# Music Generation Using LSTM and Transformers

**Keerthana Golla (kg58), Rice University**

## Abstract

In the realm of artificial intelligence and machine learning, this project embarks on a journey at the intersection of technology and music composition. Leveraging the music21 toolkit, we explore the creative potentials of two distinct neural network architectures: Long Short-Term Memory (LSTM) networks and Transformers. LSTM networks are known for their proficiency in handling sequential data, while Transformers excel in capturing long-distance dependencies and contextual nuances. This report outlines our endeavors to generate music in various predefined styles, with a particular emphasis on melodies.

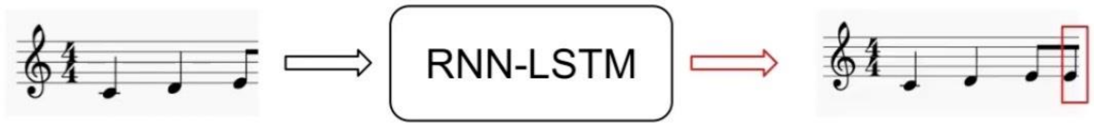


Fig 1a- Next Note is generated.

The project's core objectives lie in harnessing the unique capabilities of LSTM networks and Transformers for music composition. We meticulously curate two diverse datasets in MIDI and Kern formats, drawn from Jazz Kaggle and humdrum.org, respectively. The preprocessing steps involve simplifying musical notations, restricting note durations, and focusing on fundamental keys to ensure a manageable scope. We delve into a comprehensive literature review, exploring existing research on music generation using LSTM networks (visually shown in fig 1a, 1b), Keras, and MIDI format files, grounding our technical choices in established methodologies.

