National College of Ireland

Configuration Manual

MSc Research Project
Data Analytics

Sachin Nikam
Student ID: x19198159

School of Computing
National College of Ireland

Supervisor: Dr. Rashmi Gupta

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Sachin Nikam
Student ID:	x19198159
Programme:	Data Analytics
Year:	2021
Module:	MSc Research Project
Supervisor:	Dr. Rashmi Gupta
Submission Due Date:	16/08/2021
Project Title:	Configuration Manual
Word Count:	652
Page Count:	7

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:	Sachin Nikam
Date:	10th October 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

Sachin Nikam
x19198159

1 Introduction

Configuration manual aims to provide the system requirements like hardware, software, programming language and workflow to run the code.

2 System Configuration

2.1 Hardware

- OS: Windows 10
- RAM: 16 GB
- Processor: intel core i5 8th gen
- Hard Disk: 1 TB

2.2 Software

- Anaconda: Python Jupyter Notebook
- Google Colab

3 Libraries required to be installed

The project is implemented locally on python using Jupyter notebook as IDE. Network was train for 30 Epochs. Each Epoch could take 5 to 6 hours. To run smoothly and increase the run time Google Colab Pro is recommended. Training time to train the network can be reduced. It is important to import the following libraries in python for smooth execution.

- pickle
- numpy
- glob
- keras
- music21
- pandas
- os
- matplotlib

4 Instructions to run the code

- Unzip all the files and place in directory Artefact.zip
- Figure.1 shows the directory after unzipping the files.

