REPORT
CS 725 PROJECT
Music Generation Using Deep Learning

Sahil Patki (183059002)
Suraj Kumar (18305R008)
Sounak Bhattacharya (183050014)

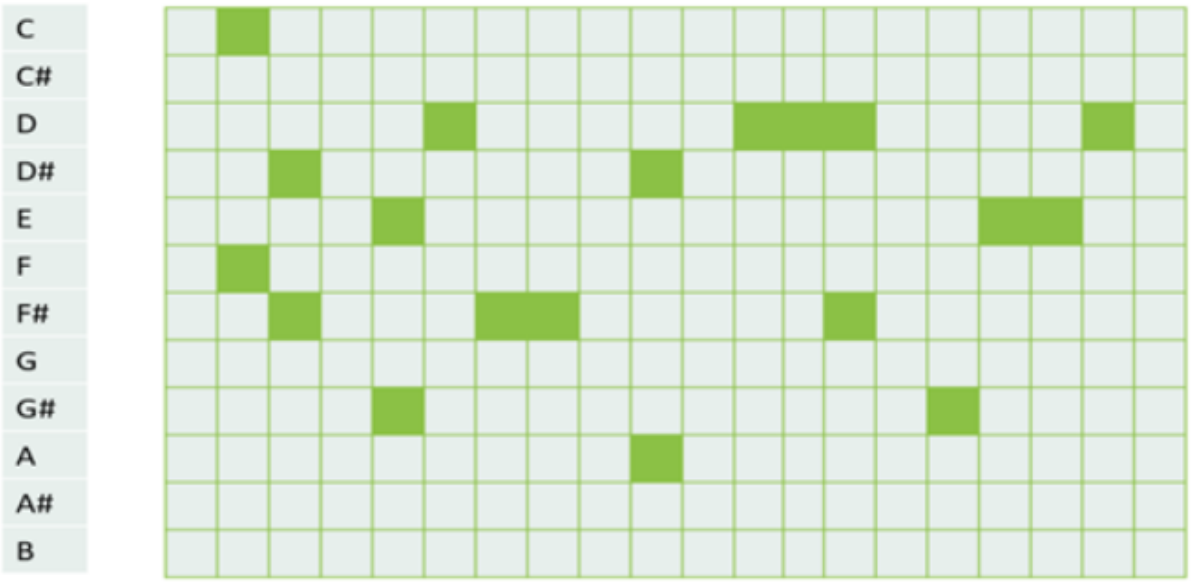
WHY MIDI FILE ?

MORE LIKE A PROGRAMING LANGUAGE DICTATING HOW TO GENERATE THE MUSIC

	MIDI	MP3
Definition	A MIDI file is software for representing musical information in a digital format.	It is a patented encoding format for digital audio.
Abbreviation	Musical Instrument Digital Interface	MPEG-1 or MPEG-2 Audio Layer III
Filename extension	.MIDI or .MID	.mp3
Format type	Compressed	Lossy Compressed
Common use	Computer based musical tools	Mobile phones
Storage	Less storage	More storage compare to MIDI
File Format	Narrow	Wide
Contain	Do not contain a recording of sound	Contain a recording of sound
Application	Compose	Record
Sound	Versatile	Less versatile

Standard MIDI Files (SMF) contain instructions that trigger sounds played by a synthesizer which is typically expected to meet the General MIDI specification. On a personal computer these instructions can be turned into sound by either a software synthesizer, or by sending them along to a hardware synthesizer. As it does not contain actual recording like MP3, it is easy to swap syntheisizing instrument and generate a completely different sound using the same file.

Piano Roll Representation

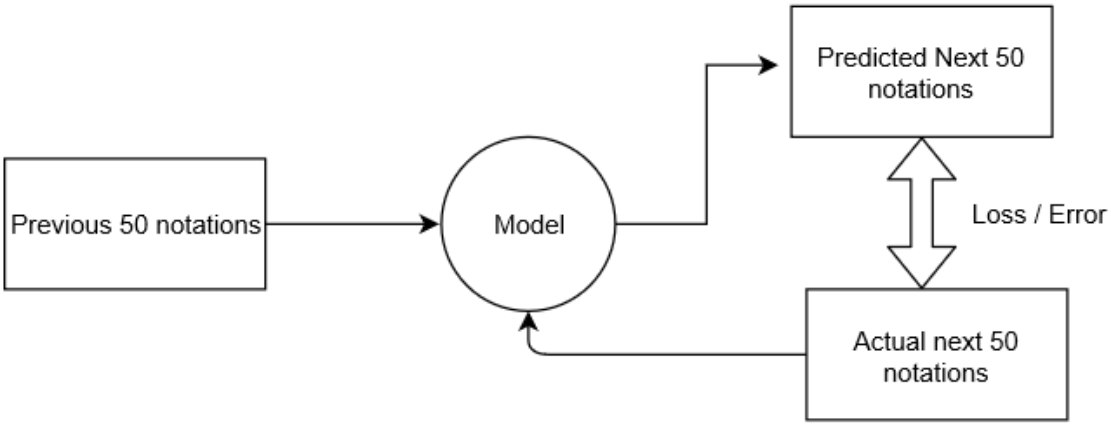


The vertical axis is a digital representation of different notes: for instance, each row in the vertical axis can be associated with its own piano key (i.e. individual musical note). The horizontal axis is a continuous representation of time.

