Music Composition using AI and Big Data Tools

Mohsin Ali Mohammed,[1] **Shravani Hariprasad,** [1] **Pegah Ojaghi,**[2]

San Diego State University, USA[1]

Abstract

This project explores the use of Big Data and Deep Learning to create new compositions of music. For this project, we used using Google Cloud Platform (GCP), Pyspark, and Tensorflow as initial tools and frameworks. Using the MusicNet Dataset, we have trained different RNN and LSTM architectures that input tokenized music files and output predictions for notes(generation of new music). Using Machine Learning for music is a relatively new area of study, our project provides an investigation and some methods for creating an effective model for the music industry.

Introduction

Machine learning has found widespread use in a wide range of domains, including image identification, processing natural language, and speech recognition, just to mention a few. However, machine learning applications in the music business are still in their infancy. Music proves to be a different challenge compared to images, among three main dimensions: Firstly, music is temporal, with a hierarchical structure with dependencies across time. Secondly, music consists of multiple instruments that are interdependent and unfold across time. Thirdly, music is grouped into chords, arpeggios, and melodies — hence each time step may have multiple outputs.

However, audio data has several properties that make them familiar in some ways to what is conventionally studied in deep learning (computer vision and natural language processing, or NLP). The sequential nature of the music reminds us of NLP, in which we can use Recurrent Neural Networks.

In this project, we investigate the use of big data and deep learning to produce new musical compositions. Employing the MusicNet dataset, we train several recurrent neural networks (RNN) and long short-term recall (LSTM) designs. We used Google Cloud Platform (GCP), Pyspark, and Tensorflow as our key resources in order to achieve our goal. We used GCP computational resources, like GCP Colab and Clustering, to build a neural network model capable of producing high-quality music compositions. The MusicNet dataset contains professional performers' recordings of classical music performances, making it a valuable resource for our project. The goal of this project is to make a contribution to the field of machine learning for music composition by providing knowledge about the aspects that have the biggest effect on the performance of neural network models in producing high-quality music. We wanted to see how different aspects, such as neural network design, training data amount, and training length, affected the quality of the obtained results.

The remaining sections of this report are structured as follows:

1. Review of the Literature: In this section, we provide an overview of pertinent research on machine learning and deep learning applications in music creation. We talk about important ideas, approaches, and developments in the area, stressing their contributions to the subject.

2. Problem Statement: The issue that this project is trying to solve is clearly stated in the problem statement. We describe the aims and objectives of our research, outlining the difficulties and possibilities in the context of musical creation utilizing machine learning methods..

3. Description of the data set and exploratory data analysis (EDA): We give a thorough explanation of the MusicNet dataset that was used in this investigation. We talk about its features, dimensions, and the data it holds. In addition, we conduct exploratory data analysis to learn more about the dataset, examine its structure, and spot any patterns or trends.

4. Methodology: The process is described in this part, along with any cleaning, modification, or feature engineering done on the dataset. We next dig into the many modeling techniques used, explaining their structures and setups, including LSTM (Long Short-Term Memory) and GANs (Generative Adversarial Networks). The training procedure is also covered in detail, along with the application of particular optimization methods and callbacks.

5. Results and Discussion: Here, we share the findings from our project, together with model evaluations and performance indicators. We go over the conclusions, explain them, and evaluate the benefits and drawbacks of each strategy. We consider the consequences of our findings in the context of musical creation and develop ideas for additional study.

6. Conclusion: We provide a summary of our study's major conclusions and contributions. We draw attention to the importance of our findings in furthering the study of machine learning in musical composition. We also recommend possibilities for future research, including prospective areas for investigation and development for more sophisticated and creative music creation models.

By organizing the report in this manner, we provide a comprehensive understanding of the project, from the theoretical background to the practical implementation and evaluation. The structure ensures clarity, coherence, and a logical flow of information, allowing readers to grasp the context, methodology, and significance of our study.

Problem Statement

We aim to create a neural network architecture to generate new music using the MusicNet dataset. Also, investigate the utilization of Google Cloud Platform computing resources, specifically BigQuery and Clustering, to create a neural network model capable of generating high-quality music compositions. We will analyze the impact of various factors, including neural network architecture, training data size, and training duration, on the quality of the generated music compositions. The motive of the project is to contribute to the field of machine learning with the help of GCP for music composition and provide insights into the factors that most strongly impact the effectiveness of neural network models in generating high-quality music.

Dataset Description

MusicNet is a large-scale public dataset of classical music, which was created to support research in music information retrieval (MIR) and machine learning.