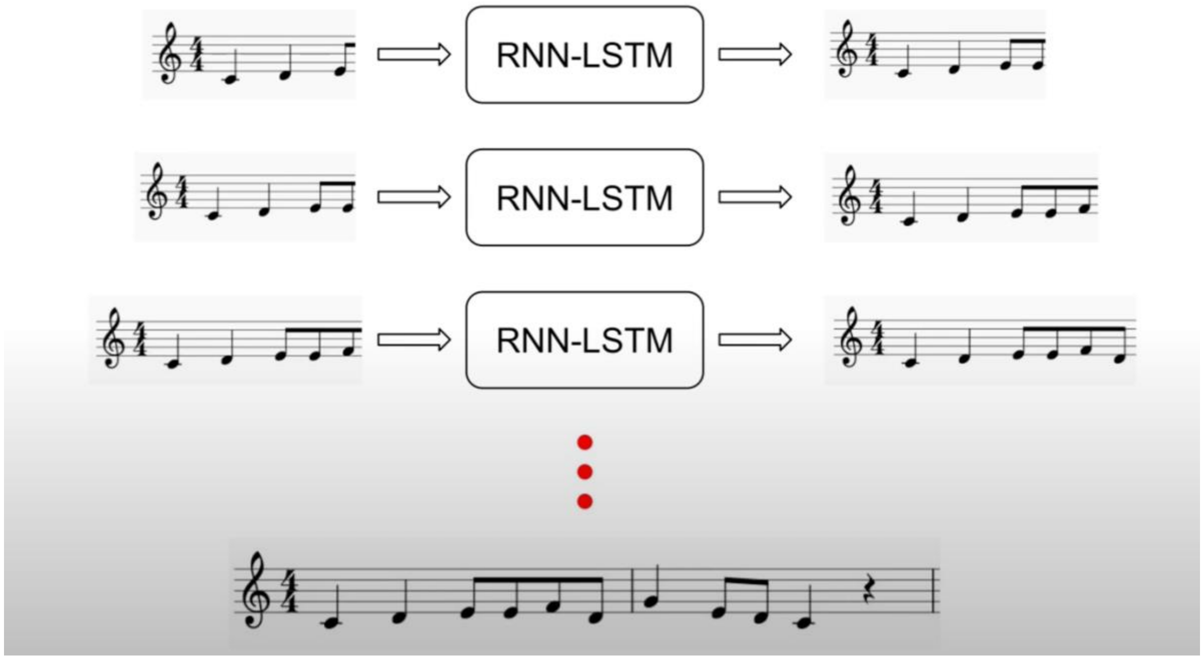


Fig 1b. Overall LSTM High Level Architecture

Our technical approach encompasses detailed preprocessing of datasets, LSTM model architecture design, training processes, and considerations for overfitting mitigation. Noteworthy modifications, such as generating sequences of a specified length, are introduced to enhance the creative aspect of melody generation. The report also delves into the initial results, shedding light on accuracy metrics, loss considerations, and challenges posed by GPU limitations. As the project unfolds, future work involves leveraging Google Colab for transformer models, providing a roadmap for further exploration and refinement in the realms of AI-driven music composition. I personally am interested in merging music domain with video domain. Especially generating music from dance videos.

## Introduction

This project navigates the intersection of neural networks and music composition, seeking to unravel the intricate threads that bind technology and creativity. Through the lens of the music21 toolkit, we embark on an exploration of the collaborative potential of two powerful neural network architectures: Long Short-Term Memory (LSTM) networks and Transformers. These architectures, renowned for their capabilities in handling sequential data and capturing intricate dependencies, hold the promise of reshaping the landscape of music generation.

## Problem Statement

Despite the strides made in AI-driven music composition, the nuanced artistry of generating melodies remains a complex challenge. The traditional dichotomy between LSTM networks and

Transformers prompts a pivotal question: can the synthesis of these two architectures amplify the creative possibilities in music generation? Additionally, the choice of dataset formats—MIDI and Kern—brings forth considerations about representation and computational efficiency. The project navigates through the intricacies of dataset preprocessing, model training, and the convergence of LSTM and Transformer models, aiming to not only decode the language of music but also push the boundaries of what can be created through artificial intelligence.

## Dataset

The dataset for this music generation project is a harmonious amalgamation of melodies sourced from two distinctive formats—MIDI and Kern. Originally comprising a vast repository of 20,000 songs procured from Kaggle and humdrum.org, the exigencies of computational limitations prompted me to focus on a more manageable subset of 2,000 songs.

The pivotal decision in crafting this dataset lies in the choice of MIDI and Kern formats. This deliberate selection is informed by the inherent structure of these formats, which encapsulate melodies as sequential arrangements of notes and rests. Unlike the convolutional complexities of MP3 or WAV formats, which demand the use of spectrograms, MIDI and Kern align seamlessly with the temporal fabric of music. Embracing the sequential nature of melodies is paramount in the realm of time-series prediction, and MIDI/Kern formats offer an unparalleled advantage in deciphering the language of musical notes. The choice of unidirectional LSTM and Bidirectional Encoder Representations from Transformers (BERT) is intricately tied to the unique sequential representation encapsulated in these formats, allowing our models to discern and capture the nuanced interplay of musical elements.

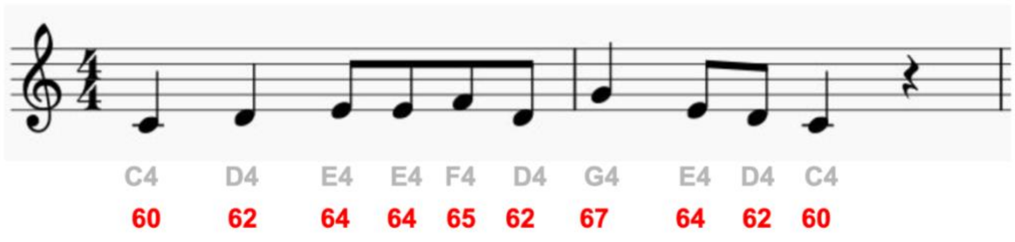


Fig 2 Notes – Int representation

## Literature review

In the pursuit of advancing music generation using artificial intelligence and machine learning, various research efforts have explored different approaches and techniques. One notable paper [1] delves into the use of MIDI format files without imposing restrictions on keys or notes. MIDI files, being a standard format for representing musical information, offer a versatile foundation for music generation.

