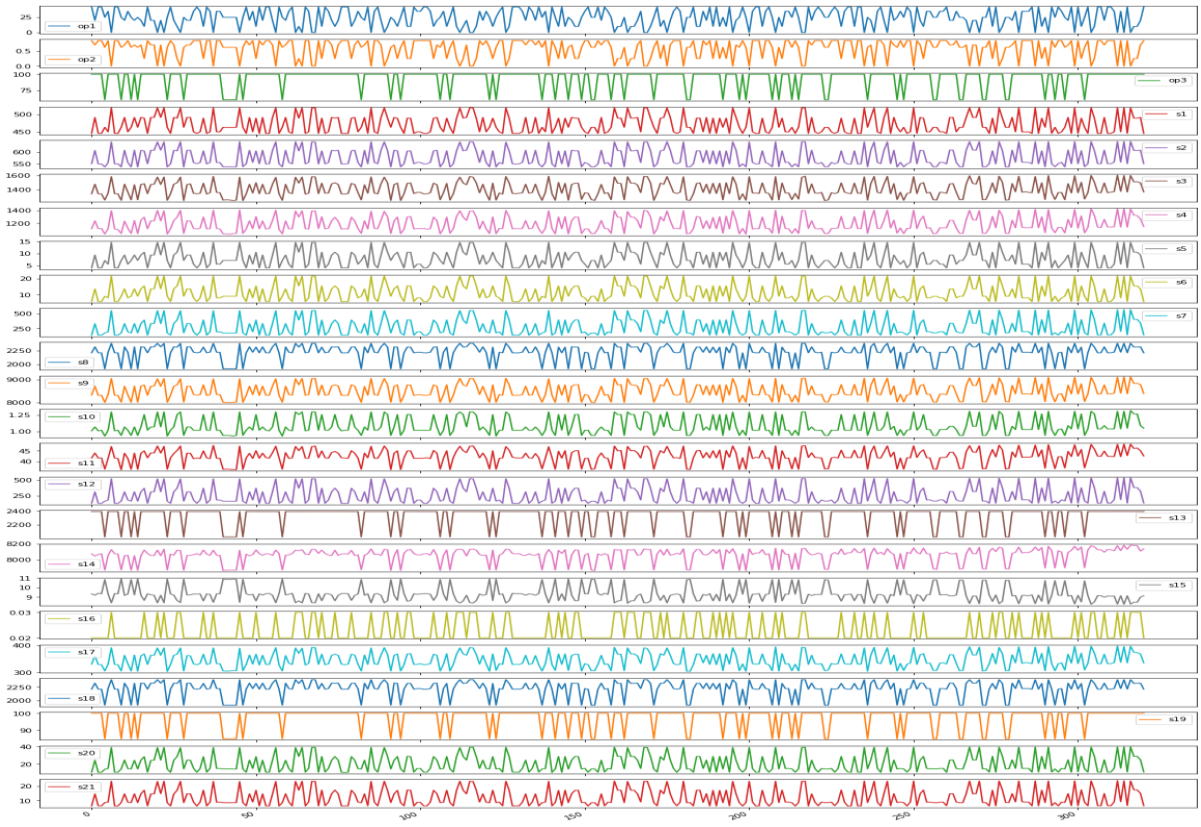

Figure 2: all settings plot of based on their value of engine ID 1

In figure 2 the engine ID number 1 all the attribute is plotted that between what values these attribute values are occurring.

