Music Generation using Neural Nets

Students Andi Vranceanu, Bicu Andrei,
Cojocariu Alexandru, Maxim Datco, Cretu Bogdan

Coordinated by Prof. PhD. Adrian Iftene and Cristian Simionescu

January 14, 2022

Contents

Abstract

In the recent period, Neural networks are used in more and more applications and services across a broader and broader spectrum of industries and domains.

One of the applications of Neural Networks is in artistic content generation, ranging from schematics and improving 3D rendering to AI-powered upscaling of videogames through DLSS, and Image and Music generation. In this project, we tried various music generation methods in order to see their limitations, unrefined results, and the most popular generation of audio files.

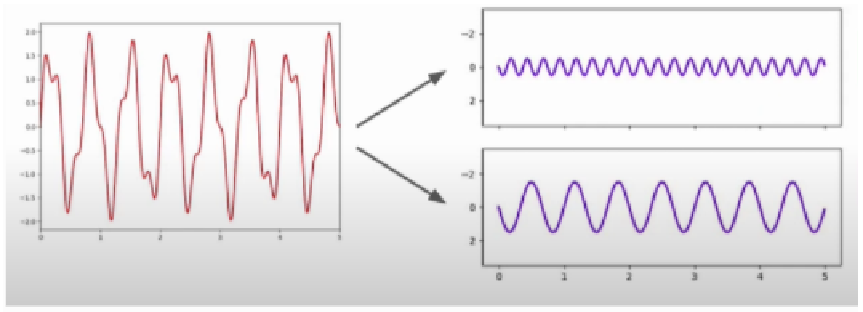
1 Introduction

In this work, multiple methods of music generations will be tested in order to see what methods lead to the best preliminary results and to gouge the complexity of the problem.

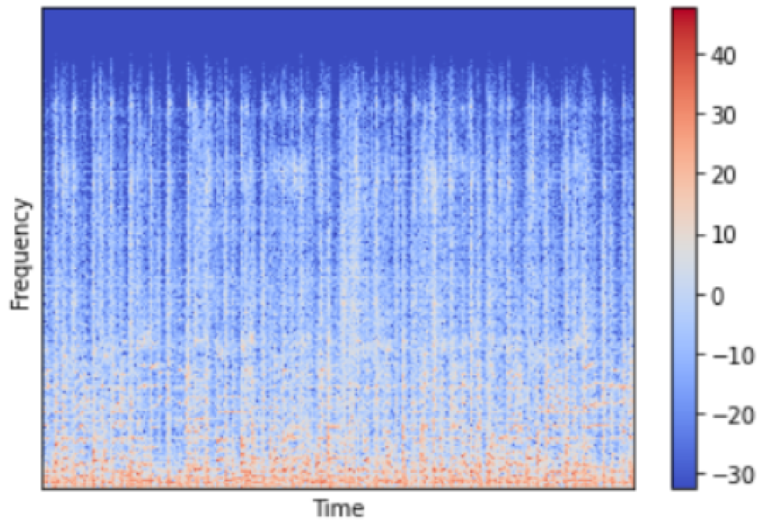
2 State-of-the-art

2.1 Audio processing

Audio signals can be represented in an efficient way using Fast Fourier transform(FFT). Each sound signal can be represented as a sum of uniform sound signals using FFT such as in the image below. This way, instead of working with an entire sound signal, we can work directly with a set of pairs of values [amplitude, frequency] which compound the original sound.



This information can be then organized in a spectogram which tells which frequency can be hard and how load at each point in time. There are 3 dimensions: time, frequency and decibels, where color is used for representing the 3rd dimension.



2.1.1 Existing models

Magenta — 2016

This project developed by Google Brain aims at creating a new tool for artists to use when working and developing new songs. They have developed several models to generate music. At the end of 2016, they published an LSTM model tuned with Reinforcement Learning. The interesting idea was to use Reinforcement Learning to teach the model to follow certain rules, while still allowing it to retain information learned from data. More recently, the Magenta team has used GAN and Transformers to generate music with improved long-term structure.

Wavenet — 2016

This is an example of a project using continuous encoding instead of discrete encoding. The model generates raw audio waveforms. Therefore, it is able to generate any kind of sound, like human voices. The model is a CNN, where the convolutional layers have multiple dilatation factors and predictions only depend on previous timesteps. Applications include the generation of piano pieces and speech generation.

