2021학년도

경기과학고등학교 심화R&E 결과보고서

CycleGAN을 활용한 멜로디 기반 음악 생성 모델 개발

2021. 01. 08

연구참여자 : 최우현(choewuheon@gmail.com)
김희서(graceheeseo@gmail.com)

지도교사: 박종화(suakii@gmail.com)

과학영재학교 경기과학고등학교

초 록

음악에 대한 사람들의 관심이 높아짐에 따라 직접 음악을 만들고자 하는 작곡의 수요도 증가하고 있다. 그러나 작곡 소프트웨어는 일반인이 사용하기에는 난해한 면이 크다. 따라서 본 연구에서는 간단한 멜로디를 입력하면 그에 맞춰 온전한 음악을 생성하는 모델을 개발하였다. 학습 데이터는 웹 크롤링을 통해 수집하였다. 또한, 음악 분야에서 paired dataset을 구하기 어려운 문제를 해결하기 위해 unpaired dataset을 처리 가능한 CycleGAN을 도입해 모델을 설계하였다. 학습 결과 모델의 loss가 감소하며 학습되는 것은 관찰할 수 있었으나, 출력물을 실제 음악으로 변환하여 확인하지는 못했다. 이를 개선하여 멜로디-음악 변환을 성공적으로 이루어낸다면 작곡의 대중화를 이루어낼 수 있을 것으로 기대된다.

I. Introduction

현대에 들어서 유튜브, 사운드클라우드 등의 플랫폼을 통해 음악이 사람들에게 더 쉽게 다가감에 따라 사람들의 음악에 대한 관심이 높아지고 있으며, 이와 동시에 직접 음악을 만들고자 하는 작곡의 수요도 증가하고 있다. 또한 다양한 가격대, 종류의 작곡 소프트웨어의 보급과 함께 작곡에 인공지능을 활용하기도 한다. 그럼에도 불구하고 작곡 소프트웨어는 여전히 난해하고 배워야 할 내용들이 많으며, 소프트웨어의 활용 외에도 화성악이나 사용되는 악기에 대한 이해가 어느정도 포함되어야 한다. 작곡에 사용되는 인공지능은 성과를 거두고 있고, 과거의 음악으로부터 비슷한 분위기의 멜로디를 만들어내는 데에는 효과적이지만, 생성된 멜로디는 최종적으로는 전문가의 손길을 거쳐야만 한다. 이 때문에 작곡 초보자들은 떠오른 멜로디가 있다고 하더라도 이를 온전한 곡의 형태로 담아내는 것을 포기해야 하는 경우가 많다.

1.1 Related Works

작곡이라는 예술의 분야에 인공지능을 사용하려는 연구는 꽤 있으며, 이 연구들의 결과는 작곡에 인공지능을 사용하는 것에 대해 긍정하고 있다.선행연구에 활용되고 있는 인공신경망은 크게 두가지로 나뉘고 있다. 하나는 Multi-Track 데이터를 활용하여 이를 2차원 형태로 바꾸고 이를 이미지와 비슷하게 생각하여 처리하는 방식이고, 다른 하나는 음악의 흐름 자체를 순환적 신경망에 넣어 결과를 받는 형태가 있다. 전자의 경우 후자에 비해 Multi-Track을 다루고 다양한 특성을 뽑아내는 데에 효과적이고, 후자의 경우에는 순환적 신경망의 특징인 흐름을 다루는 데에 좀 더 효과적이다.

1.2 Theoretical Backgrounds

1.2.1 MIDI

MIDI 파일은 mp3, wav와 같은 음악 파일 포맷의 한 종류로, 소리 자체가 아닌 악기들에 대한 명령을 담고 있다. 저장해야하는 정보가 훨씬 적기 때문에 시간에 따른 소리를 저장하는 다른 포맷에 비해 적은 메모리를 차지한다. 또한, 사용된 악기와 음에 대한 정보를 가지고 있기 때문에 다른 포맷에 비해 효과적으로 음악 정보를 분석할 수 있다.