3.2 Data preparation:

Scaling data:

Here scaling the data is done and returned the scaled data in the form of data frame and will generate the scaler if does not pass it

```
def scaleData(data, scaler=None):
    scaled_fields = setting_cols+sensor_cols
    if scaler == None:
        scaler = StandardScaler().fit(data[scaled_fields].values)
#         scaler = MinMaxScaler().fit(data[scaled_fields].values)
    scaled_data = scaler.transform(data[scaled_fields].values)
    scaled_df0 = pd.DataFrame(scaled_data)
    scaled_df0.columns = scaled_fields
    scaled_df1 = data.copy()
    for i in range(len(scaled_fields)):
        theField = scaled_fields[i]
        scaled_df1[theField] = scaled_df0[theField]
    return scaled_df1, scaler
```

```
# Scaled train
scaled_train, scaler = scaleData(train)
# Scaled test
scaled_test, scaler = scaleData(test, scaler)
```

Figure 3 : Scaling the data

Data augmentation and padded sequence:

After scaling the data , the augmentation and padding is done on the data based on feature order then data is plotted to see the difference it make

```
In [25]: import random
def getPieceWiseData(data, augmentStartCycle=None, augmentEndCycle=None, movingAverage=None):
    uniqueIds = data['id'].unique()
    if movingAverage==None:
        result = [data[data['id']==mId].values for mId in uniqueIds]
    else:
        result = [data[data['id']==mId].rolling(movingAverage).mean().dropna().values for mId in uniqueIds]
    maxlen = np.max([len(x) for x in result])
    #Augment the data now
    if(augmentStartCycle!=None and augmentEndCycle!= None):
        result1 = []
        for mc in result:
            maxCycle = len(mc)
            for i in range(50):
                idx = random.randint(max([maxCycle-145, 10]), max([maxCycle-10, 10]))
                if(len(mc[:idx, :])>0):
                    result1.append(mc[:idx, :])
            #Also add the complete sequence.
#             result1.append(mc)
        result = result1
    # calculate the ruls (-1) is the last column for RUL
    ruls = [min(mc[:, -1]) for mc in result]
    return result,ruls, maxlen
# Use this last one only (prev one is a helper)
from keras.preprocessing.sequence import pad_sequences
def getPaddedSequence(data, pad_type='pre', maxlen=None, augmentStartCycle=None, augmentEndCycle=None, movingAverage=None):
    piece_wise, ruls, ml = getPieceWiseData(data, augmentStartCycle, augmentEndCycle, movingAverage)
    if(maxlen==None): maxlen = ml
    padded_sequence = pad_sequences(piece_wise, padding=pad_type, maxlen=maxlen, dtype='float32')
    return padded_sequence, ruls, maxlen

Using TensorFlow backend.

In [26]: augmentStartCycle = 130
augmentEndCycle = 362
maxlen=200
movingAverage = None
padded_train, train_ruls, maxlen = getPaddedSequence(scaled_train, maxlen=maxlen, augmentStartCycle=augmentStartCycle, augmentEn
dCycle=augmentEndCycle, movingAverage=movingAverage)
padded_test, test_ruls, maxlen = getPaddedSequence(scaled_test, maxlen=maxlen, movingAverage=movingAverage)
```