MidiNet — 2017

This project uses a CNN for generating a series of Midi notes one bar after another. In addition to the generator, it uses a discriminator to learn the distributions of melodies, making it a generative adversarial network (GAN). The project was made by the Research Center for IT innovation at Academia Sinica, Taipei. The results of the project were comparable with the MelodyRNN models made by Google, in being realistic and pleasant to listen to, but The MidiNet's melodies are reported to be a lot more interesting.

MuseGAN — 2017

In this project, to cope with the grouping of notes, bars are used instead of notes as the basic compositional unit. Therefore, music is generated one bar after another using CNN, which are good for finding local, translation invariant patterns. An interesting approach in this paper is the evaluation metrics used.

MuseNet — 2019

This is OpenAI music generation model. It leverages state of the art NLP architecture — large-scale transformer model — to predict the next token in a sequence. It can combine styles from different famous composers as well as different music genres.

3 Proposed Solution

3.1 Architecture

We decided on a basic workflow for our application: For each chosen model we apply data augmentation and we train the model. For model usage, we run the model in the cloud with an exposed API for sending input data and retrieving output data.

We chose to use python for writing and testing our models as it is the most popular technology for neural network development.[1]

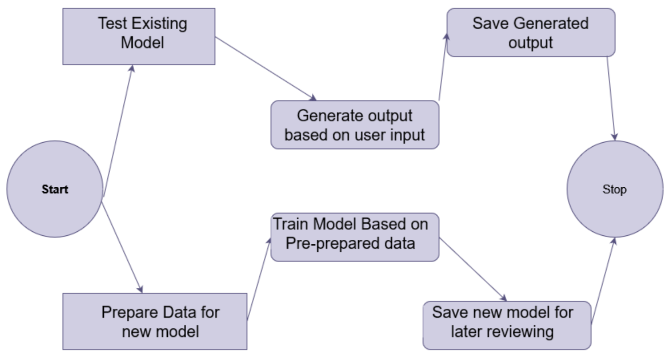

Test Existing Model
Save Generated output
Generate output based on user input
Start
Stop
Train Model Based on Pre-prepared data
Prepare Data for new model
Save new model for later reviewing

Figure 1: Basic Flows Planned for the final app

3.2 Cloud Infrastructure

We chose AWS as our cloud solution provider. To define our application we used the AWS Cloud Development Kit (AWS CDK). The AWS Cloud Development Kit (AWS CDK) is an open-source software development framework to define your cloud application resources using familiar programming languages. Provisioning cloud applications can be a challenging process that requires you to perform manual actions, write custom scripts, maintain templates, or learn domain-specific languages. AWS CDK uses the familiarity and expressive power of programming languages for modeling your applications. It provides high-level components called constructs that preconfigure cloud resources with proven defaults, so you can build cloud applications with ease. AWS CDK provisions your resources in a safe, repeatable manner through AWS CloudFormation. It also allows you to compose and share your own custom constructs incorporating your organization's requirements, helping you expedite new projects.

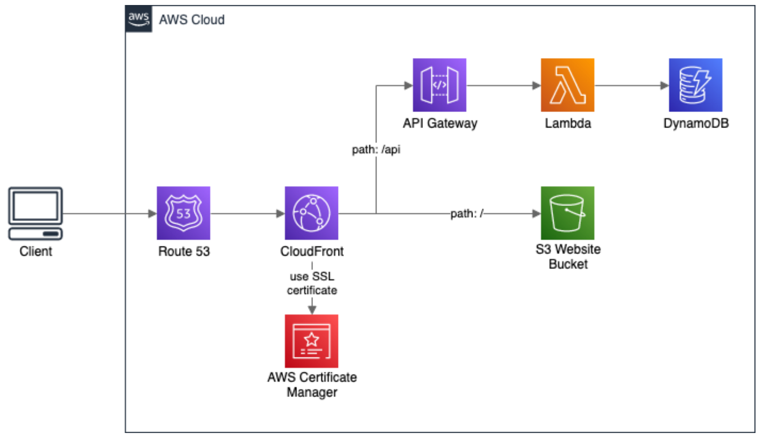


Figure 2: Cloud Infrastructure

The figure above shows the infrastructure we have defined. We used S3 to host our website. The website was written using the Angular framework.