The articles [2] and [3] provide practical insights into music generation using Long Short-Term Memory (LSTM) neural networks and the Keras library. These resources not only utilize MIDI format files but also address potential computational challenges by discussing strategies to work with a single format file while considering key restrictions.

Additionally, a paper [4] explores an alternative approach using Generative Adversarial Networks (GANs) for music generation. While GANs present a different paradigm, this project is guided by the decision to focus on LSTM neural networks and Transformer models. This choice is grounded in the understanding that music representation aligns with time series data, a concept emphasized in recent class discussions.

## Methods

To systematically address the intricacies of music generation, the project unfolds in a coherent sequence of three Python files: 'Preprocess.py,' Training py files - 'LSTMmodelTrain.py for LSTM', 'Transformers.py for Transformers' and 'MelodyGenerator.py.' Each file encapsulates a distinct facet of the music generation pipeline, ensuring modularity and clarity in the implementation.

### Preprocessing

The heart of the preprocessing pipeline is the 'Preprocess.py' script, meticulously designed to transform raw Kern format songs into a format amenable to neural network training. This initial stage commences with the loading of melodies from the Kern format, employing the music21 library to facilitate seamless manipulation of musical data. A pivotal aspect of this phase involves the judicious filtering of songs to retain only those with durations conforming to predefined acceptable patterns as shown in fig 3. This step ensures the quality of the dataset, aligning it with the desired rhythmic structures.

Once filtered, the script advances to the crucial task of transposing the songs to a standardized key, a transformation that fosters uniformity and consistency across the entire dataset. This normalization process is vital for subsequent stages, enabling the model to learn generalizable patterns and compositions. Subsequently, the encoded representation takes center stage, converting each note or rest into a time-series-like format. The script iteratively processes the loaded songs, transposing them to the C major or A minor key for standardized representation across the dataset. This representation forms the bedrock of the dataset, allowing for structured input into the neural network models.

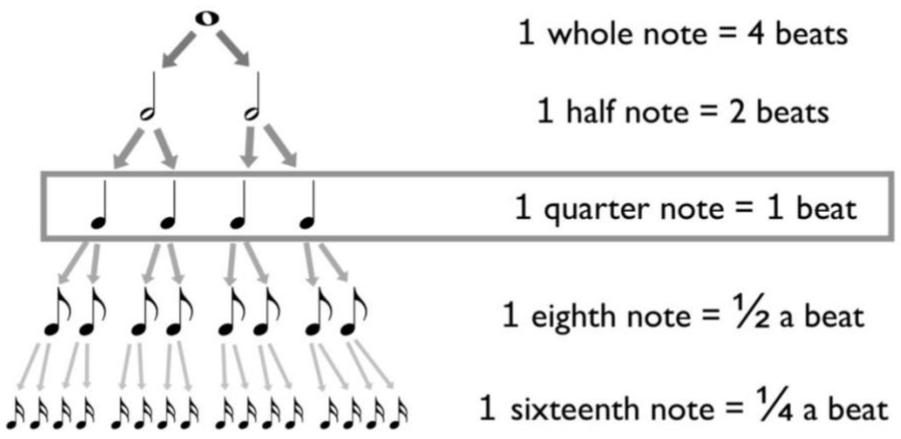


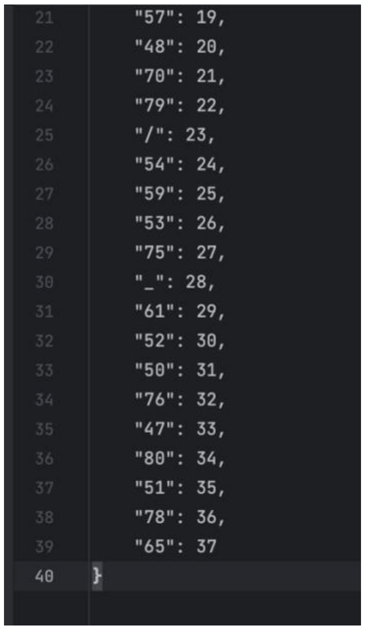
Fig 3 Acceptable Durations

As the preprocessing journey unfolds, the preprocessed songs find unity in a single, consolidated file. This strategic merging of songs streamlines accessibility and management during subsequent stages of model training and evaluation. Simultaneously, symbol-to-integer mappings are forged, establishing a structured vocabulary for the dataset. This mapping is paramount for the models, enabling them to interpret and generate melodies based on the integer representations of musical symbols.