1.2.2 GAN

GAN(생성적 적대 신경망, Generative Adversarial Networks)은 실제와 비슷한 데이터를 생성해내는 신경망이다. GAN은 일반적으로 generator와 discriminator로 이루어져 있다. Generator는 실제와 비슷한 가짜 데이터를 생성하는 역할을 하고, discriminator는 generator가 생성한 데이터와 실제 데이터를 구분하는 역할을 한다. 두 신경망을 경쟁적으로 학습시켜 최종적으로 실제와 구분할 수 없는 데이터를 생성해내는 것이 GAN의 목표이다.

1.2.3 CycleGAN

CycleGAN은 GAN의 한 종류로, 본래 image to image 변환을 위해 개발되었다. GAN과 비교되는 CycleGAN의 장점은 변환에 있어 데이터셋이 짝을 이루지 않는 unpaired dataset에도 사용할 수 있다는 것이다. 본 연구의 경우 하나의 음악에 대해 전체 MIDI 파일과 멜로디만 담은 MIDI 파일을 모두 구하는 것은 매우 어렵기 때문에, unpaired dataset을 처리할 수 있는 강점을 가진 CycleGAN을 도입하였다. CycleGAN은 A 형태를 B로 바꾸는 generator와 B 형태를 A로 바꾸는 generator, 그리고 A 형태와 B 형태에 대한 discriminator로 이루어져 있다. 이때 discriminator의 loss만을 학습에 이용하는 것이 아닌, A에서 B로 변환시킨 것을 다시 A로 돌려놓을 수 있도록, 즉 정답 데이터 없이도 형태를 변화시키며 고유한 특성은 잃지 않도록 학습시키는 cycle consistency를 추가하여 학습의 정확도를 높인다.

1.3 Aim of the Research

본 연구에서는 GAN의 변형인 CycleGAN구조를 가지는 네트워크 신경망을 설계하여 상기한 문제들을 해결하고 초보자도 간단한 멜로디만을 가지고 하나의 곡을 완성해 낼 수 있도록 하고자 한다. 간단한 멜로디에 대응되는 음악 파일의 쌍이 많지 않고, 있다고 하더라도 비공개이기 때문에 구하기에 어려움이 있다. 따라서 꼭 대응되는 쌍이 아니더라도 이를 기반으로 훈련을 진행할 수 있는 CycleGAN 구조를 이용하였다. 또한 선행 연구들은 대부분 음악을 생성해 내는 것 그 자체에 초점을 맞추고 있으나 우리는 반대로 이미 생성되어 있는 멜로디를 일반적인 곡의 형태로 다듬어 내는 인공지능을 만들고자 한다.

1.4 Background Setup

본 연구에서 사용한 Python 및 각 라이브러리의 버전은 다음과 같다.

- Python: 3.6.9
- Tensorflow: 2.4.0
- Keras: 2.4.0
- matplotlib: 3.2.2
- numpy: 1.19.4
- urllib: 1.25.9
- requests: 2.24.0
- BeautifulSoup: 4.9.1
- Music21: 6.1.0
- mido: 1.2.9

II. Dataset

본 연구에서는 학습에 사용할 MIDI 파일을 직접 다운로드하였고, 이를 가공하여 데이터셋을 제작하였다. 전체 음악을 담은 full MIDI 파일과 간단한 멜로디를 담은 melody MIDI 파일을 각각 다운받아 unpaired dataset으로 학습을 진행한다.

2.1 Data Collection

Python3에서 지원하는 urllib, requests, BeautifulSoup 라이브러리를 사용한 크롤링을 통해 데이터를 수집했다. 게임 음원을 무료로 제공하는 사이트 www.khinsider.com/midi에서 20007개의 full MIDI 파일을 다운받았다. 작곡 학습용 MIDI 파일을 제공하는 사이트 cymatics.fm에서 531개의 melody MIDI 파일을 다운받았다. 깨지거나 학습에 부적합한 파일을 제거한 후, 최종적으로 19802개의 full MIDI 파일과 531개의 melody MIDI 파일을 얻었다.