Because we wanted our website to be easily accessible and high performance, we chose to use Amazon CloudFront. Amazon CloudFront is a content delivery network (CDN) service built for high performance, security, and developer convenience. Reach viewers across the globe in milliseconds with built-in data compression, edge compute capabilities, and field-level encryption.

Because we wanted secure communication between the server and the client, we chose to use AWS Certificate Manager. AWS Certificate Manager is a service that lets you easily provision, manage, and deploy public and private Secure Sockets Layer/Transport Layer Security (SSL/TLS) certificates for use with AWS services and your internal connected resources. SSL/TLS certificates are used to secure network communications and establish the identity of websites over the Internet as well as resources on private networks. AWS Certificate

Manager removes the time-consuming manual process of purchasing, uploading, and renewing SSL/TLS certificates.

To scale more easily we decided to expose all our functionalities through an API. That's why we chose to use the Amazon API Gateway. Amazon API Gateway is a fully managed service that makes it easy for developers to create, publish, maintain, monitor, and secure APIs at any scale. APIs act as the "front door" for applications to access data, business logic, or functionality from your backend services. Using API Gateway, you can create RESTful APIs and WebSocket APIs that enable real-time two-way communication applications. API Gateway supports containerized and serverless workloads, as well as web applications. API Gateway handles all the tasks involved in accepting and processing up to hundreds of thousands of concurrent API calls, including traffic management, CORS support, authorization and access control, throttling, monitoring, and API version management.

Each call in the API involves the call of a Lambda function that handles the request. AWS Lambda is a serverless, event-driven compute service that run our code for virtually any type of application or backend service without provisioning or managing servers. These functions deal with the interaction with the seeds, our models and the generated songs.

To save information about our seeds, patterns, and generated songs, we chose to use a non-relational database, AWS DynamoDB. In this database we save metadata, the actual resources are saved in AWS S3. Amazon DynamoDB is a fully managed, serverless, key-value NoSQL database designed to run high-performance applications at any scale. DynamoDB offers built-in security, continuous backups, automated multi-region replication, in-memory caching, and data export tools. Amazon Simple Storage Service (Amazon S3) is an object storage service offering industry-leading scalability, data availability, security, and performance.

3.3 Running a model in the Cloud

With the help of AWS CDK we have defined the training and detection jobs for the model, the storage spaces of the necessary resources for training and detection, an API to interact with the models and a web page to make the interaction with the user easier.

To save the data needed for training, the artifacts of the trained models and the new songs generated, we chose to use the S3 service.

For running training and detection jobs we chose to use AWS SageMaker. Because our models are custom we need to build a Docker image. This is the only way SageMaker can run our code. To save Docker images we chose to use AWS ECR. Amazon ECR is a fully managed container registry offering high-performance hosting, so we can reliably deploy application images and artifacts anywhere.

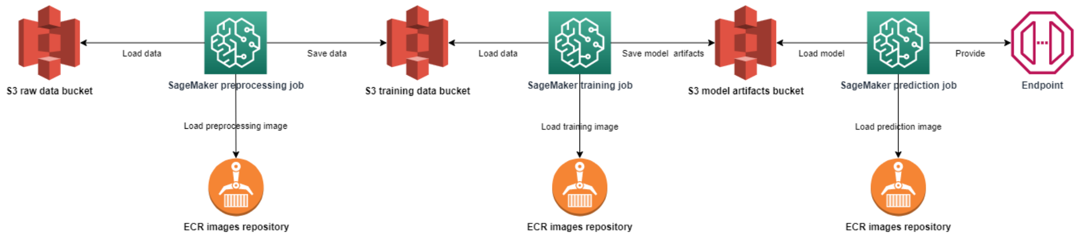


Figure 3: Jobs architecture

3.4 Getting Data

The internet is a repository of all kinds of data, the content available grows every year and is spread across countless formats and domains.

When it comes to music there are datasets intended for machine learning activities and research, some accessible ones being hosted on Kaggle.