Figure 4: Data padding

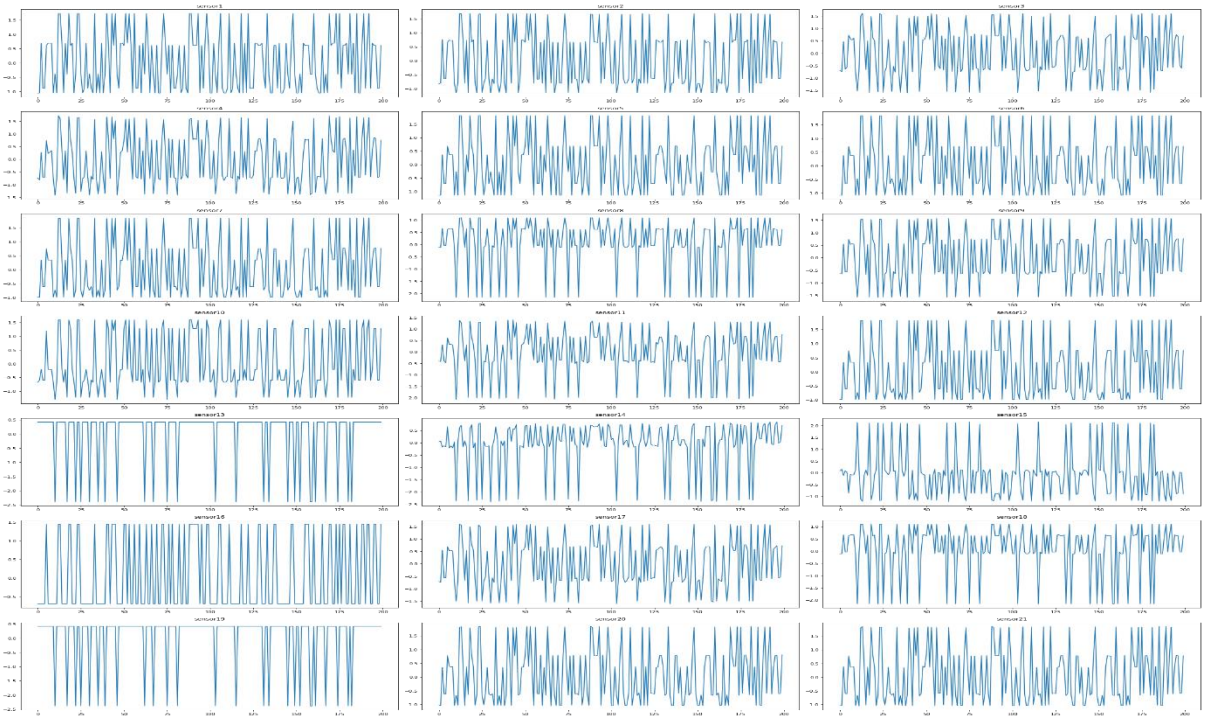

Figure 5: plot data for index for padded train data

Convert time series data to Images:

To apply CNN time series images is converted to images using recurrence plot because image contain more information regardless of their location so it is wise choice to convert time series data into images to carry out effective prediction using CNN(marco 2019)