The dataset contains high-quality recordings of 330 classical music pieces, totaling approximately 25 hours of music, along with corresponding metadata such as composer, ensemble, and genre as shown in figure 1. The recordings were performed by professional musicians and were sourced from various public-domain recordings, commercial CDs, and live performances. The pieces were selected to represent a diverse range of classical music genres and styles, including chamber music, symphonies, and operas. In addition

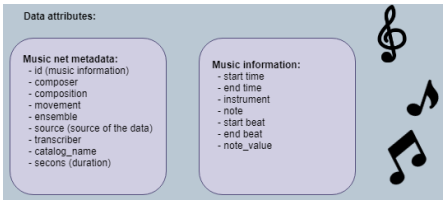

Figure 1: Data Attributes-MusicNet dataset attributes and their music information

to the audio recordings, the MusicNet dataset also includes preprocessed data in the form of annotated musical scores, which provide information on the musical structure, such as the notes played, their durations, and their positions in time. This makes the dataset useful for tasks such as music transcription, analysis, and generation.

The MusicNet dataset has been widely used in MIR research, including studies on automatic music transcription, structure analysis, and generation. This project is more like a music generation, so, this dataset is the best available for the generation task.

Exploratory Data Analysis (EDA)

EDA is vital for music generation as it enables a comprehensive understanding of the music data, identifying patterns and structures that inform the design of music generation algorithms. It aids in preprocessing and cleaning the data, addressing quality issues and anomalies. EDA also guides feature engineering, extracting meaningful musical attributes for input features. Moreover, it assists in model selection and evaluation, comparing and assessing the generated music's coherence, diversity, and quality. Overall, EDA enhances the music generation process by ensuring alignment with desired musical characteristics and improving the overall output.

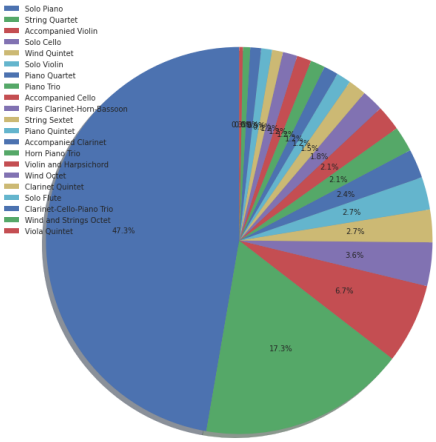

Figure 2: Ensemble - We can see that Solo piano is the most common type of ensemble (47.3%) followed by the string quartet (17.3%) and accompanied violin (6.7%).

The exploratory data analysis (EDA) reveals interesting insights about the dataset. Figure 2 indicates that the most common type of ensemble in the dataset is solo piano, accounting for 47.3% of the compositions. This is followed by the string quartet at 17.3% and accompanied violin at 6.7%. These findings suggest that solo piano compositions dominate the dataset.

Figure 3 highlights the distribution of compositions by different composers. It shows that approximately half of the music in the dataset is produced by Beethoven, with Bach and Schubert accounting for 20% and 9% respectively. This suggests that Beethoven plays a significant role in the dataset and could be a key composer to consider in our model.

Furthermore, figure 4 provides insights into the number of compositions on the violin by different composers. It reveals that Beethoven has the highest number of compositions for the violin compared to other composers. This information

can be valuable in designing our model and selecting appropriate compositions for training and analysis.

Overall, the EDA results provide a clear understanding of the distribution of ensemble types, the prominence of certain composers like Beethoven, and the prevalence of violin compositions in the dataset. These findings will guide our modeling decisions and help us focus on relevant aspects of the data for music generation and analysis.

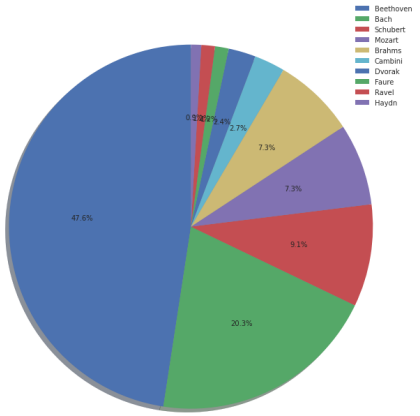

Figure 3: Composer - Almost half of the music was produced by Beethoven. Bach composed 20% and Schubert composed 9%.

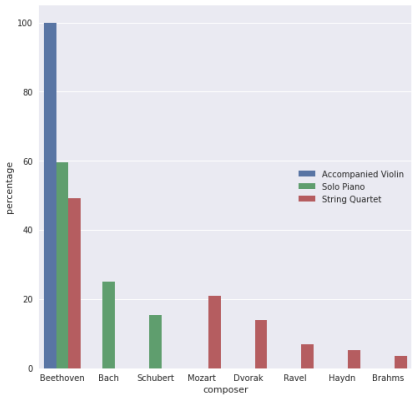

Figure 4: Compositions - Graph depicting composers and their composing in Accompanied Violin, Solo Piano, and String Quartet.

