REPORT ON

TENSORFLOW: GENERATE MUSIC WITH RNN

https://www.tensorflow.org/tutorials/audio/music_generation

BY :

Aman Srivastava - MT21007
Mahak Sharma - MT21047
Rishabh Kumar Pundhir - MT21071
Rutvikkumar Bandhaniya - MT21116

This tutorial shows you how to generate musical notes using a simple recurrent neural network (RNN). You will train a model using a collection of piano MIDI files from the MAESTRO dataset. Given a sequence of notes, your model will learn to predict the next note in the sequence.

Setup Requirements -

This tutorial uses the pretty_midi library to create and parse MIDI files, and pyfluidsynth for generating audio playback in Colab.

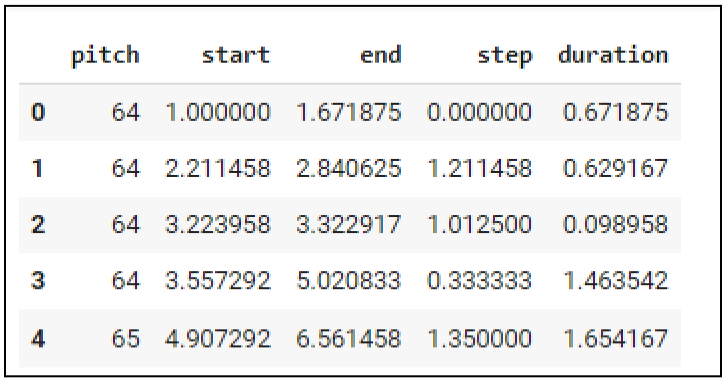
The dataset contains about 1282 MIDI files.

Playing a midi file and extracting the notes

We are using a sample MIDI file for displaying it using prettyMIDI for 30 seconds and considering the SAMPLE_RATE = 16,000.

Notes extracted from the MIDI file contain 'Pitch', 'Note Name' and 'Duration' as variables.

The **pitch** is the perceptual quality of the sound as a MIDI note number. The **step** is the time elapsed from the previous note or start of the track. The **duration** is how long the note will be playing in seconds and is the difference between the note end and note start times.

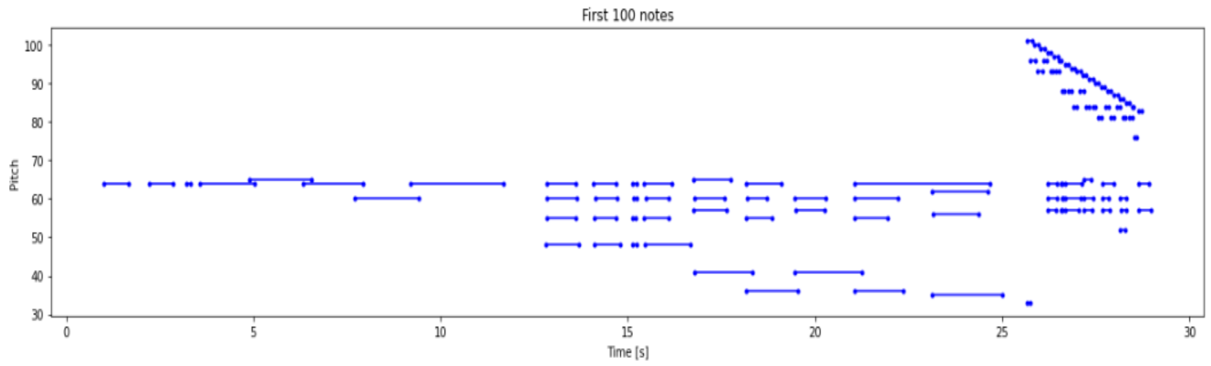


	pitch	start	end	step	duration
0	64	1.000000	1.671875	0.000000	0.671875
1	64	2.211458	2.840625	1.211458	0.629167
2	64	3.223958	3.322917	1.012500	0.098958
3	64	3.557292	5.020833	0.333333	1.463542
4	65	4.907292	6.561458	1.350000	1.654167