The final flourish of the preprocessing symphony is the creation of training sequences, each carefully constructed to encapsulate the context necessary for effective neural network training. These sequences, composed with precision, lay the foundation for the subsequent steps in the project. The 'Preprocess.py' script, an orchestration of intricate transformations, thus concludes its role in preparing the dataset for the melodious journey that follows.

**Training**

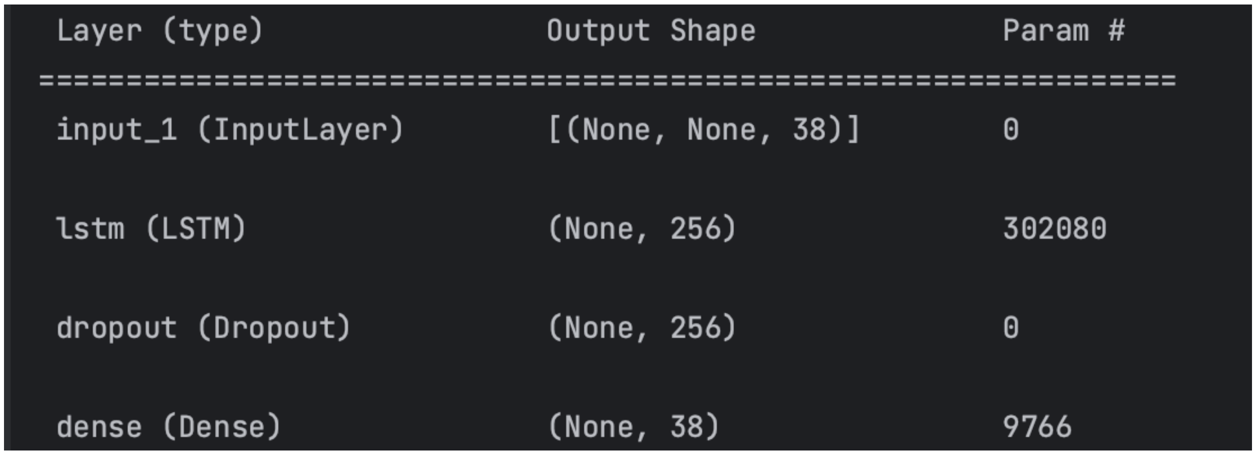
The LSTM model is meticulously engineered to decode intricate temporal patterns within the dataset, navigating through a configuration characterized by 38 nodes(mapping size is shown in fig 4) in the input layer. A strategically placed dropout layer enhances the model's adaptability, preventing overfitting by introducing randomness. Employing the Adam optimizer with sparse categorical cross-entropy loss facilitates nuanced convergence.



```
    "57": 19,
    "48": 20,
    "70": 21,
    "79": 22,
    "/": 23,
    "54": 24,
    "59": 25,
    "53": 26,
    "75": 27,
    "_": 28,
    "61": 29,
    "52": 30,
    "50": 31,
    "76": 32,
    "47": 33,
    "80": 34,
    "51": 35,
    "78": 36,
    "65": 37
}
```

Fig 4 mapping size of 38 (0-37)

The architectural finesse extends to the output layer, designed with SoftMax activation to yield a probability distribution of potential musical symbols. Optimization nuances, coupled with the choice of hyperparameters, accentuate the model's capacity to capture the essence of musical sequences. The amalgamation of dropout layers and early stopping mechanisms ensures the models to avoid overfitting, model layers can be seen in fig 5.



```
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, None, 38)]        0

 lstm (LSTM)                 (None, 256)               302080

 dropout (Dropout)           (None, 256)               0

 dense (Dense)               (None, 38)                9766
```

Fig 5 Model Summary

Post-training, the LSTM model finds its sanctum in an h5 file(added this h5 file in drive here for future access), encapsulating not just learned parameters but the very blueprint of its neural architecture. This repository becomes a linchpin for subsequent melody generation endeavors, bridging the realm of training intricacies with real-world musical creativity.