Methodology

Data preprocessing

Our first task as a group is to clean up the data. Data cleaning is an important stage in the data preparation process that involves discovering and repairing mistakes, inconsistencies, and missing data in the dataset. Data cleaning for MusicNet MIDI recordings may include deleting duplicated notes, adjusting note velocities, and fixing timing difficulties. It may also include identifying and deleting any unnecessary

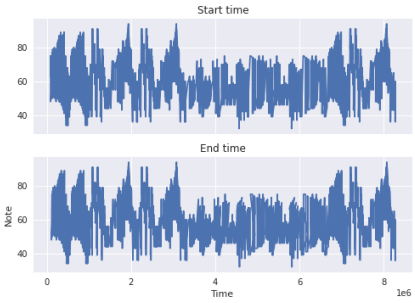

Figure 5: Note - Plot depicting the note evolution over start and end time

or noisy data that may have a bearing on the analysis's or machine learning models' accuracy. Data cleaning is a vital procedure that assures the quality of the data before doing any additional analysis or modeling business operations. We may prevent possible concerns such as biased analysis, erroneous forecasts, and lost time and money by cleansing the information on faulty models. Overall, we emphasize the significance of data cleaning as an essential step in preparing the MusicNet MIDI recordings for future analysis and modeling.

1. Key Extraction: The get_key(s) function is responsible for extracting the key from a given string "s", which represents a MIDI event. It checks if the string contains the word "key" and retrieves the key substring from specified positions. It also handles cases where the key ends with "m" or a single quote, removing the last character if necessary. This key extraction is essential for capturing the tonality of the music.

2. Note Extraction: The parse_notes(track) function processes MIDI events in the given "track" to extract note-related information. It iterates over each event and distinguishes between meta events (such as key changes) and note events (note on/off events). For meta events, the function calls "get_key(s)" to extract the key information from the event string. If a new key is found, it updates the "key" variable accordingly. For note events, it checks if the event represents a note-on event with a non-zero velocity and duration. If so, it creates a dictionary ("note_dict") containing the note, duration, velocity, and channel information. When a note-off event is encountered or a note-on event with zero velocity is detected, it adds the note dictionary to a new tune and resets the "note_dict". This process ensures that the relevant note-related data is captured for further analysis and model training.

Phrase Selection and Feature Extraction: The next step in the code involves selecting suitable musical phrases and extracting relevant features from those phrases for model training.

1. Phrase Selection: The "various(notes)" function is used to determine if a sequence of eight consecutive notes contains at least three unique notes. This function helps filter out phrases that are not musically diverse enough. By iterating over the notes, the function checks if the number of

unique notes in the eight-note sequence is greater than two. If this condition is satisfied, the phrase is considered suitable for inclusion in the training dataset.

2. Feature Extraction: Once suitable phrases are identified, the code prepares the input-output pairs (‘X‘ and ‘y‘) for training the model. Each input sequence (‘X‘) consists of note, duration, and velocity data for a specific phrase, while the corresponding output sequence (‘y‘) represents the note, duration, and velocity of the next note in the sequence.

The code loops over the tunes and extracts phrases of length defined by the ”phrase_len” variable (in this case, 60). It checks the diversity of notes within the phrase using the ‘various(notes)‘ function. If the diversity condition is met, the note, duration, and velocity data for the phrase are extracted and added to the ‘X‘ list, while the next note information is added to the ‘y‘ list.

By following these steps, the code performs data preprocessing, selects musically diverse phrases, and extracts relevant features, forming the foundation for training a model for music composition.

Modeling

Recurrent Neural Networks (RNNs) have proven to be effective in music compositions due to their ability to model temporal dependencies and capture the sequential nature of music. We are using RNNs like LSTM because of their long and short-term capabilities and below are some important reasons for us to choose these models.

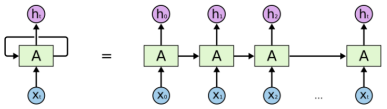

Figure 6: LSTM - Architecture of LSTM

1. Temporal Modeling: Music is inherently a time-dependent art form, where the sequence of notes and their timing are crucial. LSTMs are designed to handle sequential data, making them suitable for capturing temporal patterns in music compositions. It can learn and generate musical sequences that exhibit musical coherence and flow.

2. Long-term Dependencies: LSTMs are capable of learning and capturing long-term dependencies in music compositions. They can remember and utilize information from earlier notes in the sequence to influence the generation of subsequent notes. This capability is particularly valuable in capturing the nuanced relationships and context between different musical elements.

3. Generation of Musical Structures: LSTM can generate musical structures, such as melodies or harmonies, that follow established musical conventions. By training these models on a large dataset of music compositions, the model can learn patterns and structures present in the training data, enabling it to generate new compositions that adhere to similar musical conventions.