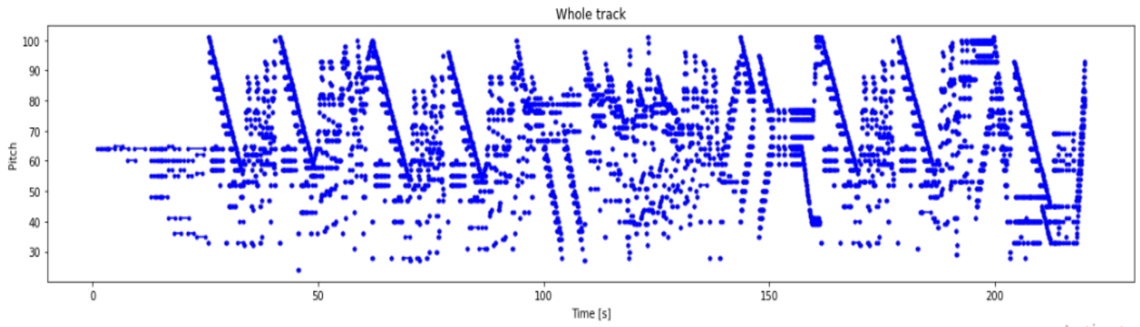
It may be easier to interpret the note names rather than the pitches, so you can use the function below to convert from the numeric pitch values to note names. The note name shows the type of note, accidental and octave number.

```
array(['E4', 'E4', 'E4', 'E4', 'F4', 'E4', 'C4', 'E4', 'C3', 'C4'],
      dtype='<U3')
```

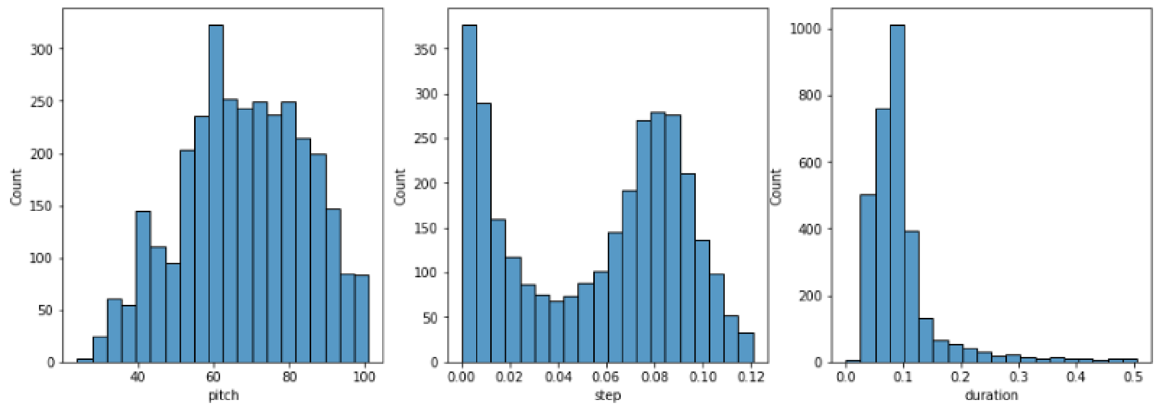
To visualize the musical piece, plot the note pitch, start and end across the length of the track (i.e. piano roll). Start with the first 100 notes



For whole track -



Distribution of each note variable -

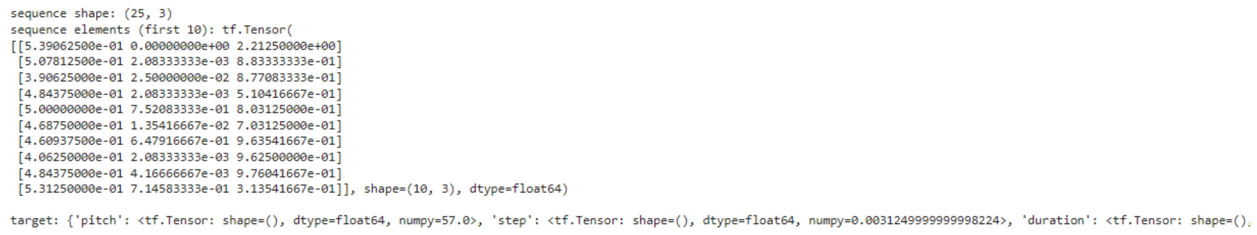


Create the training dataset -

We created the dataset using a small number of MIDI files initially and used all the notes present in these files for making a training dataset. We trained the model on batches of sequences of notes. Each example will consist of a sequence of notes as the input features, and the next note as the label. In this way, the model will be trained to predict the next note in a sequence.