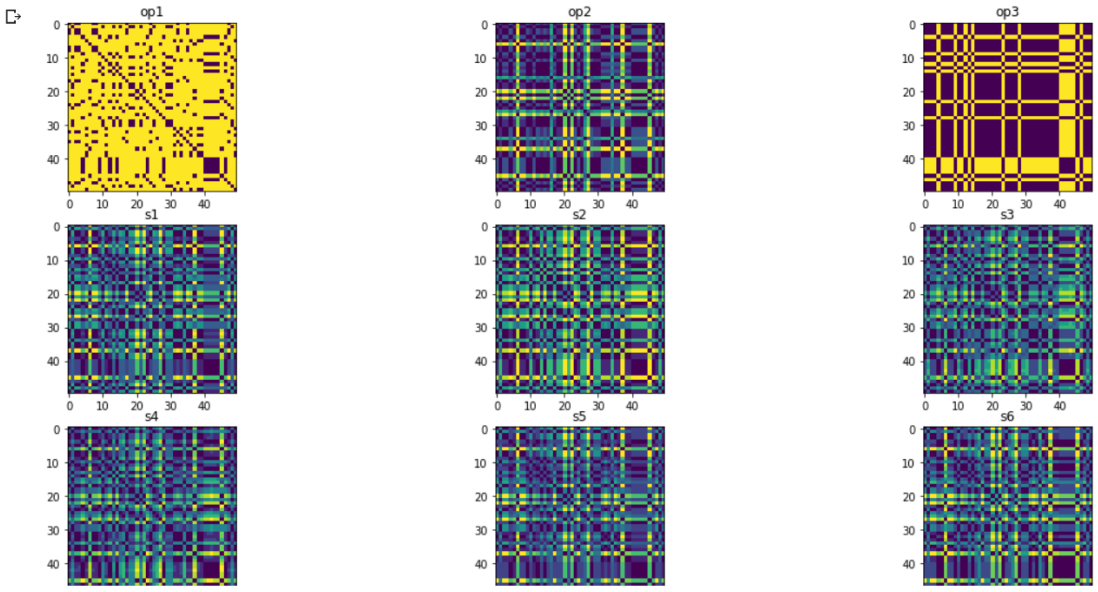

Figure 6 : time series to images

4. Implementation codes for machine learning models:

4.1 **Convolution neural network:**

CNN is applied , the model is build with flatten and dense with activation “relu”

```
[ ]  model = Sequential()

     model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(50, 50, 24)))
     model.add(Conv2D(32, (3, 3), activation='relu'))
     model.add(MaxPooling2D(pool_size=(2, 2)))
     model.add(Dropout(0.25))

     model.add(Conv2D(64, (3, 3), activation='relu'))
     model.add(Conv2D(64, (3, 3), activation='relu'))
     model.add(MaxPooling2D(pool_size=(2, 2)))
     model.add(Dropout(0.25))

     model.add(Flatten())
     model.add(Dense(256, activation='relu'))
     model.add(Dropout(0.5))
     model.add(Dense(3, activation='softmax'))

     #sgd = SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True)
     model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

     print(model.summary())
```

Figure 7: CNN model build

Validation accuracy is 67 percentage and testing accuracy is nearly 90 percentage , in code early stopping is also introduced to get better accuracy from the dataset , in the code both converted image and train data is used to train and evaluate the model.

```
[ ]  tf.random.set_seed(33)
     os.environ['PYTHONHASHSEED'] = str(33)
     np.random.seed(33)
     random.seed(33)

     session_conf = tf.compat.v1.ConfigProto(
         intra_op_parallelism_threads=1,
         inter_op_parallelism_threads=1
     )
     sess = tf.compat.v1.Session(
         graph=tf.compat.v1.get_default_graph(),
         config=session_conf
     )
     tf.compat.v1.keras.backend.set_session(sess)


     es = EarlyStopping(monitor='val_accuracy', mode='auto', restore_best_weights=True, verbose=1, patience=6)

     model.fit(x_train_img, y_train, batch_size=512, epochs=25, callbacks=[es],validation_split=0.2, verbose=2)
```

```
Epoch 1/25
13/13 - 2s - loss: 1.8426 - accuracy: 0.6497 - val_loss: 0.8457 - val_accuracy: 0.6792
Epoch 2/25
13/13 - 1s - loss: 0.7161 - accuracy: 0.7514 - val_loss: 0.7910 - val_accuracy: 0.6792
Epoch 3/25
13/13 - 1s - loss: 0.6937 - accuracy: 0.7514 - val_loss: 0.7772 - val_accuracy: 0.6792
Epoch 4/25
13/13 - 1s - loss: 0.6704 - accuracy: 0.7514 - val_loss: 0.7560 - val_accuracy: 0.6792
Epoch 5/25
13/13 - 1s - loss: 0.6631 - accuracy: 0.7514 - val_loss: 0.7410 - val_accuracy: 0.6792
```

4.2 Random forest:

Random forest with random value is used to predict the remaining useful cycle for the data. Random state is 1234 with the combination of decision tree to get ensemble learning for time series prediction.(zhao 2019)

```
import numpy as np
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1234)
# choose the model
from sklearn.ensemble import RandomForestRegressor
rf = ensemble.RandomForestRegressor()
# set up 5-fold cross-validation
from sklearn import model_selection
cv = model_selection.KFold(5)
# pipeline standardization and model
from sklearn.pipeline import Pipeline
pipeline = Pipeline(steps=[('standardize', preprocessing.StandardScaler())
                           , ('model', rf) ])
# tune the model
my_min_samples_leaf = [2, 10, 25, 50, 100]
my_max_depth = [7, 8, 9, 10, 11, 12]
# run the model using gridsearch, select the model with best search
from sklearn.model_selection import GridSearchCV
optimized_rf = GridSearchCV(estimator=pipeline
                            , cv=cv
```

Figure 8 : Random forest for RUL prediction

Mean square error for random forest is 1782

4.3 LSTM (RNN):

Long short term memory is applied to time series dataset to predict remaining useful cycle for aircraft engine , the model is created using keras importing python library the function of creating LSTM model is follow.