"midi-classic-music" is a dataset containing over 3000 midi files of classical works by 175 composers including Bach, Beethoven, Mozart, Brahms, Chopin, Tchaikovsky, Strauss, Stravinski, Prokofiev, Rachmaninov, Bernstein, Bartok, Handel, Ravel, Scriabin, and others. It is available at the following link:

https://www.kaggle.com/blanderbuss/midi-classic-music

"Classical music MIDI" is a dataset containing dozens of classical piano composition from 19 famous composers, organized by composer. It is available at the following link:

https://www.kaggle.com/soumikrakshit/classical-music-midi

"MusicNet Dataset" is a big collection of 330 freely licensed classical music recordings with very detailed labeling indicating the precise time of each note in every recording, the instrument that plays each note, and the note's position in the metrical structure of the composition. The dataset is available at the following link:

https://www.kaggle.com/imsparsh/musicnet-dataset

3.5 Variational Autoencoders

VAE is an autoencoder whose encodings distribution is regularised during the training in order to ensure that its latent space has good properties allowing us to generate some new data. A variational autoencoder (VAE) provides a probabilistic manner for describing an observation in latent space.

Encoder network defines the approximate posterior distribution q(z|x), which takes as input an observation and outputs a set of parameters for specifying the conditional distribution of the latent representation z. In this example, we simply model the distribution as a diagonal Gaussian, and the network outputs the mean and log-variance parameters of a factorized Gaussian. We output log-variance instead of the variance directly for numerical stability.

Decoder network defines the conditional distribution of the observation p(x|z),

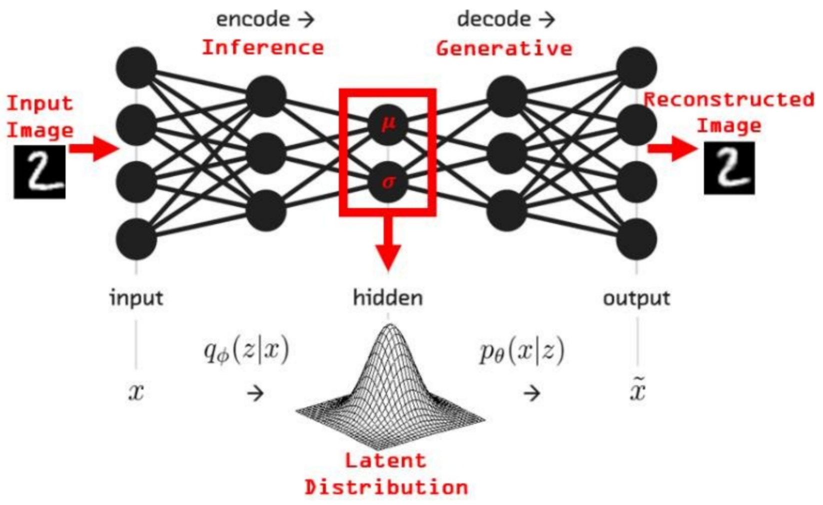


Figure 4: VAE architecture

which takes a latent sample z as input and outputs the parameters for a conditional distribution of the observation. We model the latent distribution prior p(z) as a unit Gaussian.

Our approach was to train a VAE on classical music, so that it could extract features of classical music and based on that features to reconstruct its own melody. As input the neural network had raw signal, that had been read from wav files. Convolutional and pooling layers were applied in order to extract specific features and to downsize the input dimension. Deconvolutional layers were applied to reconstruct the signal from extracted features. The model was trained on short parts of compositions that last for 10 seconds with sample rate set to 3000. As output the model generates the same short compositions that last 10 seconds. But the results look fine only if the model was trained only on single file, otherwise it generates noise or nothing at all.

3.6 RNN

In the family of neural networks, the RNN is a member of feedback neural networks with feedback connections. The ability to send information over time-steps makes the RNN different from the other members of the family of the neural network. When we talk about the working style of RNN, it considers sequential information as input and provides information in sequential form instead of accepting stable input and providing stable output.

The output provided by the layers inside the networks re-enters into the layer as the input which helps in computing the value of the layer and by this process the network makes itself learn based on the current data and previous data together. Talking about the architecture of the basic RNN, we can say that an RNN consists of many copies of the neural network connected and working in a chain.

The audio data is also sequential data where it can be considered as a signal which has modulation with time similarly to the time series data where data points are collected in a sequence with time values. Computer stores audio data in digital form of a series of 1's and 0's. Most of the time, the format for storing the data is pulse-code modulation by taking samples at a repeated rate. The Representation of the audio file can be done by the spectrograms. The spectrogram is made up of plotting frequency with time or amplitude with time. But in our case spectogram did not work well that is why the decision was made not to use them.