Set the sequence length for each example. The size of the vocabulary (vocab_size) is set to 128 representing all the pitches supported by pretty_midi. seq_length = 25 and vocab_size = 128 are chosen for creating the sequence for the notes in training data.

The Targets chosen here are 'Pitch', 'Note Name' and 'Duration' and batch size of 64.



```
sequence shape: (25, 3)
sequence elements (first 10): tf.Tensor(
[[5.39062500e-01 0.00000000e+00 2.21250000e+00]
 [5.07812500e-01 2.08333333e-03 8.83333333e-01]
 [3.90625000e-01 2.50000000e-02 8.77083333e-01]
 [4.84375000e-01 2.08333333e-03 5.10416667e-01]
 [5.00000000e-01 7.52083333e-01 8.03125000e-01]
 [4.68750000e-01 1.35416667e-02 7.03125000e-01]
 [4.60937500e-01 6.47916667e-01 9.63541667e-01]
 [4.06250000e-01 2.08333333e-03 9.62500000e-01]
 [4.84375000e-01 4.16666667e-03 9.76041667e-01]
 [5.31250000e-01 7.14583333e-01 3.13541667e-01]], shape=(10, 3), dtype=float64)

target: {'pitch': <tf.Tensor: shape=(), dtype=float64, numpy=57.0>, 'step': <tf.Tensor: shape=(), dtype=float64, numpy=0.0031249999999998224>, 'duration': <tf.Tensor: shape=()
```

Creating and training the model -

The model will have three outputs, one for each note variable. For step and duration, you will use a custom loss function based on mean squared error that encourages the model to output non-negative values.

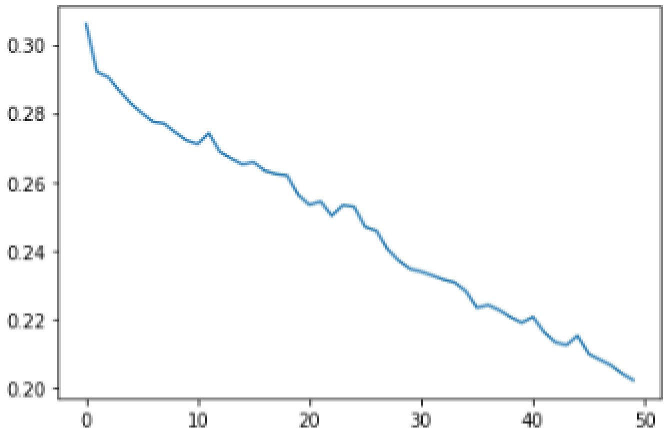
```
Model: "model"
________________________________________________________________________________
 Layer (type)              Output Shape       Param #    Connected to
================================================================================================
 input_1 (InputLayer)        [(None, 25, 3)]     0          []

 lstm (LSTM)               (None, 128)       67584      ['input_1[0][0]']

 duration (Dense)            (None, 1)         129       ['lstm[0][0]']

 pitch (Dense)             (None, 128)       16512      ['lstm[0][0]']

 step (Dense)              (None, 1)        129       ['lstm[0][0]']
================================================================================================
Total params: 84,354
Trainable params: 84,354
Non-trainable params: 0
________________________________________________________________________________
```

While testing the model , we can see that the pitch loss is significantly greater than the step and duration losses. Note that loss is the total loss computed by summing all the other losses and is currently dominated by the pitch loss. For which we have used the **loss_weights** - 'pitch': 0.05, 'step': 1.0, and 'duration':1.0.