In [38]:
```
from keras import regularizers
from keras.models import Sequential
from keras.layers import LSTM
from keras.layers import Dense
from keras.layers import Activation
from keras.layers import Dropout
from keras.layers import Flatten

# from keras import backend as K
# K.set_session(K.tf.Session(config=K.tf.ConfigProto(intra_op_parallelism_threads=36, inter_op_parallelism_threads=36)))


def createModel(l1Nodes, l2Nodes, d1Nodes, d2Nodes, inputShape):
    # input layer
    lstm1 = LSTM(l1Nodes, input_shape=inputShape, return_sequences=True, kernel_regularizer=regularizers.l2(0.1))
    do1 = Dropout(0.2)

    lstm2 = LSTM(l2Nodes, return_sequences=True, kernel_regularizer=regularizers.l2(0.1))
    do2 = Dropout(0.2)

    flatten = Flatten()

    dense1 = Dense(d1Nodes, activation='relu', kernel_regularizer=regularizers.l2(0.1))
    do3 = Dropout(0.2)

    dense2 = Dense(d2Nodes, activation='relu', kernel_regularizer=regularizers.l2(0.1))
    do4 = Dropout(0.2)

    # output layer
    outL = Dense(1, activation='relu', kernel_regularizer=regularizers.l2(0.1))
    # combine the layers
#     layers = [lstm1, do1, lstm2, do2, dense1, do3, dense2, do4, outL]
    layers = [lstm1, lstm2, do2, flatten,  dense1, dense2, outL]
    # create the model
    model = Sequential(layers)
    model.compile(optimizer='adam', loss='mse')
    return model
```

In [39]:
```
model = createModel(64, 64, 64, 8, (maxlen, numOfSensors))
```

Figure 9: LSTM model

Mean square error for RNN is 789 after three fold as shown in figure 10

```
In [42]: msescores

Out[42]: [{'path': 'best_model_set4fold1.h5', 'mse': 1091.4838755355156},
          {'path': 'best_model_set4fold2.h5', 'mse': 634.5424610845727},
          {'path': 'best_model_set4fold3.h5', 'mse': 789.9235032983678}]

In [43]: for md in msescores:
             saved_model = load_model(md['path'])
             print(saved_model.evaluate(X_test, y_test))

         248/248 [==============================] - 3s 11ms/step
         2168.0150343371974
         248/248 [==============================] - 3s 11ms/step
         1140.2524847215223
         248/248 [==============================] - 3s 12ms/step
         1480.1892798639112
```

Figure 10 :MSE of LSTM(RNN)

The plot of actual value and predicted value is shown below to compare what model has predicted the value of RUL and what the actual value of RUL was.

```
In [45]: plt.figure(figsize=(50, 10))
         plt.plot(range(len(predicted)), predicted, '-x', label='predicted')
         plt.plot(range(len(y_test)), y_test, '-o', label='actual')
         plt.legend()

Out[45]: <matplotlib.legend.Legend at 0x2a5fadb5358>
```