2.2 Data Pretreatment

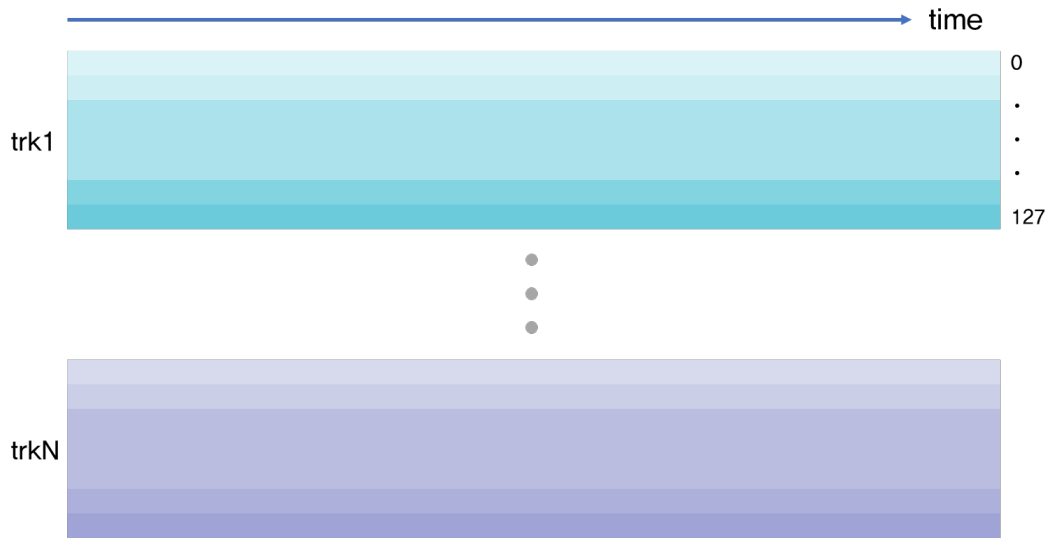

Fig. 1. MIDIsamp의 구조

Multi-Track MIDI 파일을 모델이 분석하기에 더 적합한 형태로 바꾸기 위하여 Python의 Music21과 mido 라이브러리를 사용하였다. Music21을 사용해 MIDI 파일의 각 track 별로 노트의 켜짐 이벤트, 꺼짐 이벤트, 이벤트 간 시간 간격 정보를 얻고, 이를 가공하여 노트 리스트로 변환하였다. 각 노트는 시작 시각, 끝 시각, 음계 정보를 담고 있다. Music21에서

제공하는 시간 간격은 midi tick을 기준으로 하고 있는데, 이는 MIDI 파일의 bitrate에 의해 달라질 수 있으므로 정확한 정보가 아니다. 따라서 mido 라이브러리를 이용해 bitrate를 얻고, 시간 간격은 bitrate를 4로 나눈 값으로 나누어 주었다.

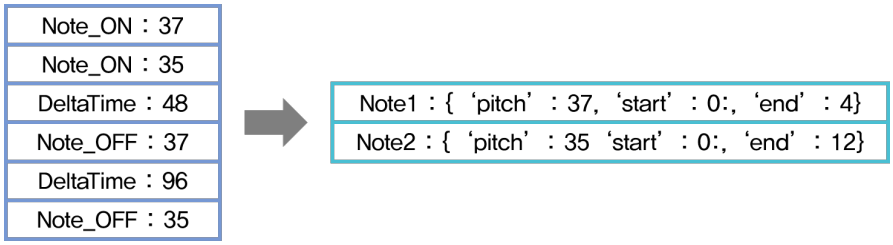

Fig. 2. Note 정보 변환 과정 예시, bitrate=48인 경우

노트 리스트를 이용해 최종적으로 음계별로 시간에 따른 켜짐/꺼짐 상태를 piano roll 형태로 만들었다. 음계는 [0, 127]의 범위를 가지며, 음계별 piano roll을 세로로 이어붙여 하나의 track에 대해 가로는 음악의 길이, 세로는 128인 numpy 배열을 얻었다. 모든 track의 numpy 배열을 세로로 이어붙여 원본 음악을 piano roll로 변환한 최종 numpy 배열을 얻었다. 이를 편의상 MIDIsamp로 명명하였다.

III. Model

본 연구에서 개발한 모델은 크게 DataGenerator, Encoder-Decoder, CycleGAN의 세 가지 부분으로 나눌 수 있다. DataGenerator는 한 번에 필요한 만큼만 데이터를 불러와 기억 장치 용량의 부담을 줄이는 역할을 한다. Encoder-Decoder는 MIDIsamp 형태로 저장된 데이터를 CycleGAN에 입력하기 적절한 형태로 바꾸는 역할을 한다. CycleGAN은 본 연구의 핵심적인 부분으로, full MIDI와 melody MIDI 간의 변환을 수행한다.