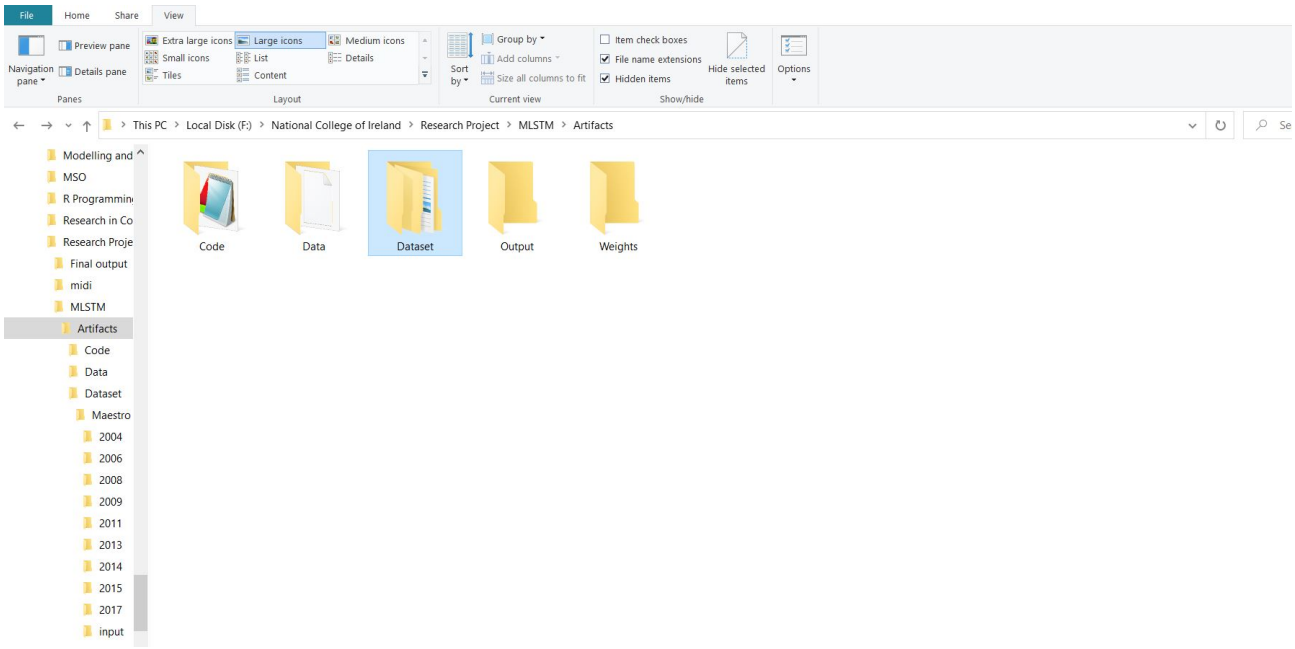

Figure 1: Directories where files are stored

- In the Artefact directory there is "Dataset" directory contains 3 sub-directories. Maestro folder contains all the files of Maestro dataset, Midi folder contains dataset from Feel my sound website and Sample folder contains the sample on which the model is implemented. This includes 20 of 2017 folder from Maestro dataset.
- In the Code directory, there is mlstm.ipynb file. Open that code file.
- At first all the important libraries are imported.Figure.2 shows the imported libraries

```
In [1]: import glob  # return file paths with specific file format or pattern
        import pickle
        import numpy
        from music21 import converter, instrument, note, chord, stream  # package to read and execute midi files
        from keras.models import Sequential
        from keras.layers import Dense
        from keras.layers import Dropout
        from keras.layers import LSTM
        from keras.layers import Activation
        from keras.layers import BatchNormalization as BatchNorm
        from keras.utils import np_utils
        from keras.callbacks import ModelCheckpoint
        from keras.optimizers import SGD # Momentum cell to be added in the model
        # Libraries for Visualization
        import numpy as np # linear algebra
        import pandas as pd # data processing
        import os
        import matplotlib.pyplot as plt
        import matplotlib.lines as mlines
```

Figure 2: importing libraries

- Figure.3 shows the path of the dataset which need to be selected. In glob object file path for the dataset need to be selected.

```
In [14]: notes = [] # Dataframe to store the notes for the Midi files
for file in glob.glob("F:/National College of Ireland/Research in Computing/Music/Maestro Dataset/maestro-v1.0.0/input/*.midi"):
    # Generate music21 objects
    Music21Obj = converter.parse(file)
    print("Parsing %s" % file)
    notes_to_parse = None
    try:
        #  subclassing, though instrument
        subclass = instrument.partitionByInstrument(Music21Obj)
        notes_to_parse = subclass.parts[0].recurse()
    except:
        notes_to_parse = Music21Obj.flat.notes
    # Extract Pitch / Chord info
    for element in notes_to_parse:
        if isinstance(element, note.Note):
            notes.append(str(element.pitch))
        elif isinstance(element, chord.Chord):
            notes.append('.'.join(str(n) for n in element.normalOrder))
```

```
Parsing F:/National College of Ireland/Research in Computing/Music/Maestro Dataset/maestro-v1.0.0/input\MIDI-Unprocessed_041_PI
ANO041_MID--AUDIO-split_07-06-17_Piano-e_1-01_wav--1.midi
Parsing F:/National College of Ireland/Research in Computing/Music/Maestro Dataset/maestro-v1.0.0/input\MIDI-Unprocessed_041_PI
ANO041_MID--AUDIO-split_07-06-17_Piano-e_1-01_wav--2.midi
Parsing F:/National College of Ireland/Research in Computing/Music/Maestro Dataset/maestro-v1.0.0/input\MIDI-Unprocessed_041_PI
ANO041_MID--AUDIO-split_07-06-17_Piano-e_1-01_wav--3.midi
Parsing F:/National College of Ireland/Research in Computing/Music/Maestro Dataset/maestro-v1.0.0/input\MIDI-Unprocessed_041_PI
ANO041_MID--AUDIO-split_07-06-17_Piano-e_1-01_wav--4.midi
Parsing F:/National College of Ireland/Research in Computing/Music/Maestro Dataset/maestro-v1.0.0/input\MIDI-Unprocessed_042_PI
ANO042_MID--AUDIO-split_07-06-17_Piano-e_1-02_wav--1.midi
Parsing F:/National College of Ireland/Research in Computing/Music/Maestro Dataset/maestro-v1.0.0/input\MIDI-Unprocessed_042_PI
ANO042_MID--AUDIO-split_07-06-17_Piano-e_1-02_wav--2.midi
Parsing F:/National College of Ireland/Research in Computing/Music/Maestro Dataset/maestro-v1.0.0/input\MIDI-Unprocessed_042_PI
ANO042_MID--AUDIO-split_07-06-17_Piano-e_1-02_wav--3.midi
Parsing F:/National College of Ireland/Research in Computing/Music/Maestro Dataset/maestro-v1.0.0/input\MIDI-Unprocessed_043_PI
ANO043_MID--AUDIO-split_07-06-17_Piano-e_1-03_wav--1.midi
Parsing F:/National College of Ireland/Research in Computing/Music/Maestro Dataset/maestro-v1.0.0/input\MIDI-Unprocessed_043_PI
ANO043_MID--AUDIO-split_07-06-17_Piano-e_1-03_wav--2.midi
Parsing F:/National College of Ireland/Research in Computing/Music/Maestro Dataset/maestro-v1.0.0/input\MIDI-Unprocessed_043_PI
ANO043_MID--AUDIO-split_07-06-17_Piano-e_1-03_wav--3.midi
Parsing F:/National College of Ireland/Research in Computing/Music/Maestro Dataset/maestro-v1.0.0/input\MIDI-Unprocessed_043_PI
ANO043_MID--AUDIO-split_07-06-17_Piano-e_1-03_wav--4.midi
Parsing F:/National College of Ireland/Research in Computing/Music/Maestro Dataset/maestro-v1.0.0/input\MIDI-Unprocessed_044_PI
ANO044_MID--AUDIO-split_07-06-17_Piano-e_1-04_wav--1.midi
Parsing F:/National College of Ireland/Research in Computing/Music/Maestro Dataset/maestro-v1.0.0/input\MIDI-Unprocessed_044_PI
ANO044_MID--AUDIO-split_07-06-17_Piano-e_1-04_wav--2.midi
Parsing F:/National College of Ireland/Research in Computing/Music/Maestro Dataset/maestro-v1.0.0/input\MIDI-Unprocessed_044_PI
ANO044_MID--AUDIO-split_07-06-17_Piano-e_1-04_wav--3.midi
Parsing F:/National College of Ireland/Research in Computing/Music/Maestro Dataset/maestro-v1.0.0/input\MIDI-Unprocessed_044_PI
ANO044_MID--AUDIO-split_07-06-17_Piano-e_1-04_wav--4.midi
Parsing F:/National College of Ireland/Research in Computing/Music/Maestro Dataset/maestro-v1.0.0/input\MIDI-Unprocessed_045_PI
ANO045_MID--AUDIO-split_07-06-17_Piano-e_2-01_wav--1.midi
Parsing F:/National College of Ireland/Research in Computing/Music/Maestro Dataset/maestro-v1.0.0/input\MIDI-Unprocessed_045_PI
ANO045_MID--AUDIO-split_07-06-17_Piano-e_2-01_wav--2.midi
Parsing F:/National College of Ireland/Research in Computing/Music/Maestro Dataset/maestro-v1.0.0/input\MIDI-Unprocessed_045_PI
ANO045_MID--AUDIO-split_07-06-17_Piano-e_2-01_wav--4.midi
```