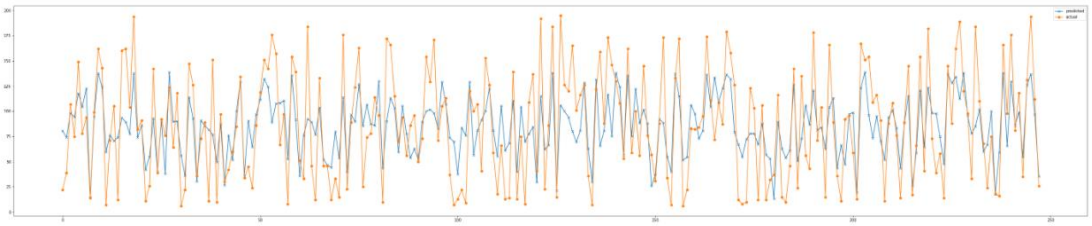

Figure 11: plot of actual vs predicted in LSTM modelling

4.4 Hybrid model (CNN +LSTM):

In the hybrid modelling the combination of CNN and LSTM is applied to predict the reaming useful cycle for the NASA turbo engine dataset, the hybrid model is useful to improve accuracy specially in case of time series.(wang 2018)
The function of hybrid model is below

```
In [53]: from keras.models import Sequential
from keras.layers import Convolution1D
from keras.layers import MaxPooling1D
from keras.layers import Flatten
from keras.layers import LSTM
from keras.layers import Dense
from keras.layers import Activation
from keras.layers import Dropout

# from keras import backend as K
# K.set_session(K.tf.Session(config=K.tf.ConfigProto(intra_op_parallelism_threads=36, inter_op_parallelism_threads=36)))


def createCNNLSTMModel(inputShape):
    cv1 = Convolution1D(input_shape=inputShape, filters=18, kernel_size=2, strides=1, padding='same', activation='relu', name='c
v1')
    mp1 = MaxPooling1D(pool_size=2, strides=2, padding='same', name = 'mp1')

    cv2 = Convolution1D(filters=36, kernel_size=2, strides=1, padding='same', activation='relu', name='cv2')
    mp2 = MaxPooling1D(pool_size=2, strides=2, padding='same', name= 'mp2')

    cv3 = Convolution1D(filters=72, kernel_size=2, strides=1, padding='same', activation='relu', name='cv3')
    mp3 = MaxPooling1D(pool_size=2, strides=2, padding='same', name= 'mp3')

    d4 = Dense(inputShape[0]*inputShape[1], activation='relu')
    do4 = Dropout(0.2)

    lstm5 = LSTM(inputShape[1]*3, return_sequences=True)
    do5 = Dropout(0.2)

    lstm6 = LSTM(inputShape[1]*3)
    do6 = Dropout(0.2)

    d7 = Dense(50, activation='relu')
    do7 = Dropout(0.2)

    dout = Dense(1)

    model = Sequential([cv1, mp1, cv2, mp2, cv3, mp3, d4, do4, lstm5, do5, lstm6, do6, d7, do7, dout])
    model.compile(optimizer='rmsprop', loss='mse')
    return model
```

Figure 12: HYBRID model CNN + LSTM

The mean square error in hybrid model is 1153 as shown in below diagram after 3 fold

```
In [55]: msescores

Out[55]: [{'path': 'best_model_set4fold1.h5', 'mse': 1155.0911572789407},
          {'path': 'best_model_set4fold2.h5', 'mse': 1155.764778411328},
          {'path': 'best_model_set4fold3.h5', 'mse': 1153.2789476611947}]

In [56]: for md in msescores:
             saved_model = load_model(md['path'])
             print(saved_model.evaluate(X_test, y_test))

248/248 [==============================] - 1s 3ms/step
2307.540314705141
248/248 [==============================] - 1s 3ms/step
2273.5552466607865
248/248 [==============================] - 1s 4ms/step
```

The output layer heat map is plotted to get outputs visualization , first the viz model is developed using keras.model library

```
In [65]: import seaborn as sns
         def plotLayerHeatmap(layer_data, mcIndex):
             mcData = layer_data[mcIndex]
             plt.figure(figsize=(30, 10))
             sns.heatmap(mcData.transpose())

In [66]: plotLayerHeatmap(layer_outputs[1], 100)
```

