In [9]:

```
#import youtube_dl
import glob
import pickle
import numpy
from music21 import converter, instrument, note, chord,stream
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
<ipython-input-9-48da3a54f0fb> in <module>
      3 import pickle
      4 import numpy
----> 5 from music21 import converter, instrument, note, chord,stream

ModuleNotFoundError: No module named 'music21'
```

In [ ]:

```
def to_categorical(y, num_classes=None, dtype='float32'):
    """Converts a class vector (integers) to binary class matrix.

    E.g. for use with categorical_crossentropy.

    # Arguments
        y: class vector to be converted into a matrix
            (integers from 0 to num_classes).
        num_classes: total number of classes.
        dtype: The data type expected by the input, as a string
            (`float32`, `float64`, `int32`...)

    # Returns
        A binary matrix representation of the input. The classes axis
        is placed last.
    """
    y = numpy.array(y, dtype='int')
    input_shape = y.shape
    if input_shape and input_shape[-1] == 1 and len(input_shape) > 1:
        input_shape = tuple(input_shape[:-1])
    y = y.ravel()
    if not num_classes:
        num_classes = numpy.max(y) + 1
    n = y.shape[0]
    categorical = numpy.zeros((n, num_classes), dtype=dtype)
    categorical[numpy.arange(n), y] = 1
    output_shape = input_shape + (num_classes,)
    categorical = numpy.reshape(categorical, output_shape)
    return categorical
```

In [ ]:

```
def get_notes(path='*.mid'):
    """ Get all the notes and chords from the midi files  """
    notes = []

    for file in glob.glob(path):
        midi = converter.parse(file)

        print("Parsing %s" % file)

        notes_to_parse = None

        try: # file has instrument parts
            s2 = instrument.partitionByInstrument(midi)
            notes_to_parse = s2.parts[0].recurse()
        except: # file has notes in a flat structure
            notes_to_parse = midi.flat.notes
        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))

    return notes
```

In [ ]:

```
def download_song_with_url(url,audio_type='mp3',quality='192'):
    #Downloads the song from the url
    ydl_opts = {
    'format': 'bestaudio/best',
    'postprocessors': [{
        'key': 'FFmpegExtractAudio',
        'preferredcodec': audio_type,
        'preferredquality': quality,
          }],
         }
    with youtube_dl.YoutubeDL(ydl_opts) as ydl:
        ydl.download([url])
```

In [ ]:

```
def create_midi(prediction_output,name='output'):
    """ convert the output from the prediction to notes and create a midi file
        from the notes """
    offset = 0
    output_notes = []

    # create note and chord objects based on the values generated by the model
    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)
        # pattern is a note
        else:
            new_note = note.Note(pattern)
            new_note.offset = offset
            new_note.storedInstrument = instrument.Piano()
            output_notes.append(new_note)

        # increase offset each iteration so that notes do not stack
        offset += 0.5

    midi_stream = stream.Stream(output_notes)

    midi_stream.write('midi', fp=name+'.mid')
```

In [ ]:

```
def prepare_sequences(notes, n_vocab,sequence_length = 100):
    """ Prepare the sequences used by the Neural Network """


    # get all pitch names
    pitchnames = sorted(set(item for item in notes))

     # create a dictionary to map pitches to integers
    note_to_int = dict((note, number) for number, note in enumerate(pitchnames))

    network_input = []
    network_output = []

    # create input sequences and the corresponding outputs
    for i in range(0, len(notes) - sequence_length, 1):
        sequence_in = notes[i:i + sequence_length]                #CHECK HERE PROPERLY
        sequence_out = notes[i + sequence_length]
        network_input.append([note_to_int[char] for char in sequence_in])
        network_output.append(note_to_int[sequence_out])

    n_patterns = len(network_input)

    # reshape the input into a format compatible with LSTM layers
    network_input = numpy.reshape(network_input, (n_patterns, sequence_length, 1))
##SEE HERE IF YOU CHANGE
    # normalize input
    network_input = network_input / float(n_vocab)

    network_output = to_categorical(network_output)

    return (network_input, network_output)
```

In [2]:

```
def download_video_with_url(url):

    ydl_opts = {}
    with youtube_dl.YoutubeDL(ydl_opts) as ydl:
        ydl.download([url])
```

In [3]:

```
def download_songs(path):
    data= open(path,'r').readlines()
    for song in data:
        download_song_with_url(song)
```

In [4]:

```
def download_videos(path):
    data= open(path,'r').readlines()
    for video in data:
        download_video_with_url(video)
```

In [5]:

```
def song_notes_to_pickle(path,output):
    notes=get_notes(path)
    with open(output, 'wb') as filepath:     #Writing
        pickle.dump(notes, filepath)
    return notes
```

In [6]:

```
def generate_notes(model, network_input, pitchnames, n_vocab):
    """ Generate notes from the neural network based on a sequence of notes """
    # pick a random sequence from the input as a starting point for the prediction
    start = numpy.random.randint(0, len(network_input)-1)

    int_to_note = dict((number, note) for number, note in enumerate(pitchnames))

    pattern = network_input[start]
    prediction_output = []

    # generate 500 notes
    for note_index in range(500):
        prediction_input = numpy.reshape(pattern, (1, len(pattern), 1))
        prediction_input = prediction_input / float(n_vocab)

        prediction = model.predict(prediction_input, verbose=0)

        index = numpy.argmax(prediction)
        result = int_to_note[index]
        prediction_output.append(result)

        pattern.append(index)
        pattern = pattern[1:len(pattern)]

    return prediction_output
```

In [7]:

```
import os
path='C:/Users/SHAGAF-G/Downloads/Schubert_dataset/'
files=[i for i in os.listdir(path) if i.endswith(".mid")]
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
<ipython-input-7-e959d89281d5> in <module>
      1 import os
      2 path='C:/Users/SHAGAF-G/Downloads/Schubert_dataset/'
----> 3 files=[i for i in os.listdir(path) if i.endswith(".mid")]

FileNotFoundError: [WinError 3] The system cannot find the path specified: 'C:/Users/SHAG
AF-G/Downloads/Schubert_dataset/'
```

In [8]:

```
get_notes('/schu_143_1.mid')
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-8-6a01c7a7a394> in <module>
----> 1 get_notes('/schu_143_1.mid')

NameError: name 'get_notes' is not defined
```

In [ ]:

```

```

In [ ]:

```

```