Figure 3: Selecting File path and Parsing Midi files

- In the Data folder "note" file is present. note file holds the signatures of the music from pickle of stored Midi files, which will be used with the weights generated by the model. Figure.4 shows the file path of notes and flow of the code.

In [15]:
```
with open('F:/National College of Ireland/Research Project/MLSTM/data/notes', 'wb') as filepath:
    pickle.dump(notes, filepath)
```

In [16]:
```
n_vocab = len(set(notes))
sequence_length = 100
# Pitch info
pitchnames = sorted(set(item for item in notes))
note_to_int = dict((note, number) for number, note in enumerate(pitchnames))

Train_X = []
Train_y = []
```

In [17]:
```
for i in range(0, len(notes) - sequence_length, 1):
    sequence_in = notes[i:i + sequence_length]
    sequence_out = notes[i + sequence_length]
    Train_X.append([note_to_int[char] for char in sequence_in])
    Train_y.append(note_to_int[sequence_out])

n_patterns = len(Train_X)

Train_X = numpy.reshape(Train_X, (n_patterns, sequence_length, 1))
Train_X = Train_X / float(n_vocab)

Train_y = np_utils.to_categorical(Train_y)
```

Figure 4: File path to open notes from Data directory

- Figure.5 shows the model design which includes two LSTM layers by adding Momentum layer in to it.

In [18]:
```
# LSTM Model
model = Sequential()
model.add(LSTM(
    512,
    input_shape=(Train_X.shape[1], Train_X.shape[2]),
    recurrent_dropout=0.3,
    return_sequences=True
))
```

In [19]:
```
# Recurrent Layer
model.add(LSTM(512, return_sequences=True, recurrent_dropout=0.3,))
model.add(LSTM(512))
model.add(BatchNorm())
model.add(Dropout(0.3))
model.add(Dense(256))
model.add(Activation('relu'))
model.add(BatchNorm())
model.add(Dropout(0.3))
model.add(Dense(n_vocab))
model.add(Activation('softmax'))
epochs = 50
learning_rate = 0.1
decay_rate = learning_rate / epochs
momentum = 0.8
sgd = SGD(learning_rate=learning_rate, momentum=momentum, decay=decay_rate, nesterov=False)
model.compile(loss='binary_crossentropy', optimizer=sgd, metrics=['accuracy'])
print(model.summary())
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_3 (LSTM)                (None, 100, 512)          1052672
_________________________________________________________________
lstm_4 (LSTM)                (None, 100, 512)          2099200
_________________________________________________________________
lstm_5 (LSTM)                (None, 512)               2099200
_________________________________________________________________
batch_normalization_2 (Batch (None, 512)               2048
_________________________________________________________________
dropout_2 (Dropout)          (None, 512)               0
_________________________________________________________________
```

