## Step 1: Cleaning dataset

After collecting the dataset, since each piano music sequences contains songs in two channels namely, left piano and right piano. Most of the main music melody, the crux, of the music is usually contained or played by left piano channel. In order to simplify our learning step for the model, we will be deleting all the notes and chords played through the right channel.

This can be done by using an online service website in which one has to upload the midi. File and the dash board give us a virtual midi player along with a panel menu keeping track of channels present or used in the midi file. We can delete any channel by right clicking on the channel and click delete track.

**Step 2: Data Preprocessing.**

In this step preprocessing of the midi files is done using music21 lib.

1. LOADING DATASET

Making the list of location of all the midi files we have and used music21 library to parse all the songs and get a Score object of those music midi files using their location. Furthermore, making a list of all the parsed midi files Score music21 object.

1. TRANSPOSING SONGS

We then transposed keys of the songs to C major if key of the song is major or A minor if key of the song is minor.

Transposing can be understood as if we have a text in Times New Roman changing the font to Calibri. One could see the visual changes but the content remains same. Likewise, transposing here makes the sound of the music bit in lighter tone but the melodic information remains same.

**Step 3: Encoding and forming MIDI sequences.**

Now these transposed music21 Score Objects are converted into sequences or list of time series representation for eg:[60,"\_","\_","\_",62,"\_","\_","\_","\_",62,"\_",72,"\_",70,"\_",80,"r",........]

In a grand piano there are 88 keys and in midi reference each keys are assigned a number like C4 key is assigned number 60. Likewise these 88 keys are assigned number from 21 – 108.In this step we read the events such as occurrences of notes and rests (excluding chords i.e. when two or more notes played together in that same channel)

The list of a song is appended with the note number when that event is encountered, is appended with ‘r’ symbol if there is the rest within notes or ‘\_’ symbol signifies the holding of the note currently pressed. Like if C4 is pressed and is held for then the sequence will be like [60 , ‘\_’ , ‘\_’,’\_’] . now A list is made of all these converted music sequence lists ( Rows = songs , columns = event encountered)

Now this list is saved as a single file dataset file in the text form and end of each song sequence is appended with N number of “/”. Where N is usually number of sequence lengths we want our model to output.

Then all these symbols inside are the list are replaced by the numbers from 0 to numbers of unique events present. Example “\_” will become 0, “r” will become 1 and notes would be given numbers.

This is done to get better functioning of the model and easy understanding, as RNN only deal with numbers.

**Step 4: Generating Training Sequences.**

After forming midi sequences, since all the song sequence are varying in length they are converted into sequences containing 64 events but in a progressive manner. Like suppose there is a sequence having 10 events numbered from 1 to 10 and we want to make sequences out of it containing 4 events each, then first sequence would be [1,2,3,4] then the next sequences would be [2,3,4,5] , [3,4,5,6], [4,5,6,7],[5,6,7,8],[6,7,8,9] and lastly [7,8,9,10].

This done because in music has high temporal dependency hence we need our model to learn this notes time dependency.

This equal length sequences will be our input and the first index number of the next sequence would be the target for that sequence. Hence we get our input and targets. Since we are treating this prediction problem as supervised learning so we need inputs and targeted output.

**Step 5: Training model, Experimenting architectures.**

We are using 64 length sequences (model hyper parameter) as input for our model and argmax of Softmax. And Categorical Cross Entropy as our loss function. Thus, we need to convert list of target values to One-hot Encodings.

Our model is then trained with these 3d input as (no. of sequences, events in a sequence, batch size =1)

Using adam as our optimizer.

Further, saving our model with .h5 extension using keras’ save model function.

**Step 6: Post-Processing i.e. generating predictions.**

We will then load the model and use a seed sequence, feed it to the model and let it generate next event till the specified steps.

The returned probabilities by our Softmax function for all the probable events are scaled by using a factor named temperature.

TEMPERATURE:

* we can return the index with the highest probability , but it’s a rigid choice
* thus we going to use temperature to bring these probability values closer AKA scaling the distribution
* temp->inf. ----- all the different values tend to have same value
* tem->0 ------ all the values become deterministic highest value become 1 low 0
* temp-> 1 ------ does nothing to the original distribution
* intuition ==> value is directly proportional to predictability of the sampling
* 0 - deterministic inf. - random

**Step 7: Converting sequences into MIDI files.**

We will then convert the generated sequence map them to appropriate midi note number and using music21 library to map those events to appropriate notes. Saving the file as .midi which can be played again using the virtual instrument simulators and players available online.

***Using dropout inside the function parameter or outside?***

***You can use a Dropout(...) layer, it's not "wrong", but it will possibly drop "timesteps" too! (Unless you set noise\_shape properly or use SpatialDropout1D, which is currently not documented yet)***

***Maybe you want it, maybe you dont. If you use the parameters in the recurrent layer, you will be applying dropouts only to the other dimensions, without dropping a single step. This seems healthy for recurrent layers, unless you want your network to learn how to deal with sequences containing gaps (this last sentence is a supposal).***

***Also, with the dropout parameters, you will be really dropping parts of the kernel as the operations are dropped "in every step", while using a separate layer will let your RNN perform non-dropped operations internally, since your dropout will affect only the final output.***