Our approach was to create new data set, that consists of a value from signal as Y and previous 100 values to that value as X. The model we used was build out of n number of LSTM layers and a single dense layer with single neuron with linear activation. The model was trained on that data set. For generation a new melody the first 100 values were chosen randomly but the next n values were predicted by model and the duration could set to different values.

3.7 MIDI based Generation

For the music MIDI based generation we used a Long Short-Term Memory (LSTM) network. The LSTM network are a type of Recurrent Neural Network (RNN) that can efficiently learn via gradient descent. Using a gating mechanism, LSTMs are able to recognise and encode long-term patterns. LSTMs are extremely useful to solve problems where the network has to remember information for a long period of time as is the case in music and text generation. The model used for the experiment consisted in three LSTM layers, three Dropout layers, two Dense layers and one activation layer.

The data that the model was training on was the "midi-classic-music" and by using the music21 library we could read the data and split it into: Notes and Chords (a set of notes that are played at the same time). We would map the notes and chords to integers that we could easily feed into the neural network.

The training was done for 16 epochs, on a Nvidia RTX 2060 GPU, an epoch taking approximately 50 minutes to finish training. To the trained model was then given a midi song as a seed, from which a sequence of 100 notes and chords was taken, and the model would predict the following 500 notes. We would then create a new midi using piano notes from the result of the prediction. Unfortunately, because of the high training times we couldn't experiment a lot with this type of this generation, so the resulted midi files tend to be very repetitive in the notes and chords used.

3.8 Data augumentation

3.8.1 Mix noise

This methods adds a background noise to a melody in order to obtain data augmentation. This background noise can be Gaussian noise for example. This background noise will be concatenated to itself in order to match the length of the original song. Combining two noises is easy as a noise is nothing more than a sequence of frequencies. The code can be seen below:

```
noise_signal, noise_sr = librosa.load(noise_path, sr=self.hertz)
while(noise_signal.shape[0] < self.signal.shape[0]):
    noise_signal = np.concatenate((noise_signal, noise_signal))
    noise_signal = noise_signal[:self.signal.shape[0]]
    wav_with_bg = self.signal * np.random.uniform(0.8, 1.2) +
                                  noise_signal * np.random.
                                  uniform(0, 0.1)
```

3.8.2 Cut mix

Cut mix works by combining two different songs in a single one in order to achieve data augmentation. This is done by concatenating parts of each song together according to the original paper. The code used can be seen below:

```
while(new_sound.shape[0] < self.signal.shape[0]):
    new_sound = np.concatenate((new_sound, new_sound))
    new_sound = new_sound[:self.signal.shape[0]]

    lm = random.uniform(0, 1)
    start_point = random.randrange(0, self.signal.shape[0])
    length_cut = int(self.signal.shape[0] * math.sqrt(1 - lm))
    if(start_point + length_cut > self.signal.shape[0]):
        length_cut = self.signal.shape[0] - start_point

    constructed_sound = self.signal[0:start_point]
    constructed_sound = np.concatenate((constructed_sound,
                                  new_sound[start_point:
                                  start_point+length_cut]))
    constructed_sound = np.concatenate((constructed_sound, self.
                                  signal[start_point+length_cut
                                  :]))
```

3.8.3 Time shift

Timeshifting works by slightly moving a song to left or right within its length, this is useful in a similar manner to the usage of slightly transposed images to introduce noise and variety to the data.

3.8.4 Speed tune

Having an effect similar to timeshifting for the data, speed tunning increases or decreases the distance between sounds in an audio file. This adds variety while maintaining the properties of the initial audio file.

3.9 Style Transfer

Style transfer can be used to apply the style of an audio file to another one, the limitation is that it requires two inputs in order to make a single one, and properties of the inputs can be recognized within the output.

This method uses two types of loss, style and content loss, each related to the level of the likeness of the output with each input. Ideally, these values are both minimized so that properties from both inputs are retained.

The content loss will be calculated in a straightforward manner so that the output of the audio file resembles the first input - the content is extracted.

For the style loss, the extraction should be done with a gram matrix so that properties of the sounds are extracted, but not the sounds themselves, which will be mainly extracted from the content input.