For fitting the model, batch_size=64, epochs=50.

Training of model took around 10 mins using GPU runtime.

```
plt.plot(history.epoch, history.history['loss'], label='total loss')
plt.show()
```



Here X-axis - Training Loss and Y-axis - No. of Epochs

Results from the Trained model -

Sample file for which we are going to generate the notes/MIDI files.

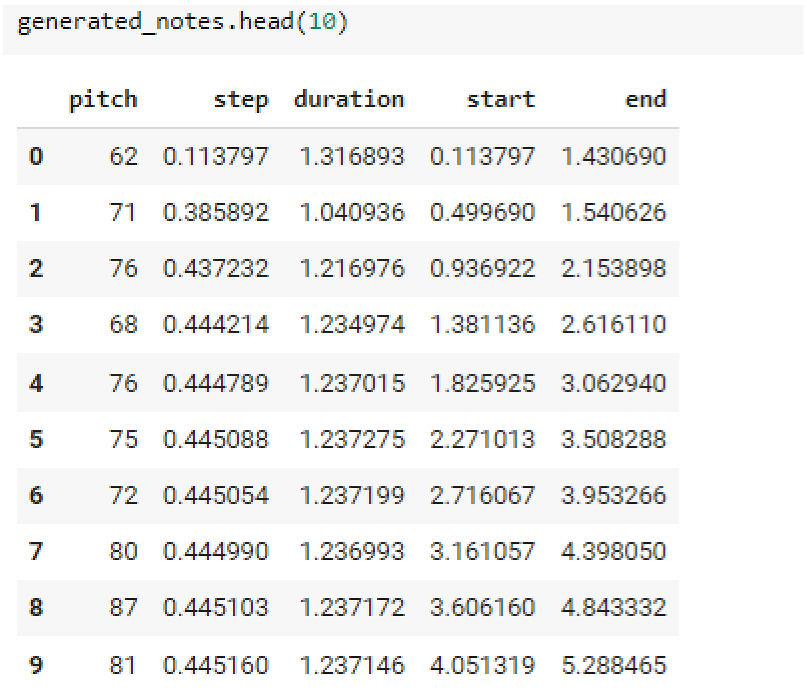
```
sample_file = filenames[2]
print(sample_file)
```

```
data/maestro-v2.0.0/2017/MIDI-Unprocessed_051_PIANO051_MID--AUDIO-split_07-06-17_Piano-e_3-02_wav--3.midi
```

We first provide the starting sequences of the notes, for which we generated one note for a sequence of notes. We have generated longer sequences of notes by calling the model repeatedly for a set of sequence notes.

Used **'temperature'** as it can be used to control the randomness of notes generated, temperature = 2.0 and num_predictions = 120 were used for generating a new sequence of notes.

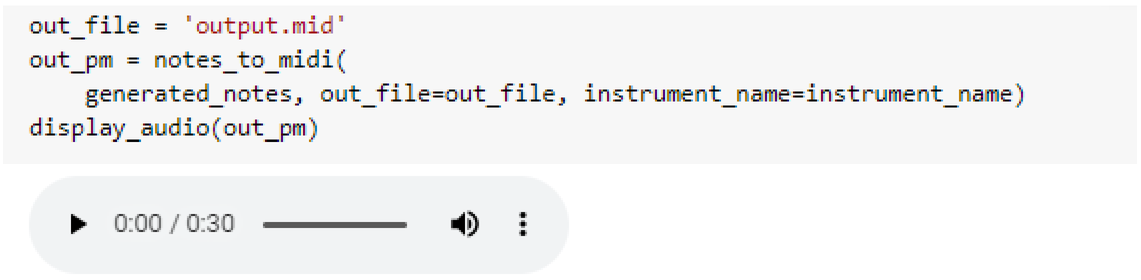
Description of new notes generated after repeatedly calling the model for a random file from the MAESTRO Dataset.



```
generated_notes.head(10)
```