Figure 5: Model design

- In Figure.6 file path is given where the weights are stored. Separate folder is created in Artifacts where the weights can be stored. In the Artifacts already generated weights are stored which can be used and model can be designed with "notes" file. Model is trained in this step. Epochs can be changed depending on the machine performance, 30 Epochs are given in this code for smooth execution which took 3 days to run the code.

```
In [21]: # Training
filepath = "weights-improvement-{epoch:02d}-{loss:.4f}-bigger.hdf5"
checkpoint = ModelCheckpoint(
    filepath,
    monitor='loss',
    verbose=0,
    save_best_only=True,
    mode='min'
)
callbacks_list = [checkpoint]

model.fit(Train_X, Train_y, epochs=30, batch_size=1000, callbacks=callbacks_list)
scores = model.evaluate(Train_X, Train_y, verbose=0)
print("Accuracy: %.2f%%" % (scores[1]*100))
```

```
Epoch 1/30
56/56 [==============================] - 2129s 38s/step - loss: 0.6807 - accuracy: 0.0019
Epoch 2/30
56/56 [==============================] - 2466s 44s/step - loss: 0.6453 - accuracy: 0.0016
Epoch 3/30
56/56 [==============================] - 2024s 36s/step - loss: 0.5933 - accuracy: 0.0020
Epoch 4/30
56/56 [==============================] - 2399s 43s/step - loss: 0.5101 - accuracy: 0.0019
Epoch 5/30
56/56 [==============================] - 2021s 36s/step - loss: 0.4004 - accuracy: 0.0025
Epoch 6/30
56/56 [==============================] - 2148s 38s/step - loss: 0.2904 - accuracy: 0.0029
Epoch 7/30
56/56 [==============================] - 1985s 35s/step - loss: 0.2055 - accuracy: 0.0032
Epoch 8/30
56/56 [==============================] - 1987s 36s/step - loss: 0.1492 - accuracy: 0.0034
Epoch 9/30
56/56 [==============================] - 1880s 33s/step - loss: 0.1135 - accuracy: 0.0038
Epoch 10/30
56/56 [==============================] - 1930s 35s/step - loss: 0.0904 - accuracy: 0.0052
Epoch 11/30
56/56 [==============================] - 2118s 38s/step - loss: 0.0751 - accuracy: 0.0049
Epoch 12/30
56/56 [==============================] - 1806s 32s/step - loss: 0.0643 - accuracy: 0.0066
Epoch 13/30
56/56 [==============================] - 1914s 34s/step - loss: 0.0565 - accuracy: 0.0072
```

Figure 6: Training model and generating weights

- In Figure.7 shows the loading of weights to get the sequences and generating the output file. File path for loading weights need to change. The last generated weight is the weight with the minimum loss. Weights generated by the model are stored in Weights directory. Due to submission memory limits, weight with the minimum loss is placed in the directory.

```python
def create_network(Test_X, n_vocab):
    """ create the structure of the neural network """
    model = Sequential()
    model.add(LSTM(
        512,
        input_shape=(Test_X.shape[1], Test_X.shape[2]),
        recurrent_dropout=0.3,
        return_sequences=True
    ))
    model.add(LSTM(512, return_sequences=True, recurrent_dropout=0.3,))
    model.add(LSTM(512))
    model.add(BatchNorm())
    model.add(Dropout(0.3))
    model.add(Dense(256))
    model.add(Activation('relu'))
    model.add(BatchNorm())
    model.add(Dropout(0.3))
    model.add(Dense(n_vocab))
    model.add(Activation('softmax'))
    epochs = 50
    learning_rate = 0.1
    decay_rate = learning_rate / epochs
    momentum = 0.8
    sgd = SGD(learning_rate==learning_rate, momentum=momentum, decay=decay_rate, nesterov=False)
    model.compile(loss='binary_crossentropy', optimizer=sgd, metrics=['accuracy'])
    print(model.summary())
    model.load_weights('C:/Users/SACHIN/weights-improvement-30-0.0254-bigger.hdf5')

    return model
```

Figure 7: Loading weights

- In Figure.8 shows the loading the weight and taking signatures from "notes" file and generating output file. The path needs to change for the output file and loading notes data.