Overall, LSTMs offer a powerful framework for modeling music compositions, capturing temporal dependencies, generating coherent musical sequences, and fostering creativity and novelty. Their ability to learn from large music datasets and generate new compositions makes them a valuable tool for composers, music producers, and researchers in the field of music composition.

We have trained 5 different models as follows

1. Plain LSTM

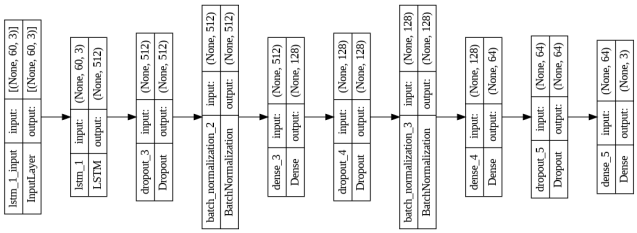

Figure 7: Plain LSTM - Architecture of Model 1 with Plain LSTM

The LSTM-based sequential model is constructed using TensorFlow. The model consists of an LSTM layer with 512 units, followed by a dropout layer to prevent overfitting. The output of the dropout layer is passed through a batch normalization layer to improve regularization. The model then includes additional dense layers with ReLU activation and dropout layers for further feature extraction and non-linearity. Finally, a dense layer with 3 units and ReLU activation is added to produce the output representing the **note, duration, and velocity**

2. LSTM with Embedding

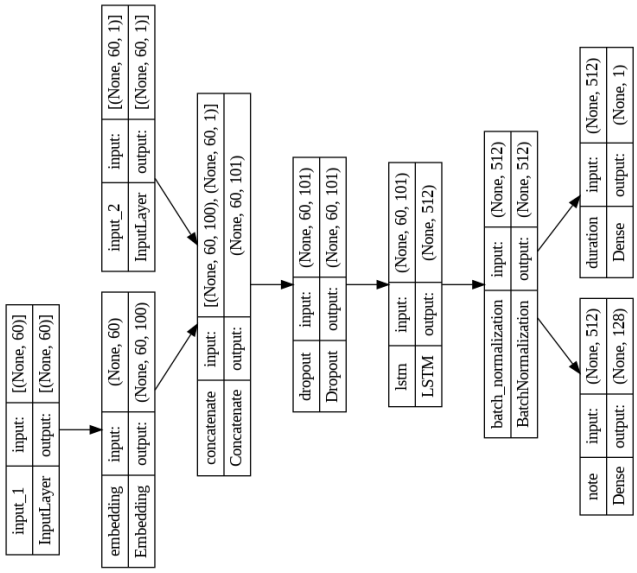

Figure 8: Embedded LSTM - Architecture of Model 2 with Embedded LSTM

The architecture starts with two input layers, notes_in, and durations_in, corresponding to the chord notes and durations respectively. The chord notes input is embedded using the Embedding layer, which converts the categorical data into dense vectors of size embed_size (100 in this case).

The embedded chord notes and durations input are then concatenated using the Concatenate layer. A dropout layer is applied to prevent overfitting, followed by an LSTM layer with 512 units to capture the sequential patterns in the data. Batch normalization is applied to normalize the output of the LSTM layer.

Output Layers: The concatenated model is then passed through two separate dense layers. The first dense layer (notes_out) uses a softmax activation function to output probabilities for each possible chord note (128 in this case). The second dense layer (durations_out) uses a ReLU activation function to predict the duration values.

Model Compilation: The model is compiled using the sparse_categorical_crossentropy loss function for chord notes and the mean squared error (MSE) loss function for durations. The RMSprop optimizer is used with a learning rate of 0.001.

The model is trained using the fit() function of the embed_model. The input arrays (train_chords, train_durations) and target arrays (target_chords, target_durations) are provided as inputs. The training is performed over 500 epochs with a batch size of 256. A validation split of 0.2 is used for monitoring the model's performance on unseen data. Additionally, callbacks such as TensorBoard logging, learning rate schedule, and reducing learning rate on the plateau are employed for enhanced training control of the model.

3. Generative Adversarial Networks(GAN)

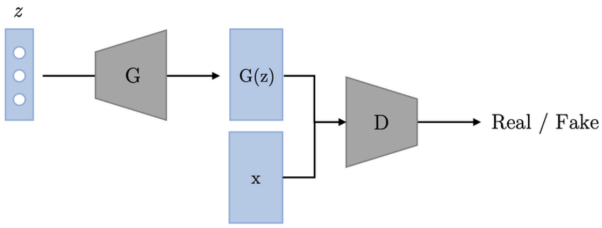

Figure 9: GAN

GANs are used in music generation to overcome the challenges of generating creative and high-quality music compositions. GANs can learn the underlying patterns and structures of musical data, allowing them to generate new and unique compositions that mimic the style and characteristics of the training dataset. By leveraging the adversarial framework, GANs enable the generator network to improve over time by competing against the discriminator network, resulting in the generation of more realistic and diverse music compositions.