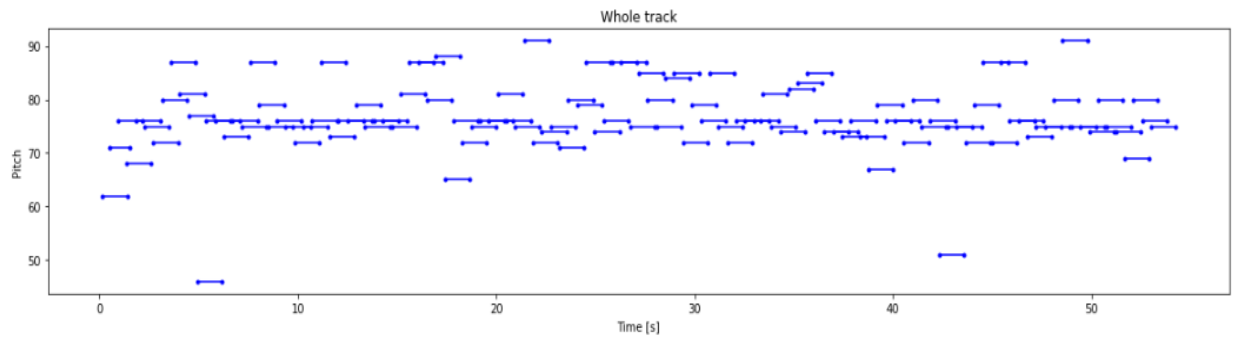
	pitch	step	duration	start	end
0	62	0.113797	1.316893	0.113797	1.430690
1	71	0.385892	1.040936	0.499690	1.540626
2	76	0.437232	1.216976	0.936922	2.153898
3	68	0.444214	1.234974	1.381136	2.616110
4	76	0.444789	1.237015	1.825925	3.062940
5	75	0.445088	1.237275	2.271013	3.508288
6	72	0.445054	1.237199	2.716067	3.953266
7	80	0.444990	1.236993	3.161057	4.398050
8	87	0.445103	1.237172	3.606160	4.843332
9	81	0.445160	1.237146	4.051319	5.288465

After which notes were converted to MIDI file and displayed on our colab notebook.



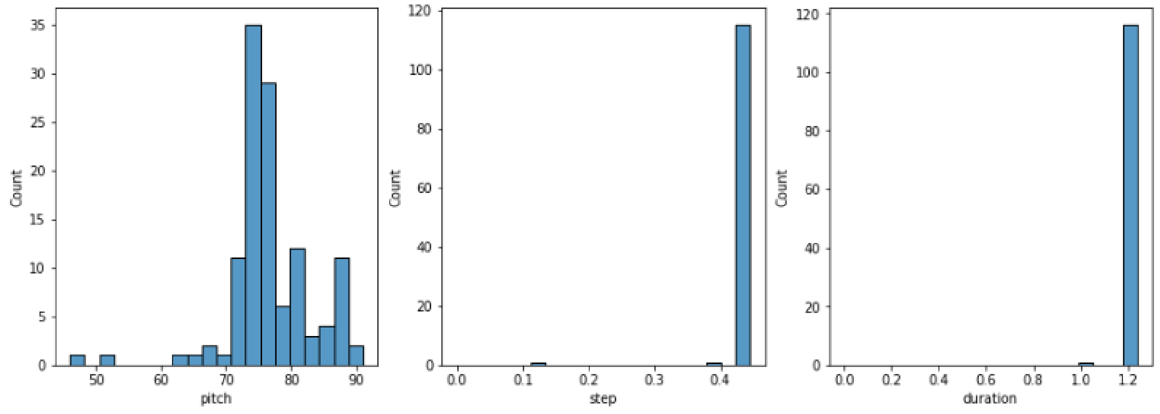
```
out_file = 'output.mid'
out_pm = notes_to_midi(
    generated_notes, out_file=out_file, instrument_name=instrument_name)
display_audio(out_pm)
```

To visualize the generated music, plot the note pitch, start and end for the whole track.



Distribution of each note variable -

```
plot_distributions(generated_notes)
```



Using the above plots we can also compare our results with the sample file which we have used to generate new musical notes from our trained model.