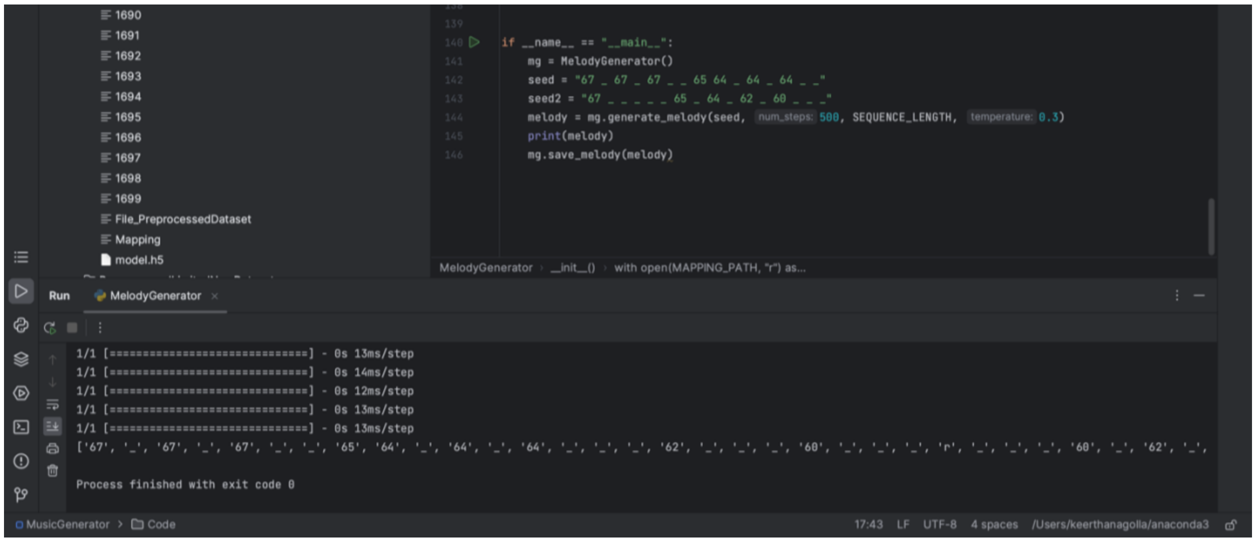
The Transformer model, celebrated for its prowess in deciphering long-range dependencies, is sculpted with BERT embeddings and a pooling layer. Its input layer, tailored for BERT input_ids and attention_mask, leverages contextual embeddings for effective music sequence comprehension. Persisting with the Adam optimizer, the Transformer's hyperparameters are meticulously tuned to optimize convergence and overall model performance.

The introduction of a pooling layer plays a pivotal role, summarizing embeddings for a comprehensive understanding of the entire music sequence. Dropout layers, strategically inserted, infuse randomness during training, enhancing the model's capacity for diverse and coherent melody production. The training process is diligently logged, with progress tracked using a variety of technical metrics, ensuring a fine-tuned model ready for deployment. Like its LSTM counterpart, the trained Transformer model is preserved in an h5 file (added this h5 file in drive here for future access), a testament to the encoded musical knowledge ready for real-world creative endeavors.

## Melody Generation

After the LSTM model is trained, the MelodyGenerator class takes center stage in the composition process. When invoking the generate_melody method, a seed melody is provided, along with parameters specifying the number of steps to generate, the maximum sequence length to consider, and the temperature for controlling the stochasticity of predictions. The seed is then augmented with start symbols, mapped to integers, and fed into the LSTM model, as shown in fig 6.

Internally, the model predicts the next symbols in the sequence based on the seed. The _sample_with_temperature function employs a temperature adjusted SoftMax mechanism to select the most probable symbol. This process iterates for the specified number of steps, gradually extending the seed into a full-fledged melody. Notably, the output includes symbols representing both notes and rests, reflecting the LSTM's comprehension of musical structure.