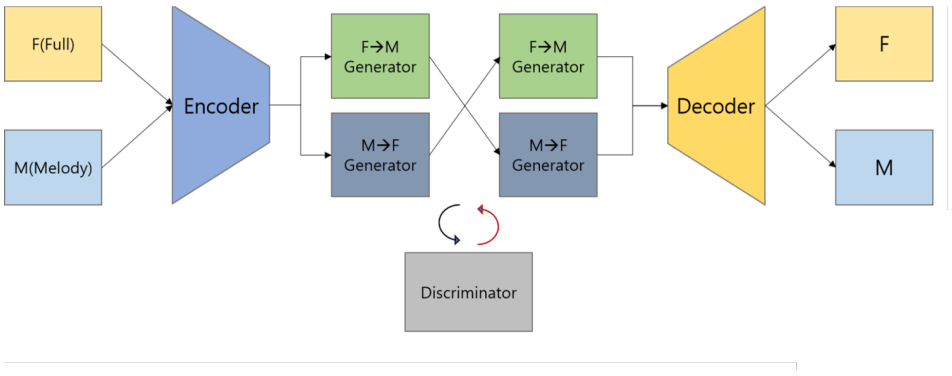

Fig. 3. Encoder-Decoder와 CycleGAN

3.1 DataGenerator

본 연구에서 사용한 모델은 구조가 여러 단계로 나뉘어 있고 모델 하나하나의 파라미터 개수가 적지 않다. 또한, 모든 데이터를 MIDIsamp로 사전에 바꾸어 저장할 경우 용량이 비약적으로 증가한다. 때문에 연산의 부담을 줄이고자 DataGenerator를 제작하였다. DataGenerator는 학습을 위한 데이터를 매 훈련을 시작할 때 마다 Batch 만큼만 불러와, MIDIsamp로 바꾸고 필요한 전처리를 해준다. 이를 통해 한 번에 데이터를 전부 불러 옴으로서 가해지는 기억장치 용량 부담을 줄이는 역할을 한다.

3.2 Encoder-Decoder

CycleGAN은 본래 image-to-image 변환을 위해 개발된 모델로, 2D Convolution Layer를 포함하고 있기 때문에 입력 데이터를 정사각형 꼴로 만드는 것이 적합하다. 이를 위해 MIDIsamp numpy 배열에 0을 padding하여 정사각형 꼴로 변환하여 사용하고자 하였다. 그러나 이를 그대로 입력으로 사용하면 의미 없는 정보를 너무 많이 담고 있게 되고, 크키가 커서 연산의 복잡도도 증가한다. 따라서 CycleGAN에 데이터를 입력하기 전 이를 의미 있는 정보만 담고 있도록 압축하는 Encoder-Decoder 모델을 따로 설계하였다.

Encoder는 0을 paddding한 MIDIsamp를 입력으로 받으며, 2D Convolution Layer를 통과시켜 2차원 벡터를 출력한다. 이를 MIDIvec으로 명명하였다. Decoder는 Encoder와 대칭적인 구조로 이루어져 있으며, MIDIvec 형태의 데이터를 입력받아 MIDIsamp로 변환하는 역할을 한다. Decoder가 이 과정을 잘 수행하도록 학습된다면 만들어진 MIDIvec이 원본 데이터의 특징을 잘 담고 있는 것으로 간주할 수 있다. Decoder는 또한 최종 결과 출력에 관여하기 때문에, decoding 과정에서 부가적인 작용을 통해 출력물이 온전한 음악의 형태를 갖도록 돕는 효과가 있을 것으로 기대된다. Encoder-Decoder 모델은 MSE를 손실함수로 사용하였다.