In [31]:
```python
def create_midi(prediction_output):
    offset = 0
    output_notes = []
    for pattern in prediction_output:
        # pattern is a chord
        if ('.' in pattern) or pattern.isdigit():
            notes_in_chord = pattern.split('.')
            notes = []
            for current_note in notes_in_chord:
                new_note = note.Note(int(current_note))
                new_note.storedInstrument = instrument.Piano()
                notes.append(new_note)
            new_chord = chord.Chord(notes)
            new_chord.offset = offset
            output_notes.append(new_chord)
        else:
            new_note = note.Note(pattern)
            new_note.offset = offset
            new_note.storedInstrument = instrument.Piano()
            output_notes.append(new_note)
        offset += 0.5

    midi_stream = stream.Stream(output_notes)

    midi_stream.write('midi', fp='test_output.mid')
with open('F:/National College of Ireland/Research Project/MLSTM/data/notes', 'rb') as filepath:
    music21_obj = pickle.load(filepath)

pitchnames = sorted(set(item for item in music21_obj))
n_vocab = len(set(music21_obj))
Test_X, normalized_input = prepare_sequences(music21_obj, pitchnames, n_vocab)
model = create_network(normalized_input, n_vocab)
prediction_output = generate_notes(model, Test_X, pitchnames, n_vocab)
create_midi(prediction_output)
```

```
Model: "sequential_3"
```

Figure 8: Generating output

- Figure.9 shows the reading the output file for the analysis and further code shows the visualisations of output file and the analysis.

```
In [43]:
FILENAME="F:/National College of Ireland/Research Project/MLSTM/output.mid"

# Listing current data on our folder.

print(os.listdir("."))


from music21 import converter, corpus, instrument, midi, note, chord, pitch

def open_midi(midi_path, remove_drums):
    # There is an one-line method to read MIDIs
    # but to remove the drums we need to manipulate some
    # low level MIDI events.
    mf = midi.MidiFile()
    mf.open(midi_path)
    mf.read()
    mf.close()
    if (remove_drums):
        for i in range(len(mf.tracks)):
            mf.tracks[i].events = [ev for ev in mf.tracks[i].events if ev.channel != 10]

    return midi.translate.midiFileToStream(mf)

base_midi = open_midi(FILENAME, True)
#print(base_midi)
```

Figure 9: Reading output file to for visualisations

- Sometimes the midi files does not support the normal media players present in the system. These files need DAW or the special player to play the files. The same can be played by using online midi player[1]. Figure.10 shows the representation of output midi file on the online midi player.

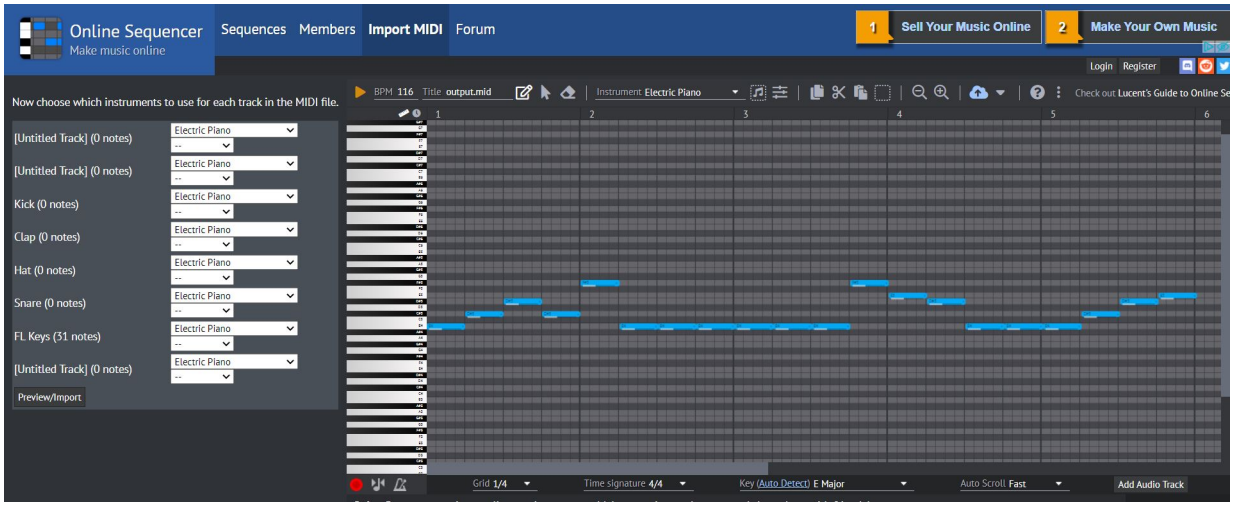

Figure 10: Online Midi player

References

[1]https://onlinesequencer.net/import