1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
File_PreprocessedDataset
Mapping
model.h5
MelodyGenerator › __init__() › with open(MAPPING_PATH, "r") as...
Run
MelodyGenerator
MusicGenerator › Code
17:43 LF UTF-8 4 spaces /Users/keerthanagolla/anaconda3

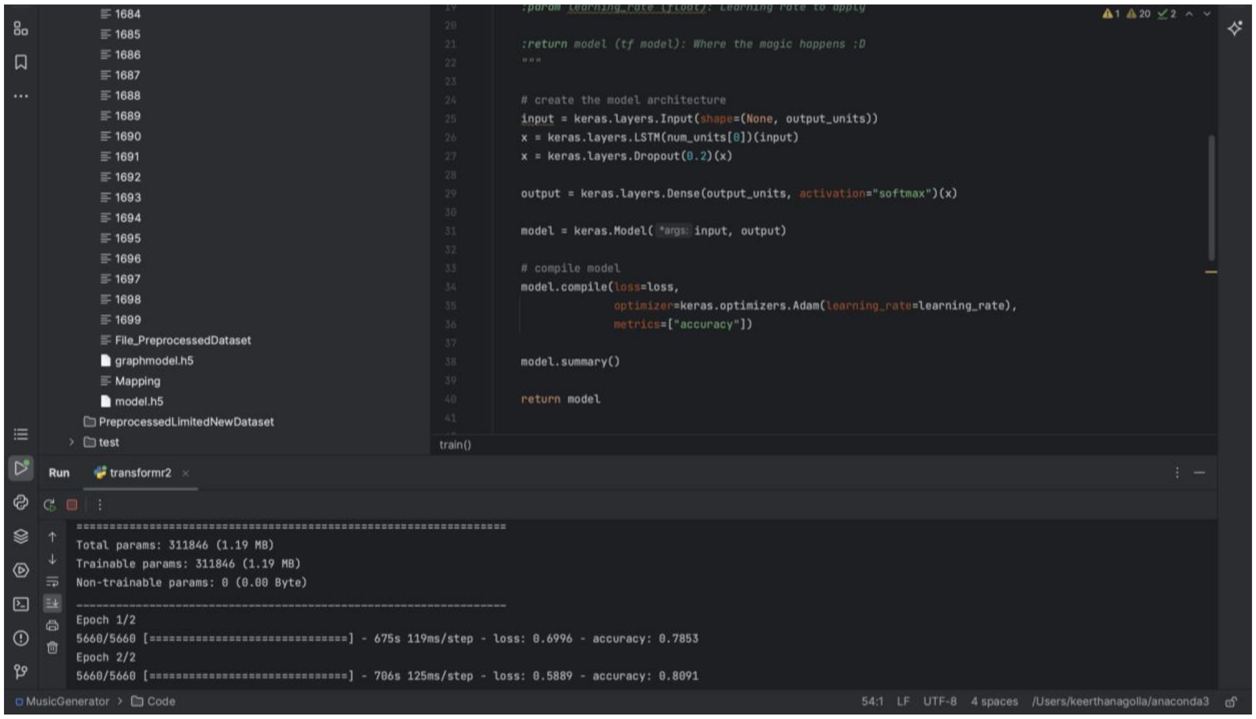
```python
if __name__ == "__main__":
    mg = MelodyGenerator()
    seed = "67 _ 67 _ 67 _ _ 65 64 _ 64 _ 64 _ _"
    seed2 = "67 _ _ _ _ _ 65 _ 64 _ 62 _ 60 _ _ _"
    melody = mg.generate_melody(seed, num_steps: 500, SEQUENCE_LENGTH, temperature: 0.3)
    print(melody)
    mg.save_melody(melody)
```

```
1/1 [==============================] - 0s 13ms/step
1/1 [==============================] - 0s 14ms/step
1/1 [==============================] - 0s 12ms/step
1/1 [==============================] - 0s 13ms/step
1/1 [==============================] - 0s 13ms/step
['67', '_', '67', '_', '67', '_', '_', '65', '64', '_', '64', '_', '64', '_', '_', '_', '62', '_', '_', '_', '60', '_', '_', '_', 'r', '_', '_', '_', '60', '_', '62', '_',

Process finished with exit code 0
```

Fig 6 Melody Generation with seed and symbols displayed.