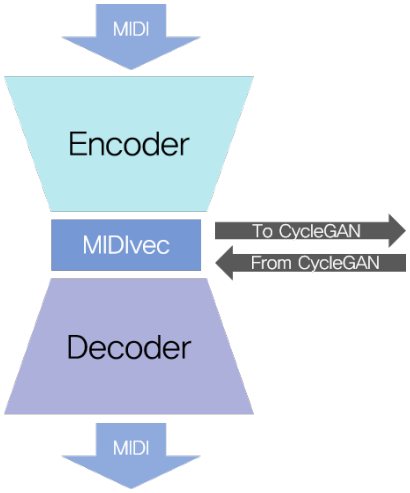

Fig. 4. Encoder-Decoder 모델의 모식도

Layer (type)	Output Shape	Param #
input_79 (InputLayer)	[(None, 2048, 2048, 1)]	0
conv2d_1028 (Conv2D)	(None, 1024, 1024, 4)	104
instance_normalization_788 (	(None, 1024, 1024, 4)	8
activation_884 (Activation)	(None, 1024, 1024, 4)	0
conv2d_1029 (Conv2D)	(None, 512, 512, 8)	808
instance_normalization_789 (	(None, 512, 512, 8)	16
activation_885 (Activation)	(None, 512, 512, 8)	0
conv2d_1030 (Conv2D)	(None, 256, 256, 16)	3216
instance_normalization_790 (	(None, 256, 256, 16)	32
activation_886 (Activation)	(None, 256, 256, 16)	0
conv2d_1031 (Conv2D)	(None, 128, 128, 4)	1604
instance_normalization_791 (	(None, 128, 128, 4)	8
activation_887 (Activation)	(None, 128, 128, 4)	0
conv2d_1032 (Conv2D)	(None, 64, 64, 1)	101
instance_normalization_792 (	(None, 64, 64, 1)	2
activation_888 (Activation)	(None, 64, 64, 1)	0

Total params: 5,899
Trainable params: 5,899
Non-trainable params: 0

decoder
Model: "model_71"

Layer (type)	Output Shape	Param #
input_80 (InputLayer)	[(None, 64, 64, 1)]	0
up_sampling2d_50 (UpSampling	(None, 128, 128, 1)	0
conv2d_1033 (Conv2D)	(None, 128, 128, 4)	104
instance_normalization_793 (	(None, 128, 128, 4)	8
activation_889 (Activation)	(None, 128, 128, 4)	0
up_sampling2d_51 (UpSampling	(None, 256, 256, 4)	0
conv2d_1034 (Conv2D)	(None, 256, 256, 16)	1616
instance_normalization_794 (	(None, 256, 256, 16)	32
activation_890 (Activation)	(None, 256, 256, 16)	0
up_sampling2d_52 (UpSampling	(None, 512, 512, 16)	0
conv2d_1035 (Conv2D)	(None, 512, 512, 8)	3208
instance_normalization_795 (	(None, 512, 512, 8)	16
activation_891 (Activation)	(None, 512, 512, 8)	0
up_sampling2d_53 (UpSampling	(None, 1024, 1024, 8)	0
conv2d_1036 (Conv2D)	(None, 1024, 1024, 4)	804
instance_normalization_796 (	(None, 1024, 1024, 4)	8
activation_892 (Activation)	(None, 1024, 1024, 4)	0
up_sampling2d_54 (UpSampling	(None, 2048, 2048, 4)	0
conv2d_1037 (Conv2D)	(None, 2048, 2048, 1)	101
instance_normalization_797 (	(None, 2048, 2048, 1)	2
activation_893 (Activation)	(None, 2048, 2048, 1)	0

Total params: 5,899
Trainable params: 5,899
Non-trainable params: 0

Fig. 5. encoder와 decoder

3.3 CycleGAN

CycleGAN은 full MIDI와 melody MIDI 간의 변환을 수행한다. 일반적인 구성과 같이 full->melody, melody->full 변환을 각각 수행하는 Generator 2개와 full/melody 각각에 대한 Discriminator 2개로 이루어져 있다.

CycleGAN의 Generator는 실질적으로 변환을 수행하는 부분으로, 본 연구에서 핵심적인 부분이다. 일반적으로 성능 향상을 위해서는 신경망을 깊게 설계하는 것이 유리하므로, 이를 위해 ResNet 구조를 도입하였다. ResNet 구조는 신경망이 깊을 때 역전파가 제대로 일어나지 않는 vanishing gradient 문제가 적게 발생한다. 또한, 깊이를 늘리더라도 파라미터 개수가 많지 않아 학습 부담이 적은 장점이 있다.

Discriminator는 일반적인 CycleGAN과 동일한 구조로 설계하였으며, Convolution 2D layer로 구성하였다.

