# Configuration Manual

MSc Research Project
Data Analytics


Raul Damian Sainz Calderon
Student ID: 19158696

School of Computing
National College of Ireland


Supervisor: Paul Stynes

**National College of Ireland**
**Project Submission Sheet**
**School of Computing**

| | |
|---|---|
| **Student Name:** | Raul Damian Sainz Calderon |
| **Student ID:** | 19158696 |
| **Programme:** | Data Analytics |
| **Year:** | 2021 |
| **Module:** | MSc Research Project |
| **Supervisor:** | Paul Stynes |
| **Submission Due Date:** | 16/12/2021 |
| **Project Title:** | Configuration Manual |
| **Word Count:** | 782 |
| **Page Count:** | 8 |

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 bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

| | |
|---|---|
| **Signature:** | Raul Sainz |
| **Date:** | 16th December 2021 |

**PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:**

| | |
|---|---|
| Attach a completed copy of this sheet to each project (including multiple copies). | ☐ |
| **Attach a Moodle submission receipt of the online project submission**, to each project (including multiple copies). | ☐ |
| **You must ensure that you retain a HARD COPY of the project**, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer. | ☐ |

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

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

# Configuration Manual

Raul Damian Sainz Calderon
19158696

## 1 About this Manual

The objective for configuration manual is provide details of the system setup, software specification and the required instructions to run and replicate the experiments in Google Colab.

## 2 Resources and Equipment

The following tools and components were used for the implementation of this research, Table. 1" describes required resources, software and services used during the project.

Table 1: Resources, Software and Services

| Category | Item | Description |
|---|---|---|
| Computing | RAM | 8GB (16GB are recommended) |
| | Processor | 64-bit multi-core processor (Intel i5 or superior) |
| | Storage | 250+ GB of available space in hard disk |
| | Operating System | Ubuntu, macOS or Windows |
| Software | Python | Main Programming language. |
| | Anaconda | Distribution of to simplify package management. |
| | Tensorflow & Keras | Library to develop and train models. |
| | Jupyter Notebook | ML and data processing and modeling. |
| | VS code | Programming IDE. |
| Cloud Services | Google Colab | Run Notebooks for Neural Networks. |
| | Github | Code repository and version control. |

## 3 Code Version control Repository- Github

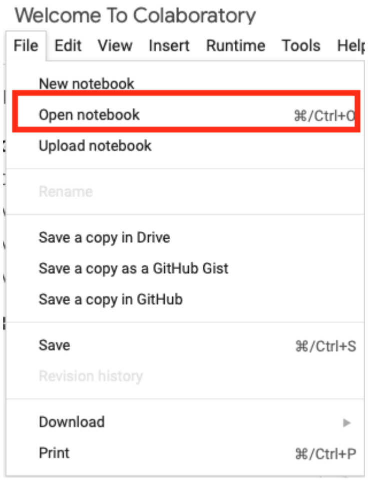
For the pourpuse to have a better control and track of the changes made into the code, a git repository was created, the data files, notebooks and python code used during this research can be found in the following public Github links:

- Repository URL: `https://github.com/raulsainz/MSCDAD_JAN21A_Research`
- Clone URL: `https://github.com/raulsainz/MSCDAD_JAN21A_Research.git`

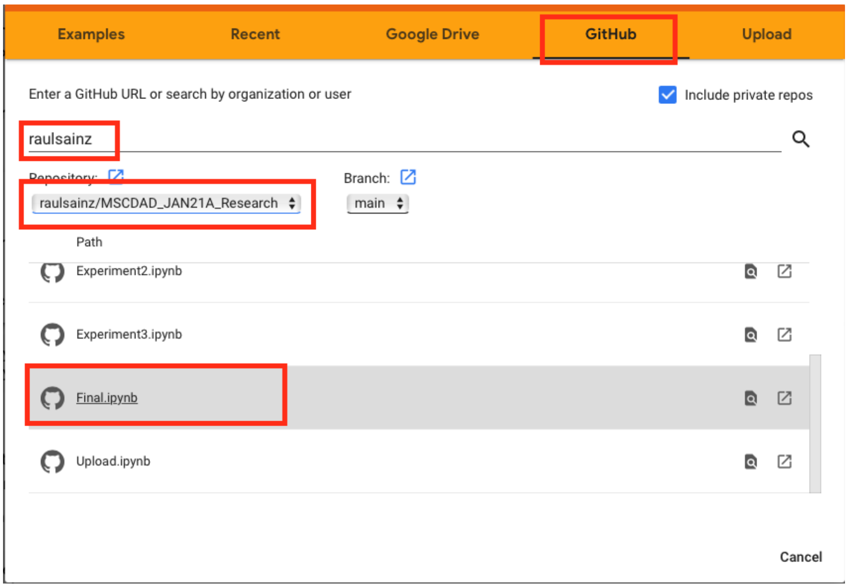
# 4 Using Google Colab

To run the notebook using google Colab follow the next steps:

1. Open a browser and go to google colab `https://colab.research.google.com`
2. Go to File, Open Notebook" menu option as shown in "Fig. 2"




3. Open Notbook Final.ipynb" from Github "Fig. 3"
   (a) Select the Github tab
   (b) Search for raulsainz" user
   (c) Select the repository raulsainz/MSCDAD_JAN21A_Research.
   (d) Click on **Final_FD001.ipynb** file.