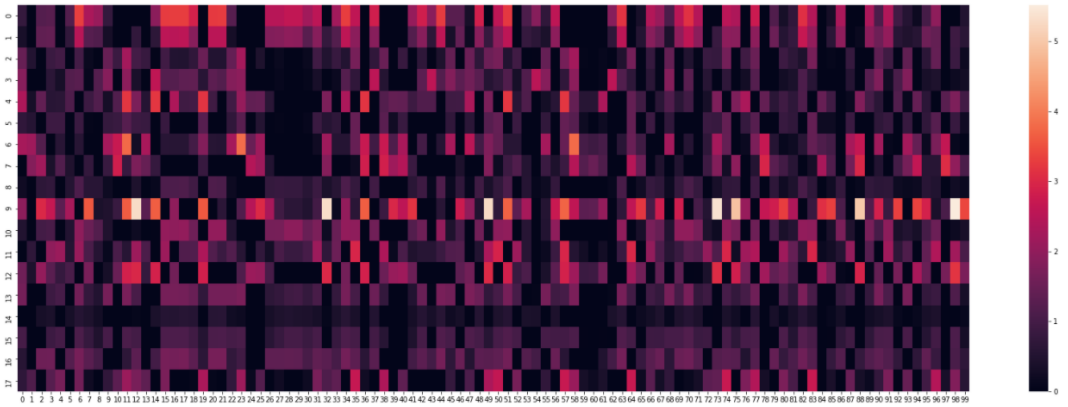

Figure 13 :Heat map for output for hybrid model

4.5 Transfer learning:

Transfer learning model is applied on time series dataset to solve or predict the remaining useful cycle of engine. In this approach already LSTM based transfer learning model is which is trained is applied to test data to check mean square error is better than all previous models.

Before applying the feature is taken into one array and label data is taken into different array

```
    def get_feature_slice(self, input_data):
        # Reshape the data to (samples, time steps, features)
        def reshapeFeatures(input, columns, sequence_length):
            data = input[columns].values
            num_elements = data.shape[0]
            #print(num_elements)
            for start, stop in zip(range(0, num_elements-sequence_length), range
(sequence_length, num_elements)):
                yield(data[start:stop, :])

        feature_list = [list(reshapeFeatures(input_data[input_data['id'] == i],
feature_columns, self.sequence_length))
                        for i in range(1, self.engine_size + 1) if len(input_dat
a[input_data['id']  == i]) > self.sequence_length]
        ##for i in range(len(feature_list)):
        ##    print(np.array(feature_list[i]).shape)
        feature_array = np.concatenate(list(feature_list), axis=0).astype(np.flo
at32)

        length = len(feature_array) // self.batch_size
        return feature_array[:length*self.batch_size]
```

```
    def get_last_data_slice(self, input_data):
        num_engine = input_data['id'].unique().max()
        test_feature_list = [input_data[input_data['id'] == i][feature_columns].
values[-self.sequence_length:]
                             for i in range(1, num_engine+1) if len(input_data[i
nput_data['id']  == i]) >= self.sequence_length]
        test_feature_array = np.asarray(test_feature_list).astype(np.float32)
        length_test = len(test_feature_array) // self.batch_size

        test_label_list = [input_data[input_data['id'] == i]['RUL'].values[-1:]
                           for i in range(1, num_engine+1) if len(input_data[inp
ut_data['id'] == i]) >= self.sequence_length]
        test_label_array = np.asarray(test_label_list).astype(np.float32)
        length_label = len(test_label_array) // self.batch_size

        return test_feature_array[:length_test*self.batch_size], test_label_arra
y[:length_label*self.batch_size]

    #
    def set_test_data_encoding(self, test_data_encoding):
        self.test_data_encoding = test_data_encoding

    def set_train_data_encoding(self, train_data_encoding):
        self.train_data_encoding = train_data_encoding
```

Then transfer learning model is applied (LSTM based transfer learning model is which is trained is applied to test data)