Discriminator에 의한 loss는 MSE, cycle consistency에 의한 loss는 MAE를 손실함수로 사용하였다.

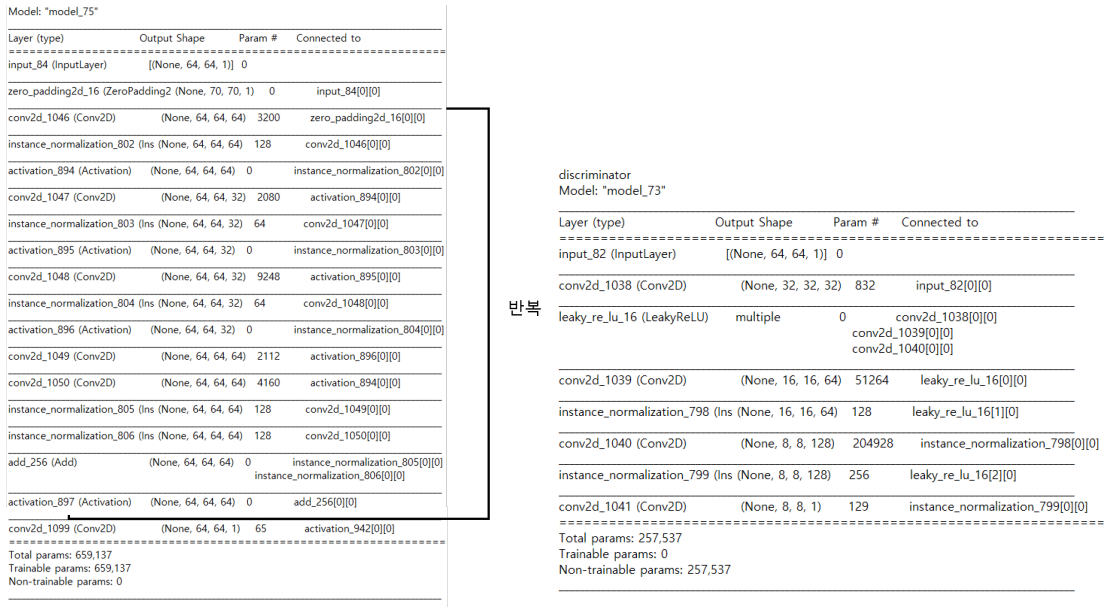

Fig. 6. Generator and Discriminator

IV. Training

본 연구에서는 데이터를 12개씩 묶어 batch별로 훈련을 진행하였으며, GAN에서 Generator 부분과 Discriminator 부분을 나눠서 따로 훈련하는 것과 비슷하게 Encoder-Decoder 부분과 CycleGAN 부분을 나누어서 훈련을 진행하였다. 또한 모델에 맞는 입력으로 변환하는 과정에서 zero padding으로 인해 채워진 부분이 많아 기존 midi 파일에 비해 용량이 비약적으로 증가하여 사전에 전처리를 전부 진행할 경우 용량이 부족한 현상이 발생하였다. 이를 해결하기 위해 전처리하는 부분을 Data Generator 내부로 집어넣어 훈련과 동시에 전처리를 진행하도록 하여 실질적인 사용 용량을 크게 감소시켰다. 또한 사전에 문제가 있는 데이터들을 모두 걸러내는 과정을 거쳤음에도 불구하고 문제가 있는 파일들이 발생하였다. 이를 해결하기 위해서 전체 데이터를 한번에 훈련하는 것이 아닌 여러 부분으로 나누고, 한 부분에 대한 훈련이 끝난 뒤에 모델을 저장하고, 이후 이 모델을 불러와서 남은 부분들에 대한 훈련을 진행하는 방식으로 이 문제를 해결하였다