The final CNN used is quite shallow since training times can span over hours when using small learning rates and many epochs. The activation function used is ReLU because it is a well-rounded activation function for general neural networks.

Implementation link:
https://colab.research.google.com/drive/11zQjoNgv3ceqA525Z1au3xPd61TeCsPxscrollTo=kTThJ06tWMzz

4 Evaluation and comparison

4.1 Methods results

VAE loss	Style transfer loss	
0.8757	0.017801	0.071829
0.4361	0.015921	0.063078
0.3891	0.015387	0.058685
0.3546	0.015484	0.056174
0.1833	0.016830	0.049146

Both methods take around 30 minutes to train until they converge.

4.2 Statistics

Most obtained results were extremely noisy, and the training duration spanned from approximately 30 minutes for a fast test of the method to a few hours for a more fine tunned generation.

For Style transfer generation we obtained melodies that had at least the style music genre recognizable and the original content also recognizable in over 50 percent of the cases. However, Due to the high level of noise, the obtained audio files did not constitute a satisfactory result.

4.3 Main problems

There was a number of problems encountered during the research and implementation phases of the project:

- Resources about models and network architectures regarding Music generation are limited in variety and amount by comparison with image generation resources.
- Music file formats mainly used for music generation are wav and midi, these formats are difficult to convert from one to the other.

4.4 Limitations

Limitations met RNN:

- Overfits very soon.
- Cannot generalize through big amounts of melodies.
- Only short time dependencies are taken into account.
- Processes only raw wav files.

Limitations met VAE:

- Input/output dimension is very high.
- Cannot generalize through big amounts of melodies.
- Hard to configure, hard to find a working configuration.
- Processes only raw wav files.

Hardware Limitations:

- Training on Nvidia GTX1060 GPU is limited in speed and maximum network size.
- Training on Google Collaboratory has time limits and resource usage quotas for training networks.

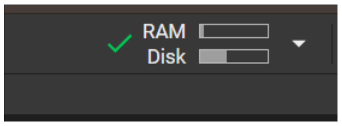


Figure 5: Google Collab RAM and Disk availability, when nothing is running

5 Future work

There are a number of tasks that were not sufficiently covered during this project:

- Testing Combinations of methods for generation.
- Testing GANs, adapting image generation methods to music, or to spectrograms.
- Creating a functional workflow for augmenting and saving batches of input data.
- Further exploration of midi format generation.

6 Results

6.1 Style transfer

First song can be downloaded from: link

This generated song can be easily recognised as classical music. The main notes that can be heard come from a piano. However, the notes seem to be more aggressive due to applying rock style to the original song.

6.2 RNN

Second song can be downloaded from: link

We consider this to be the best sample generated. The notes can be clearly heard and they resemble classical music. The only downside is that it tends to become repetitve for larger sample size.

7 Conclusions

Generating audio files that are enjoyable to hear similarly to a song is possible but more difficult than working with image files.

There are numerous methods that can be employed and refined. However for good results carefully tailored architectures and powerful computing resources are required in order to achieve good results, to circumvent long training times, and avoid noisy or repetitive outputs.

song1

8 Bibliography

References

[1] Francois Chollet, *Deep Learning With Python* 2018

[2] Audio processing *https://github.com/musikalkemist/DeepLearningForAudioWithPython*

[3] Sangdoo Yun, Dongyoon Han, Seong Joon Oh, Sanghyuk Chun, Junsuk Choe, Youngjoon Yoo, *CutMix: Regularization Strategy to Train Strong Classifiers with Localizable Features* 2019

[4] Data preparation and augmentation *https://towardsdatascience.com/audio-deep-learning-made-simple-part-3-data-preparation-and-augmentation-24c6e1f6b52*

[5] VAE arhitecture *https://www.jeremyjordan.me/variational-autoencoders/*

[6] VAE arhitecture *https://medium.com/@realityenginesai/understanding-variational-autoencoders-and-their-applications-81a4f99efc0d*

[7] Music Generation with RNN *https://towardsdatascience.com/how-to-generate-music-using-a-lstm-neural-network-in-keras-68786834d4c5*

[8] LSTM arhitecture and generation *https://github.com/Skuldur/Classical-Piano-Composer*