The generated melody, a sequence of encoded symbols, is then translated into a musically meaningful form. The save_melody method meticulously parses the encoded symbols, considering both note/rest durations and specific pitches. The resulting melody is crafted as a Music21 stream, a musical representation, and then saved as a MIDI file. This integration of neural network predictions and subsequent translation into a musical format is where the technical and artistic realms coalesce, showcasing the model's ability to produce harmonious and structured melodies.

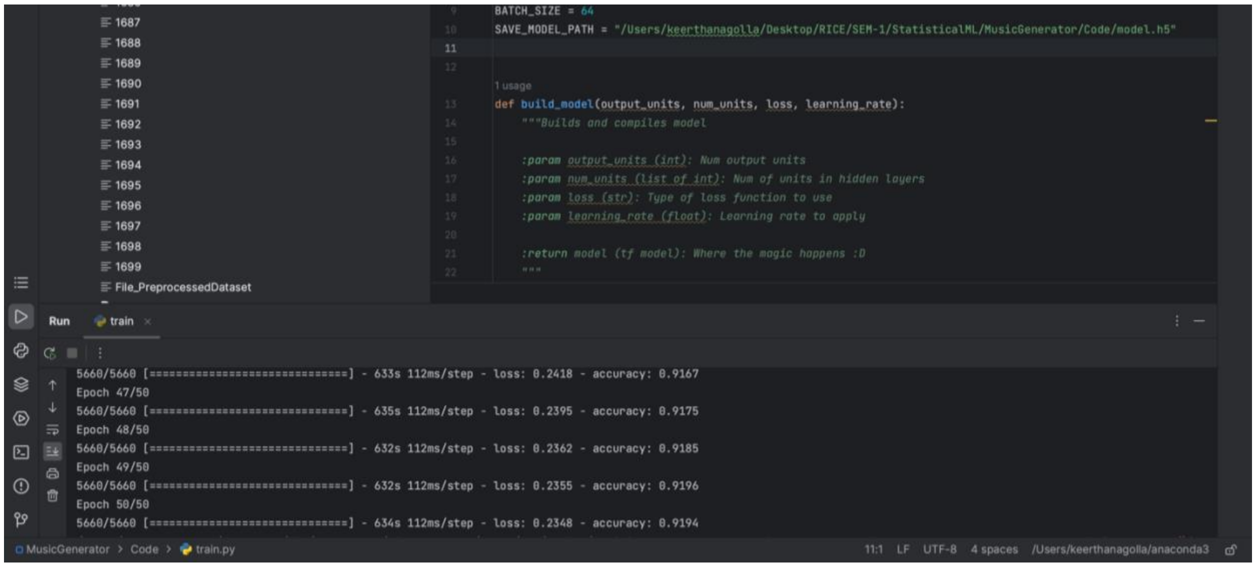
## Experiments



```
    :param learning_rate (float): Learning rate to apply

    :return model (tf model): Where the magic happens :D
    """

    # create the model architecture
    input = keras.layers.Input(shape=(None, output_units))
    x = keras.layers.LSTM(num_units[0])(input)
    x = keras.layers.Dropout(0.2)(x)

    output = keras.layers.Dense(output_units, activation="softmax")(x)

    model = keras.Model( *args: input, output)

    # compile model
    model.compile(loss=loss,
                  optimizer=keras.optimizers.Adam(learning_rate=learning_rate),
                  metrics=["accuracy"])

    model.summary()

    return model
```

```
=================================================================
Total params: 311846 (1.19 MB)
Trainable params: 311846 (1.19 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Epoch 1/2
5660/5660 [==============================] - 675s 119ms/step - loss: 0.6996 - accuracy: 0.7853
Epoch 2/2
5660/5660 [==============================] - 706s 125ms/step - loss: 0.5889 - accuracy: 0.8091
```

Fig 7. Accuracy and Loss can be seen after second epoch.