The GAN consists of a generator and a discriminator, and it is trained to generate music tunes.

Data Preparation: - The variable 'train_matrixes' is a list that stores the training data matrixes. Each matrix in the list represents a music tune, with a shape of '(n_notes, tune_len)'.

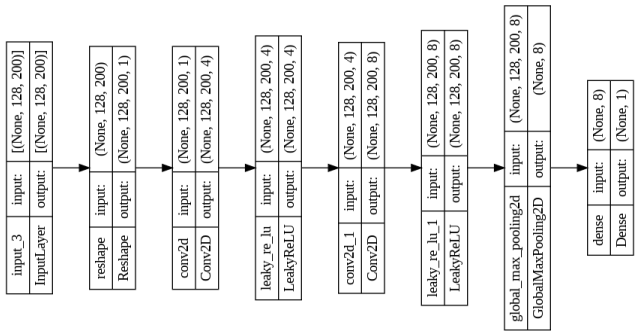

Figure 10: Discriminator

Discriminator Architecture: The discriminator performs the following actions on the input shape '(n_notes, tune_len)':

- Using the "Reshape" layer, reshapes the input into a 4D tensor using the parameters '(n_notes, tune_len, 1)'
- Runs the input through two convolutional layers ('Conv2D' and 'LeakyReLU'). with leaky ReLU activation functions.
- Uses global max pooling to cut down on the dimensions of space.
- Establishes a connection with a dense layer that uses sigmoid activation to produce an output probability indicating whether the input is genuine or not.

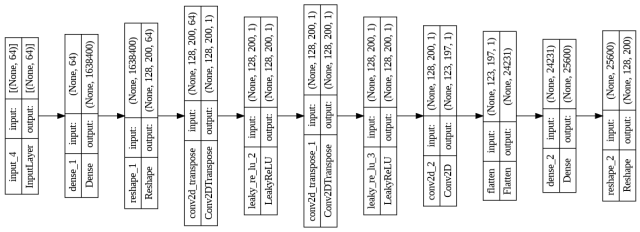

Figure 11: Generator

Generator Architecture:

- A latent vector of length 'latent_dim'is provided as input to the generator.
- It uses a number of operations to create a musical composition:
- Connects to a thick layer, increasing the latent vector's dimensionality.
- Reconfigures the output as a 4D tensor with the values '(n_notes, tune_len, latent_dim)'.
- Upsamples the data using two transpose convolutional layers with leaky ReLU activation functions ('Conv2DTranspose' and 'LeakyReLU').
- Uses a convolutional layer connection to change the number of channels.
- In order to create the created music tune, flattens the output and connects to a dense layer with sigmoid activation.

GAN Training: The GAN class is defined, which inherits from TensorFlow.

- The GAN is compiled with optimizers for the discriminator and generator, as well as loss functions for the discriminator and generator.
- The 'train_step' method is overridden to define the training process:
- Random latent vectors are sampled.
- The generator generates fake tunes using the random latent vectors.
- The discriminator is trained to distinguish between real and fake tunes by computing the discriminator loss.
- The generator is trained to fool the discriminator by generating fake tunes and computing the generator loss.
- The GAN is trained using the 'fit' method, where the dataset (a subset of 'train_matrixes') is passed as input, and the training is performed over 60 epochs.

Fine-tuning

LSTM models are compiled with the mean absolute error (MAE) loss function and the Adam optimizer. During training, the input data (X) and target values (y) are fed to the model in batches. The training process is carried out over multiple epochs, with a batch size of 256. A validation split of 0.2 is used to monitor the model's performance on unseen data. Additionally, callbacks such as TensorBoard, LearningRateScheduler, and ReduceLROnPlateau are employed for tracking training progress, dynamically adjusting the learning rate, and reducing the learning rate when the validation loss plateaus.

Callbacks

Callbacks play a crucial role in controlling the training process of deep learning models. In our training process, we utilized several important callbacks to enhance the training performance and monitor the model's progress.

The first callback we employed is the learning rate scheduler. This callback exponentially reduces the learning rate after every 10 epochs. By gradually decreasing the learning rate, we enable the model to converge more effectively and find a better optima.

The second callback is ReduceLROnPlateau. It monitors the validation loss and reduces the learning rate by 10% if the validation loss does not improve for 2 consecutive epochs. This adaptive learning rate adjustment helps the model overcome plateaus and make finer adjustments as it approaches convergence.

The third callback, ModelCheckpoint, is utilized to save the model whenever the validation loss decreases. This allows us to keep track of the best-performing model during the training process and avoid losing progress in case of unexpected interruptions.

Lastly, we utilized the TensorBoard callback, which provides a visual representation of the model's loss and weights over time. This visualization aids in monitoring the model's performance and identifying patterns or trends during training.