In [8]:
```
model.compile(loss='mse', optimizer='rmsprop', metrics=['mae'])
print(model.summary())
```

```
WARNING:tensorflow:From C:\Users\Anaconda3\lib\site-packages\tensorflow\python
\keras\utils\losses_utils.py:170: to_float (from tensorflow.python.ops.math_op
s) is deprecated and will be removed in a future version.
Instructions for updating:
Use tf.cast instead.
```


```
WARNING:tensorflow:From C:\Users\Anaconda3\lib\site-packages\tensorflow\python
\keras\utils\losses_utils.py:170: to_float (from tensorflow.python.ops.math_op
s) is deprecated and will be removed in a future version.
Instructions for updating:
Use tf.cast instead.
```


```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_0 (LSTM)                (None, 50, 100)           50400
_________________________________________________________________
dropout_0 (Dropout)          (None, 50, 100)           0
_________________________________________________________________
lstm_1 (LSTM)                (None, 50, 50)            30200
_________________________________________________________________
dropout_1 (Dropout)          (None, 50, 50)            0
_________________________________________________________________
lstm_2 (LSTM)                (None, 25)                7600
_________________________________________________________________
dropout_2 (Dropout)          (None, 25)                0
_________________________________________________________________
dense_0 (Dense)              (None, 1)                 26
_________________________________________________________________
activation_0 (Activation)    (None, 1)                 0
=================================================================
Total params: 88,226
Trainable params: 88,226
Non-trainable params: 0
```

The plot of true value versed actual is shown in below diagram: we can see how close both values are

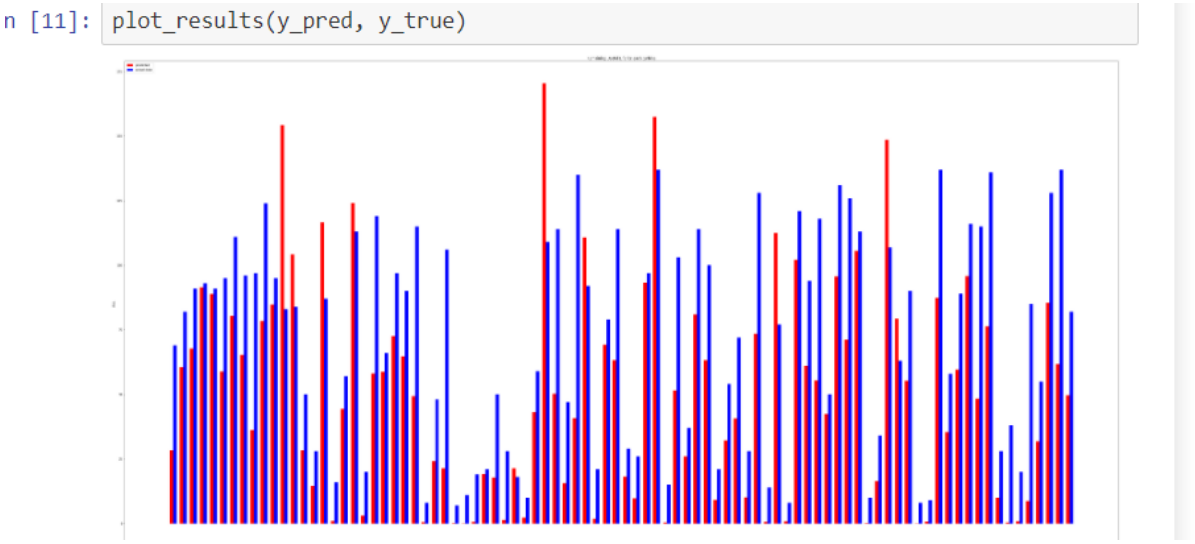

Figure 14: actual value vs predicted for transfer learning

Reference:

D. Wang, K.-L. Tsui and Q. Miao, "Prognostics and health management: A review of vibration based bearing and gear health indicators", *IEEE Access*, vol. 6, pp. 665-676, 2018.

R. Zhao, R. Yan, Z. Chen, K. Mao, P. Wang and R. X. Gao, "Deep learning and its applications to machine health monitoring", *Mech. Syst. Signal Process.*, vol. 115, pp. 213-237, Jan. 2019

Amirfathi, M., Morris, S., O'Rorke, P., Bond, W., Clair, D.S., 1991. Pattern recognition for nondestructive evaluation, in: 1991 IEEE Aerospace Applications Conference Digest. IEEE, pp. 6–1.

Marco 2019 CNN on RUL prediction URL -https://towardsdatascience.com/remaining-life-estimation-with-keras-2334514f9c61