1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
File_PreprocessedDataset
Run
train
MusicGenerator > Code > train.py
11:1 LF UTF-8 4 spaces /Users/keerthanagolla/anaconda3

```python
BATCH_SIZE = 64
SAVE_MODEL_PATH = "/Users/keerthanagolla/Desktop/RICE/SEM-1/StatisticalML/MusicGenerator/Code/model.h5"


1 usage
def build_model(output_units, num_units, loss, learning_rate):
    """Builds and compiles model

    :param output_units (int): Num output units
    :param num_units (list of int): Num of units in hidden layers
    :param loss (str): Type of loss function to use
    :param learning_rate (float): Learning rate to apply

    :return model (tf model): Where the magic happens :D
    """
```

```
5660/5660 [==============================] - 633s 112ms/step - loss: 0.2418 - accuracy: 0.9167
Epoch 47/50
5660/5660 [==============================] - 635s 112ms/step - loss: 0.2395 - accuracy: 0.9175
Epoch 48/50
5660/5660 [==============================] - 632s 112ms/step - loss: 0.2362 - accuracy: 0.9185
Epoch 49/50
5660/5660 [==============================] - 632s 112ms/step - loss: 0.2355 - accuracy: 0.9196
Epoch 50/50
5660/5660 [==============================] - 634s 112ms/step - loss: 0.2348 - accuracy: 0.9194
```

Fig 8. Accuracy and Loss can be seen after fiftieth epoch.

I have uploaded respective generated music in drive:
LSTM 5 epochs - here
LSTM 50 epochs - here
Transformers 5 epochs - here
Code repository - here

## Conclusion

In conclusion, our project has successfully navigated the intricate interplay of preprocessing, model training, and the novel integration of BERT embeddings to unveil the latent complexities within musical compositions. The Melody Generator class stands as a tangible manifestation of our research's practical implications, showcasing the synergy between deep learning and music composition. By meticulously encoding musical structures and leveraging the capabilities of advanced neural networks, we have demonstrated the model's proficiency in capturing and generating intricate musical patterns.

Looking forward, the project paves the way for exciting future explorations. Fine-tuning hyperparameters, experimenting with diverse datasets, and employing more sophisticated sampling techniques present avenues for refining the melody generation process. Moreover, our foray into the intersection of the music and video fields opens intriguing possibilities. The potential to generate music from dance videos represents just one captivating direction for future endeavors. Ultimately, our project not only contributes to the evolving landscape of AI-driven music composition but also sparks curiosity about the untapped potential at the crossroads of deep learning and the arts.

## References

[1] Paper on MIDI Format for Music Generation, at https://arxiv.org/pdf/1908.01080.pdf
[2] Practical Insights into Music Generation with LSTM Neural Networks and Keras, at https://towardsdatascience.com/how-to-generate-music-using-a-lstm-neural-network-in-keras-68786834d4c5
[3] Music Generation with LSTMs, 'https://blog.paperspace.com/music-generation-with-lstms/'
[4] GANs for Music Generation, 'https://ieeexplore.ieee.org/document/9581146'
[5] Steve Hiehn, "How to Generate Music with Python: The Basics," Medium, 'https://medium.com/@stevehiehn/how-to-generate-music-with-python-the-basics-62e8ea9b99a5'
[6] Humdrum, "Understanding Music Representation," at https://www.humdrum.org/Humdrum/commands/midi.html.
[7] Valerio Velardo, "The Sound of AI," at https://thesoundofai.com/
[8] arXiv, at https://arxiv.org/abs/2203.12105
[9] Karnika Kapoor, "Music Generation LSTM," Kaggle, at https://www.kaggle.com/code/karnikakapoor/music-generation-lstm#EVALUATING-MODELS
These references provide valuable insights into mapping music symbols [5], understanding music representation [6], gaining technical terms and in-depth knowledge of music [7], exploring recent advancements in music generation [8], and following steps for music generation using RNN and LSTM [9].