V. Results

우선 Encoder-Decoder 부분의 경우에는 훈련 과정에서 손실이 줄어드는 것이 확인되었다. 본 연구에서는 훈련이 진행된 후에도 손실이 여전히 남아 있었으나 그 수치가 크지는 않았다. 또한, 일반적인 모델과 다르게 데이터에 별다른 작업을 하지 않고, 단순히 압축과 압축 해제만을 반복하기 때문에 훈련 시간을 늘리고 깊이를 증가시킬 경우 손실을 더 감소시키는 것이 어렵지 않게 가능할 것으로 기대된다. CycleGAN 부분 역시 훈련이 진행됨에 따라 손실이 감소하였으며, 훈련 후 이를 테스트한 결과 입력된 데이터와 비교했을때 배열이 크게 복잡해진 것을 확인할 수 있었다. 다만 이 데이터가 본 연구에서 제작하고자 했던 올바르고 풍성한 음악을 의미하는지를 판가름하기 위해서는 처음에 진행했던 midi 파일을 배열로 변환하는 과정의 역변환을 구현해야 하는데 이 부분이 쉽지 않았고, 또한 바꾼다 하더라도 많은 사람들의 정성적인 평가를 취합해야 하기 때문에 제대로 된 평가를 진행하지 못하였다.

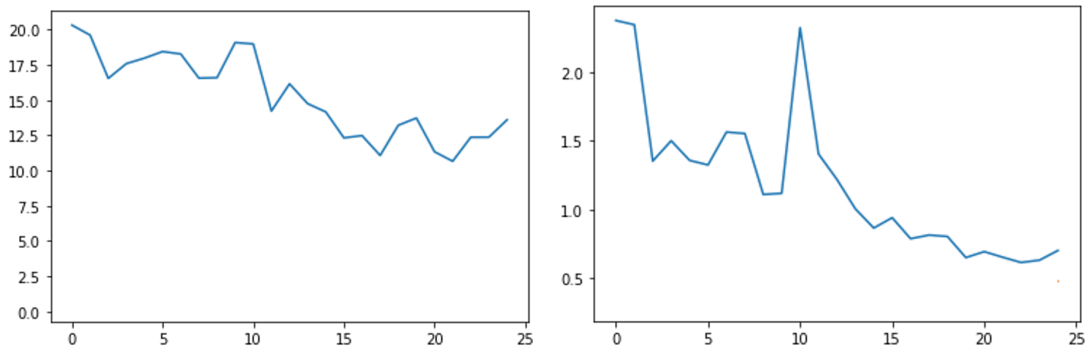

Fig. 7. Loss

VI. Conclusion and Suggestions

학습이 진행됨에 따라 Generator와 Discriminator 양쪽의 loss가 감소하는 경향성을 관찰할 수 있었다. 시간의 부족으로 학습을 충분히 오래 진행하지 못하였는데, 계속해서 학습을 진행할 경우 loss가 더 감소하였을 것으로 기대된다.

그러나 시간과 자원의 부족으로 출력된 벡터를 실제 음악으로 변환하여 확인하지는 못했다. 실제 MIDI 파일을 이용하여 만들어진 MIDIsamp 벡터는 가로에 비해 세로가 매우 짧으며, 학습 시에는 zero padding을 통해 정사각형 꼴로 만들어 사용하였다. 그러나 모델에서 출력된 벡터는 정사각형 꼴이지만 벡터 전체에 값이 고르게 분포되어 있을 확률이 높다. 따라서 출력값을 MIDI 파일로 변환할 때는 이러한 부분을 추가적으로 고려해 주어야 할 것으로 생각된다. 또한, 입력값은 0 또는 1의 값만을 갖지만 출력값은 그렇지 않다. 따라서 정규화 후 특정 음이 연주되고 있는지 여부를 판단하는 역치를 적절하게 설정해 주어야 할 것이다. 출력 벡터를 MIDI 파일로 변환하는 방법으로는 데이터 전처리의 역과정으로 노트를 생성하고, Music21 라이브러리의 Stream 객체를 이용해 노트를 합쳐 하나의 음악으로 출력하는 방법을 제안한다.

그러나 출력을 성공적으로 음악으로 바꾸어 내더라도, 이 음악이 정말 '멜로디가 보존되어 있고', '들을만한' 음악인지를 판별하는 것에는 객관적인 기준이 없다. 따라서 이를 정확하게 판별하기 위한 추가적인 기준이 필요할 것이다.