4. Make sure to run the first cell to setup the environment "Fig. 4"
5. Run The rest of the cells.

```
try:
    import google.colab
    COLAB = True
    print("Note: using Google CoLab")
    #Clone the repository
    !git clone https://github.com/raulsainz/MSCDAD_JAN21A_Research.git
    # Install package dependencies
    !pip install keras-tcn
    # adding repo folder to the system path
    import sys
    sys.path.insert(0, '/content/MSCDAD_JAN21A_Research/')
except:
    print("Note: Using Local enviroment")
    COLAB = False
```

```
Note: using Google CoLab
Cloning into 'MSCDAD_JAN21A_Research'...
remote: Enumerating objects: 66, done.
remote: Counting objects: 100% (66/66), done.
remote: Compressing objects: 100% (47/47), done.
remote: Total 66 (delta 18), reused 59 (delta 15), pack-reused 0
Unpacking objects: 100% (66/66), done.
Checking out files: 100% (37/37), done.
Collecting keras-tcn
  Downloading keras_tcn-3.4.0-py2.py3-none-any.whl (13 kB)
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-packages (from keras-tcn) (1.19.5)
Requirement already satisfied: tensorflow in /usr/local/lib/python3.7/dist-packages (from keras-tcn) (2.7.0)
Collecting tensorflow-addons
  Downloading tensorflow_addons-0.15.0-cp37-cp37m-manylinux_2_12_x86_64.manylinux2010_x86_64.whl (1.1 MB)
     |████████████████████████████████| 1.1 MB 16.2 MB/s
Requirement already satisfied: six>=1.12.0 in /usr/local/lib/python3.7/dist-packages (from tensorflow->keras-tcn) (1.15.0)
Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.7/dist-packages (from tensorflow->keras-tcn) (1.1.0)
Requirement already satisfied: astunparse>=1.6.0 in /usr/local/lib/python3.7/dist-packages (from tensorflow->keras-tcn) (1.6.3)
Requirement already satisfied: google-pasta>=0.1.1 in /usr/local/lib/python3.7/dist-packages (from tensorflow->keras-tcn) (0.2.0)
Requirement already satisfied: protobuf>=3.9.2 in /usr/local/lib/python3.7/dist-packages (from tensorflow->keras-tcn) (3.17.3)
```

# 5 Repository File Structure

The project files are divided into 2 folders: root and datasets

## 5.1 Root

The Root of the folder contains the jupiter notebooks used during the research, but the one containing the final code is named **Final.ipynb"**. Also whithin the root folder the file **Models.py** contains the functions used to construct the models architecture along with other the functions to run, train and evaluate them.

## 5.2 Datasets

Contains the files downloaded from the Turbofan Engine Degradation Simulation Data Set A. Saxena , K. Goebel (2008):

- train_FD001.txt, train_FD002.txt, train_FD003.txt, train_FD004.txt
- test_FD001.txt, test_FD002.txt, test_FD003.txt, test_FD004.txt
- RUL_FD001.txt, RUL_FD002.txt, RUL_FD003.txt, RUL_FD004.txt, readme.txt

# 6 Enviroment Setup

The flowing code snippet is placed at the beginning of the notebook to detect the environment (local or Google Colab). If the environment is Google Colab, it will clone the repository, install the package dependencies and add the repository folder to the system path to be able to run the notebook.

```
try:
    import google.colab
    COLAB = True
    print("Note: using Google CoLab")
    #Clone the repository
```

```
    !git clone https://github.com/raulsainz/MSCDAD_JAN21A_Research.git
    # Install package dependencies
    !pip install keras-tcn
    # adding repo folder to the system path
    import sys
    sys.path.insert(0, '/content/MSCDAD_JAN21A_Research/')
except:
    print("Note: Using Local enviroment")
    COLAB = False
```