The Input

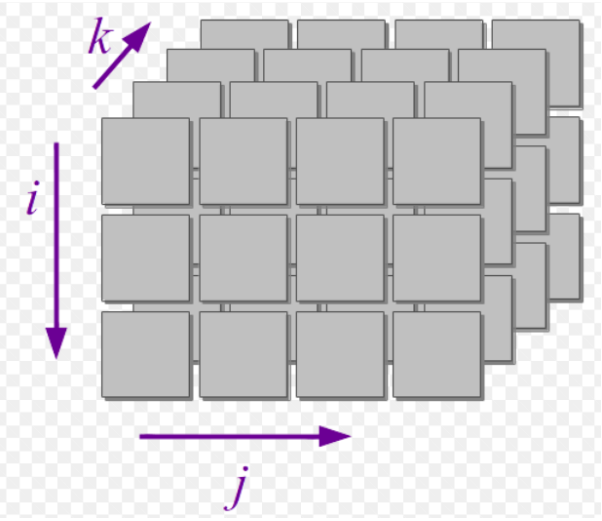
MIDI represents time in ‘ticks’ which essentially represent delta times, i.e. each event’s tick is relative to the previous one. Each MIDI file’s header contains the resolution of that file which gives us the number of ticks per beat. To transform the MIDI file into a piano roll we need to quantize the MIDI events by time. To quantize MIDI events, we simply multiply the tempo (beats per minute) by the resolution (ticks per beat) and that gives us the ticks per second.

Each input is a 2D array: <no_of_notes> x <time_units> = 49 x 50

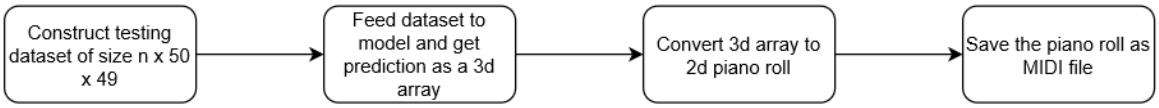


We create a dataset containing the above input format from a single midi file using time slicing

Dataset D of size 2800 x 49 x 50



Generating Music



In [15]:
```
# generate music using RNN
for i,song in enumerate(test_input):
    net_output = model_rnn.predict(song)
    net_roll = seqNetOutToPianoroll(net_output)
    pianorollToMidi(net_roll, 'output_harmony_rnn.mid')
```

In [17]:
```
# generate music using LSTM
for i,song in enumerate(test_input):
    net_output = model_lstm.predict(song)
    net_roll = seqNetOutToPianoroll(net_output)
    pianorollToMidi(net_roll, 'output_harmony_lstm.mid')
```

seqNetOutToPianoRoll() flattens the 3d floating array (due to softmax output) to a 2d binary array by comparing with a threshold.

```
def seqNetOutToPianoroll(output, threshold = 0.1):
    piano_roll = []
    for seq_out in output:
        for time_slice in seq_out:
            idx = [i for i,t in enumerate(time_slice) if t > threshold]
            pianoroll_slice = np.zeros(time_slice.shape)
            pianoroll_slice[idx] = 1
            piano_roll.append(pianoroll_slice)

    return np.array(piano_roll)
```

The Models

INPUT_DIM = OUTPUT_DIM = number of notes in the file

Dropout used as regularization technique

```
def createSeq2Seq_LSTM():
    #encoder
    model = Sequential()
    model.add(LSTM(input_dim = INPUT_DIM, output_dim = NUM_UNITS, dropout=0.2, recurrent_dropout=0.2, return_sequences = True
    model.add(BatchNormalization())
    model.add(LSTM(NUM_UNITS))

    #decoder
    model.add(RepeatVector(Y_SEQ_LENGTH))
    num_layers= 2
    for _ in range(num_layers):
        model.add(LSTM(NUM_UNITS, dropout=0.2, recurrent_dropout=0.2, return_sequences = True))
        model.add(BatchNormalization())

    model.add(TimeDistributed(Dense(OUTPUT_DIM, activation= 'softmax')))
    return model
```

```
def createSeq2Seq_RNN():
    #encoder
    model = Sequential()
    model.add(SimpleRNN(input_dim = INPUT_DIM, output_dim = NUM_UNITS, dropout=0.2, recurrent_dropout=0.2, return_sequences =
    model.add(BatchNormalization())
    model.add(SimpleRNN(NUM_UNITS))

    #decoder
    model.add(RepeatVector(Y_SEQ_LENGTH))
    num_layers= 2
    for _ in range(num_layers):
        model.add(SimpleRNN(NUM_UNITS, dropout=0.2, recurrent_dropout=0.2, return_sequences = True))
        model.add(BatchNormalization())

    model.add(TimeDistributed(Dense(OUTPUT_DIM, activation= 'softmax')))
    return model
```