By leveraging these callbacks, we can optimize the training process, dynamically adjust the learning rate, save the best model, and gain valuable insights through visualizations. This comprehensive callback strategy enhances the efficiency, stability, and understanding of our models during training.

Google Cloud Computing(GCP)

1. GCP Data Buckets Musicnet is a popular dataset used for music-related research and machine learning applications. It contains a vast collection of classical music recordings, including performances by various artists and orchestras. Storing Musicnet data in a GCP bucket offers several advantages. Firstly, GCS provides durability and redundancy, ensuring that your data is safe and available at all times. GCS automatically replicates data across multiple regions, making it highly resilient to failures and data loss.

Additionally, GCS allows seamless integration with other GCP services, such as BigQuery, Cloud Machine Learning Engine, and Dataflow, enabling efficient data processing and analysis workflows. With GCS, you can easily transfer Musicnet data to these services for tasks like data preprocessing, model training, and generating insights from the music dataset.

2. GCP Dataprocs

Cloud by Google The Google Cloud Platform (GCP) managed service called Dataproc makes and speeds up the processing of huge data workloads. For the purpose of conducting Apache Spark and Hadoop workloads, it enables users to quickly construct and maintain clusters. Users can set up virtual machine clusters using Dataproc, scale them as necessary, and take advantage of automatic scaling features to maximize resource usage. Users are able to take advantage of the power of Google Cloud's storage, analytics, and machine learning capabilities thanks to Dataproc's easy integration with other GCP services. With the flexibility and scalability of the cloud, it offers a cost-effective and efficient alternative for processing large-scale data tasks.

3. GCP Notebooks

The Google Cloud Platform (GCP) Notebook is a web-based platform for interactive data exploration, analysis, and development that is offered by Google Cloud. Users can share their work with others, write and run code, and visualize data in a collaborative workplace. Multiple programming languages, including Python, R, and Julia, are supported by GCP Notebook, which also comes with pre-installed libraries and frameworks for basic data science and machine learning applications.

Deployment

FastAPI is a Python framework for building APIs, known for its high performance and ease of use. When combined with GCP, it allows developers to deploy and scale LSTM models for tasks like music composition. LSTM (Long Short-Term Memory) is a type of recurrent neural network commonly used in sequential data analysis. By leveraging the power of GCP's infrastructure, developers can efficiently train and deploy LSTM models using FastAPI, enabling them to create

sophisticated applications that generate music compositions based on learned patterns and structures

Results

A close alignment between the training and validation loss suggests that the model is well-optimized, and its performance is reliable. It provides confidence in the model's ability to perform accurately on unseen data and indicates that the training process has been successful in achieving a balance between learning from the training data and generalizing to new instances.

Table 1 summarizes the results of different neural network model experiments on the MusicNet dataset. The models evaluated include LSTM, BiLSTM, and GAN (Generative Adversarial Network). The table presents the train loss, train accuracy, test loss, and test accuracy for each model.

From the results, it is observed that when the models are trained with embeddings, specifically the LSTM with Embedding, they achieve better performance compared to the models without embeddings. The LSTM model with embeddings achieves a lower train loss of 3.31 and higher train accuracy of 81%, indicating effective learning and capturing patterns in the music data. Similarly, the test loss of 3.1 and test accuracy of 83% further validate the model's performance on unseen data.

Comparing the different models, it can be concluded that the LSTM model with Embedding is the most effective approach for training music generation models on the MusicNet dataset. It demonstrates better performance in terms of both train and test metrics, indicating its capability to generate music sequences with high accuracy and low loss.

These results suggest that utilizing LSTM models with embeddings can be a promising strategy for training music generation models and provide insights into improving the quality and accuracy of generated music. Further experimentation and fine-tuning of the models could lead to even better results and advancements in the field of music generation

Models	Embedding	Train Loss	Train Accuracy	Test Loss	Test Accuracy
LSTM	✗	5.6	71	5.9	56
LSTM	✓	3.31	81	3.1	83
BiLSTM	✗	5.3	65	5.5	63
BiLSTM	✓	3.8	79	3.92	80
GAN	✗	4.1	71	4.7	67

Table 1: Results of various Neural Network model experiments on MusicNet Data. It is evident from the results that LSTM with Embedding could be an effective way to train Music generation models.

Discussion

In this project, we explored various topics related to music composition, deep learning, and cloud computing on Google Cloud Platform (GCP). We dused of recurrent neural networks (RNNs) like LSTM for music composition and how they can generate music based on learned patterns. We also looked at data preprocessing techniques, phrase selection, and feature extraction in the context of music composition.