# 7 Generating Train and Test Datasets

To create the data to be feed into the neural networks LSTM and TCN, we need to format the data into sequences, we do this by generating sliding window sequences, with this script “Fig. 1” we generate X_train and X_test

```
# Generate sliding window sequences for training dataset
seq_gen = (list(gen_sequence(df_train[df_train['machine_id']==id], window_size, sequence_cols))
               for id in df_train['machine_id'].unique())
# generate sequences and convert to numpy array
seq_array = np.concatenate(list(seq_gen)).astype(np.float32)
print(seq_array.shape)
```

```
(48799, 50, 16)
```

```
x_train, x_test = [], []
for machine_id in df_train.machine_id.unique():
    for sequence in gen_sequence(df_train[df_train.machine_id==machine_id], window_size, sequence_cols):
        x_train.append(sequence)
    for sequence in gen_sequence(df_test[df_test.machine_id==machine_id], window_size, sequence_cols):
        x_test.append(sequence)
x_train = np.asarray(x_train)
x_test = np.asarray(x_test)

print("X_Train shape:", x_train.shape)
print("X_Test shape:", x_test.shape)
```

```
X_Train shape: (48799, 50, 16)
X_Test shape: (29188, 50, 16)
```

Figure 1: Snippet to generate sliding window sequences

For the CNN we need to generate the recurrence plots wit the code show in “Fig. 2”

**Generate recurrence plots for CNN**

```
# Create new array with recurrence plot training
x_train_img = np.apply_along_axis(gen_rec_plot, 1, x_train).astype('float16')
print(x_train_img.shape)
# Create new array with recurrence plot testing
x_test_img = np.apply_along_axis(gen_rec_plot, 1, x_test).astype('float16')
print(x_test_img.shape)
```

```
(48799, 50, 50, 16)
(29188, 50, 50, 16)
```

Figure 2: Snippet to generate recurrence plots for CNN

# 8 Custom Functions

All the models architecture and other functions are included in the file **models.py**. This script contains and loads most of the python packages required to run the experiments. The following custom functions are included:

- **lstm_classification:** This function creates the architecture for LSTM model “Fig. 3”

```python
def lstm_classification(seq_array, label_array, sequence_length):
    # The first layer is an LSTM layer with 100 units followed by another LSTM layer with 50 units.
    # Dropout is also applied after each LSTM layer to control overfitting.
    # Final layer is a Dense output layer with single unit and linear activation since this is a regression problem.
    nb_features = seq_array.shape[2]
    nb_out = label_array.shape[1]

    model = Sequential()
    model.add(LSTM(
             input_shape=(sequence_length, nb_features),
             units=100,
             return_sequences=True))
    model.add(Dropout(0.2))
    model.add(LSTM(
              units=50,
              return_sequences=False))
    model.add(Dropout(0.2))
    model.add(Dense(2, activation='softmax'))
    model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
    return model
```

Figure 3: LSTM Model architecture

- **cnn_classification:** This function creates the architecture for CNN model “Fig. 4”

```python
def cnn_classification(seq_array,label_array):
        model = Sequential()

        model.add(Conv2D(32, (3, 3), activation='relu',
                   input_shape=(seq_array.shape[1], seq_array.shape[2], seq_array.shape[3])))
        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(label_array.shape[1], activation='softmax'))

        model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

        return model
```

Figure 4: CNN Model architecture

- **tcn_classification:** This function creates the architecture for TCN model “Fig. 5”

- **run_model:** trains and test the provided model with the provided data and labels, and calculates the evaluation metrics and plots “Fig. 6”.

```python
from tcn import TCN, tcn_full_summary
from tensorflow.keras.layers import Dense
from tensorflow.keras.models import Sequential
def tcn_classification(seq_array,label_array,sequence_length):
        batch_size, time_steps, input_dim = None, sequence_length, 1
        tcn_layer = TCN(input_shape=(seq_array.shape[1], seq_array.shape[2]))
        # The receptive field tells you how far the model can see in terms of timesteps.
        print('Receptive field size =', tcn_layer.receptive_field)

        model = Sequential([
        tcn_layer,
        Dense(label_array.shape[1], activation='softmax')
        ])

        model.compile(optimizer='adam', loss='categorical_crossentropy',metrics=['accuracy'])
        return model
```

Figure 5: TCN Model architecture

```python
def run_model(model, X_train, y_train, X_test, y_test, verbose=True, desc = 'No Name',labels = ['True','False'],class_weight=None,
        """
        This function trains and test the provided model with the given datasets and labes, and calculates the evaluation metrics
        Returns: Results Dictionary
        """
        # Defines random seed
        seed = 7
        np.random.seed(seed)
        # Checks if weights were passed to the function
        if class_weight is None:
                # Train with no weights
                history = model.fit(X_train, y_train,
                epochs=epochs, batch_size=batch_size,
                validation_split=0.1, verbose=1,
                callbacks=[EarlyStopping])
        else:
                # Train with weights
                print("training with weights")
                history = model.fit(X_train, y_train,
                epochs=epochs, batch_size=batch_size, validation_split=0.1,
                verbose=1,callbacks=[EarlyStopping],class_weight=class_weight)

        # calculate accuracy
        training_loss, training_acc = model.evaluate(X_test, y_test, verbose=0)
        # Obtain the predictions
        y_pred = model.predict(X_test)
        y_test = np.argmax(y_test, axis=1, out=None)
        # convert categorical probability to binary label
        y_pred = np.argmax(y_pred,axis=1)

        # Calculate Overall Acuracy
        model_acc = metrics.accuracy_score(y_test, y_pred )
        # calculaten ROC Curve
        fpr , tpr , thresholds = roc_curve ( y_test , y_pred)
        class_report = classification_report(y_test,y_pred,digits=2,output_dict=True)
        roc_auc = roc_auc_score(y_test, y_pred)
        model_kappa = cohen_kappa_score(y_test, y_pred )
        matrix = confusion_matrix(y_true=y_test, y_pred=y_pred)
```

Figure 6: Custom Function to tran and test the model performance.

# 9 Performance Plots

After running the models through the custom functon the results include: Train Accuracy and Loss, Accuracy, ROC_AUC, Precision, Recall and F1 Score. The three resulting plots returned by the function are Training vs Validation, ROC curve plot and confusion matrix as shown in "Fig. 7".

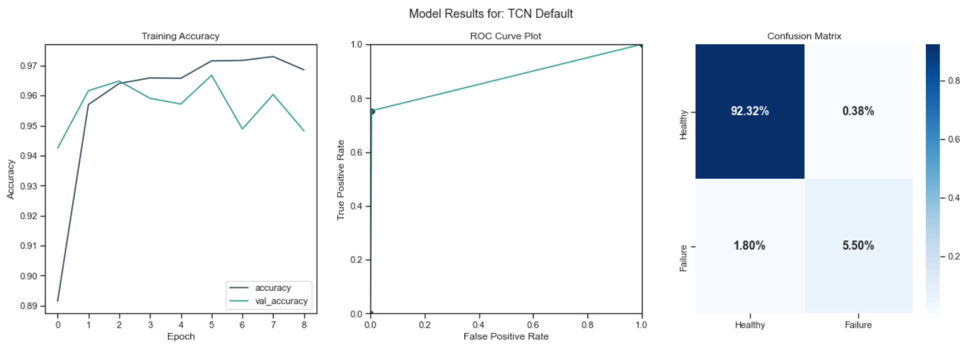



Figure 7: Three resulting plots to evaluate the performance of the model.

# 10 Comparison Plots

Because multiple models and experiments are implemented whithin the notebook, each time we call the *run_model* function the resulting dictionary is stored in a variable called *model_results*, at the end of the experiments we use the function *printClassificationResults* to print a heatmap of the results with all the performance values and the ROC_AUC data of each model to compare between them, the resulting plots are shown in "Fig. 8". The results are ordered by accuracy but we can use the other parameters to evaluate their performance in depth.

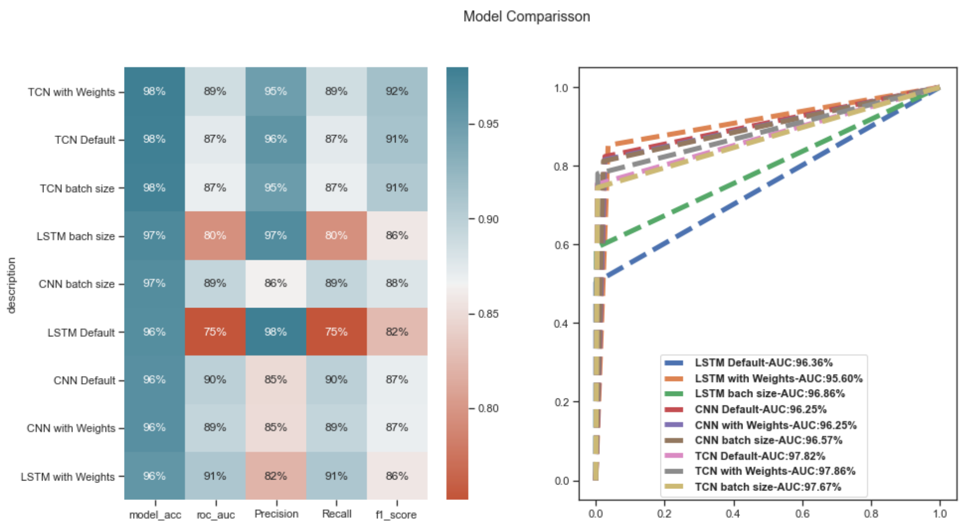



Figure 8: Heatmap and ROC curve plot for comparing the performance of all the models

# References

A. Saxena , K. Goebel (2008). Turbofan engine degradation simulation data set, http://ti.arc.nasa.gov/project/prognostic-data-repository. Accessed: 2018-12-06.