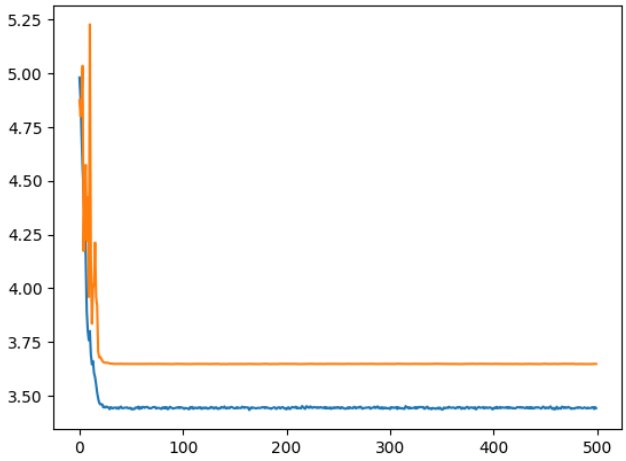

Figure 12: Loss of the best-performed model. The loss of the best-performing model indicates the effectiveness of the training process. In our case, the plot demonstrates that the model is under control as there is minimal difference between the training and validation loss. This suggests that the model has achieved a good balance between generalization and fitting the training data

We then delved into the practical aspects of working with GCP, including storing Musicnet data in GCP buckets, utilizing GCP Dataproc for distributed data processing, and leveraging GCP Notebooks for development and experimentation. We also explored the deployment of LSTM models using FastAPI on GCP, enabling the creation of scalable and accessible music composition APIs.

Overall, in this project we showcased the intersection of music composition, deep learning, and cloud computing, highlighting the potential of these technologies to revolutionize the creation and generation of music.

Elephas is a system that combines Spark with Keras to enable distributed deep learning on massive datasets. It attempts to keep Keras' ease-of-use and simplicity, enabling quick model prototyping and training on large amounts of data.

Elephas did not match our goals, while having the benefit of distributed computing capabilities through Spark. We decided to use TensorFlow as a substitute as a result. A popular deep learning framework called TensorFlow is renowned for its adaptability, scalability, and broad community support.

By switching to TensorFlow, we most certainly gained compatibility with a wider range of features and resources as well as access to more resources. TensorFlow is a well-liked option in the industry because it offers powerful tools for creating, honing, and deploying deep learning models.

The choice to transition to TensorFlow ultimately represented the need for a framework that better matched our unique requirements and provided the essential capabilities and support to help us achieve our objectives for developing and deploying deep learning models.

Limitations

While our project aims to explore and develop a neural network architecture for music generation using the MusicNet dataset and leverage the capabilities of Google Cloud Platform (GCP), there are several limitations to consider:

Data Availability: The quality and quantity of the MusicNet dataset may impose limitations on the diversity and complexity of the generated music. The dataset's coverage of different music genres, styles, and compositions may not be exhaustive, potentially impacting the creativity and variety of the generated music.

Subjective Evaluation: Assessing the quality and artistic value of the generated music is a subjective process. While we can employ metrics and evaluation methods, determining the true musicality and appeal of the compositions is challenging. Evaluating the generated music solely based on quantitative metrics may not capture the full essence of musical creativity.

Computational Resources: Generating music using neural network models can be computationally intensive, especially for large-scale datasets and complex architectures. Limited computational resources or time constraints may restrict the size of the dataset used for training or the complexity of the neural network models employed, potentially affecting the overall performance and quality of the generated music.

Generalization and Originality: Ensuring that the generated music is original and exhibits a high level of generalization beyond the training dataset is a challenging task. Neural network models may struggle to capture the intricacies and nuances of human composition, leading to potential biases or limitations in the generated music.

Legal and Copyright Issues: When using music datasets, it is crucial to consider legal and copyright implications. Generating music that closely resembles existing compositions or violates copyright restrictions can lead to legal issues. Care must be taken to ensure that the generated music is original and does not infringe upon any intellectual property rights.

User Preferences and Creativity: Music is a highly personal and subjective art form, and user preferences vary significantly. The generated music may not resonate with all listeners or meet individual expectations of creativity and musicality. It is important to acknowledge that the generated music may not please everyone's taste or fulfill the expectations of all users.

Despite these limitations, our project aims to explore the potential of neural networks, GCP, and the MusicNet dataset in music generation, providing valuable insights and paving the way for future advancements in the field.

Conclusion

In conclusion, our exploration of music generation using deep learning and GCP revealed several key insights. We analyzed the MusicNet dataset, which provided valuable information about the distribution of ensembles and compositions across different composers. Solo piano emerged as the most common ensemble type, while Beethoven stood out as the dominant composer.

We leveraged the power of LSTM and GAN models for music generation, training them using GCP services such as GCP Bucket for data storage, GCP Dataproc for distributed processing, GCP Notebooks for development, and FastAPI for deployment. These technologies offered efficient data handling, scalable computing capabilities, collaborative development environments, and streamlined deployment options.

We employed various callbacks in the training process, including learning rate scheduling, reducing learning rate on plateau, model checkpointing, and TensorBoard visualization. These callbacks played crucial roles in controlling the training process and monitoring model performance.

While we initially experimented with Elephas, an integration of Keras and Spark, we ultimately switched to TensorFlow for its flexibility, compatibility, and extensive community support. TensorFlow provided a robust framework for deep learning model development and allowed us to harness the full potential of GCP's infrastructure.

Overall, our journey through music generation using deep learning and GCP showcased the potential of these technologies in creating AI-powered music compositions. By combining cutting-edge machine learning algorithms, cloud computing resources, and efficient development and deployment tools, we unlocked new possibilities in music creation and expanded our understanding of the intersection between AI and music.

References

[1] Kaggle. (n.d.). MusicNet Dataset. Retrieved from https://www.kaggle.com/datasets/imsparsh/musicnet-dataset

[2] Analytics Steps. (n.d.). Music Composition using Deep Learning. Retrieved from https://www.analyticssteps.com/blogs/music-composition-using-deep-learning

[3] Microsoft. (2021). Tutorial on AI Music Composition. Retrieved from https://www.microsoft.com/en-us/research/uploads/prod/2021/10/Tutorial-on-AI-Music-Composition-@ACM-MM-2021.pdf

[4] Analytics Vidhya. (2020). How to Perform Automatic Music Generation? Retrieved from https://www.analyticsvidhya.com/blog/2020/01/how-to-perform-automatic-music-generation/

[5] Towards Data Science. (n.d.). Generating Music Using Deep Learning. Retrieved from https://towardsdatascience.com/generating-music-using-deep-learning-cb5843a9d55e

[6] Panos, G., Stasis, A., Vidakis, N., Katsikis, V., & Mporas, I. (2021). A Quantum Natural Language Processing Approach to Musical Intelligence. In *2021 International Conference on Computing, Networking and Communications (ICNC)* (pp. 362-367). IEEE.

[7] Shah, A., & Bhatt, N. (2021). Music Composition with Deep Learning: A Review. Retrieved from https://nime.pubpub.org/pub/7a6ij1ak/release/1

[8] NVIDIA Developer. (n.d.). Leveraging AI Music with NVIDIA DGX-2. Retrieved from https://developer.nvidia.com/blog/leveraging-ai-music-with-nvidia-dgx-2/

[9] Xyonix. (n.d.). How AI is Transforming Music Composition. Retrieved from https://www.xyonix.com/blog/how-ai-is-transforming-music-composition

[10] Cotterell, R., Ren, X., Brundage, M., & Socher, R. (2021). Towards AI Music Generation. In *Proceedings of the 29th ACM International Conference on Multimedia* (pp. 3523-3527).

[11] Ye, J., Qiu, C., Huang, X., He, Q., & Xu, J. (2022). AI-MuComposer: An AI-Driven Music Composition Framework. *Wireless Communications and Mobile Computing*, 2022, Article 8123671.

[12] Bhatia, A. (n.d.). AI's Growing Role in Musical Composition. Retrieved from https://medium.com/syncedreview/ais-growing-role-in-musical-composition-ec105417899

[13] Wikipedia. (n.d.). Music and Artificial Intelligence. Retrieved from https://en.wikipedia.org/wiki/Music_and_artificial_intelligence

[14] Huang, D. (2019). Artificial Intelligence and Musical Composition: A Review. Harvard University. Retrieved from https://dash.harvard.edu/bitstream/handle/1/42029468/HUANG-DISSERTATION-2019.pdf

[15] Colombo, A. (2022). Learning to Compose: Automatic Music Composition Using Machine Learning Techniques. Theses Hal. Retrieved from https://theses.hal.science/tel-03135082/document

[16] MusicTech. (n.d.). 8 Ways to Use ChatGPT for Music Making. Retrieved from https://musictech.com/news/gear/ways-to-use-chatgpt-for-music-making/

[17] Zyxware Technologies. (n.d.). 11 Amazing AI Tools for Everyone to Revolutionize Your Work. Retrieved from https://www.zyxware.com/article/6548/11-amazing-ai-tools-for-everyone-revolutionize-your-work

[18] DeepLearning.AI. (n.d.). Google Introduces an AI that Generates Music from Text. Retrieved from https://www.deeplearning.ai/the-batch/google-introduces-an-ai-that-generates-music-from-text/

[19] MarkTechPost. (2023). Best AI Music Generators in 2023. Retrieved from https://www.marktechpost.com/2023/01/31/best-ai-music-generators-in-2023/

[20] TensorFlow. (n.d.). Music Generation with Magenta. Retrieved from https://www.tensorflow.org/tutorials/audio/music_generation