앞에서 언급된 사항을 개선하여 연구를 발전시키고 이를 통해 멜로디를 바탕으로 빠르게 음악을 만들 수 있게 된다면, 작곡의 진입장벽이 낮아지고 그로 인해 작곡의 대중화가 이루어질 것으로 기대된다. 야마하에서 제작한 보컬로이드라는 프로그램은 직접 노래를 부르지 못하거나 부를 사람을 구하지 못하더라도 프로그램을 사용하여 노래를 만들어 낼 수 있게 만들어 주었다. 그 결과 많은 사람들이 노래 제작에 적은 자원을 가지고도 참여 할 수 있게 되었고, 더 다양한 음악들이 만들어졌다. 또한 이렇게 만들어진 노래를 아마추어 가수들이 부르면서 작곡가와 가수 사이를 연결시켜 주었다. 이러한 선례를 참고한다면 본 연구의 개선에 성공할 경우 작곡에 들어가는 자원을 더욱 줄여서 하고자 하는 모든 사람들이 작곡을 할 수 있도록 도와 줄 수 있을 것이다. 이는 결국 음악의 다양성을 더욱 확장하고, 가능성을 가진 많은 사람들이 음악을 만드는 데에 참여할 수 있는 결과를 가져올 것이다.

VII. Appendix

```
from tensorflow.keras.models import Model
from tensorflow.keras.models import Sequential
from tensorflow.keras.optimizers import *
from tensorflow.keras.utils import *
from tensorflow.keras import preprocessing
from tensorflow.keras.layers import *
from tensorflow.keras import losses
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import load_model
import random
import matplotlib.pyplot as plt
from tensorflow_addons.layers import *
import os
import glob
import music21
import mido
from music21 import *
import math

data_path = '/content/gdrive/My Drive/MIDIdataset'
model_path = os.path.join(data_path, "model")
batch_size = 12
start_batch = 0
end_batch = 300
pre_epoch = 1
epoch = 1
```

```
28
29 def midi2keystrikes(mf,tracknum, div):
30     events = mf.tracks[tracknum].events
31     result = []
32     noteon = {}
33     notetime = []
34     chk = []
35     t = 0
36     div = div//4
37
38     for i in range(0, 128):
39         noteon[i] = 0
40
41     for e in events:
42
43         if e.isDeltaTime and (e.time is not None) and (e.
               time != 0):
44             t += e.time
45
46         elif (e.type == midi.channelVoiceMessages.NOTE_ON
               and ( e.pitch is not None) and (e.velocity != 0)
               ):
47             noteon[e.pitch] = t
48
49         elif ((e.pitch is not None) and (e.velocity == 0)):
50             result.append({'stime':noteon[e.pitch]//div, '
                   etime':t//div, 'note':e.pitch})
51             noteon[e.pitch] = 0
52
```

```
53
54      for i in range(0, 128):
55          if(noteon[i]!=0):
56              result.append({'stime':noteon[i]//div, 'etime':
                    t//div, 'note':i})
57              noteon[i]=0
58
59      '''
60      if (len(result) == 0) and (tracknum < len(mf.tracks)-1)
            :
61          # if it didn't work, scan another track.
62          return midi2keystrikes(mf,tracknum+1)
63      '''
64      return result
65
66  def getMidiData(mf, div):
67      alltrknum = len(mf.tracks)
68      mn = 2147483647
69      mx = 0
70      cnt = 0
71      mus = []
72      for i in range(0, alltrknum):
73          mus.append(midi2keystrikes(mf, i,  div))
74
75      for i in mus:
76          for j in i:
77              mx = max(mx, j['etime'])
78              mn = min(mn, j['stime'])
79
```

```
80      ret = np.zeros((alltrknum*128, mx+4))
81      for i in mus:
82          for j in i:
83              ss = j['stime']
84              ee = j['etime']
85              nn = j['note']
86              for k in range(ss, ee):
87                  ret[cnt*128+nn][k] = 1
88          cnt = cnt + 1
89      return ret, mx, mus
90
91  class Pre():
92      def __init__(self):
93          self.shape = (2048,2048,1)
94          self.encoder = self.encode()
95          self.decoder = self.decode()
96          optimizer = Adam(0.0002, 0.5)
97
98          mid = Input(shape = self.shape)
99          vec = self.encoder(mid)
100         re_mid = self.decoder(vec)
101
102         self.combined = Model(inputs = mid, outputs =
                re_mid)
103         self.combined.compile(loss = 'mse', optimizer=
                optimizer, metrics = ['accuracy'])
104         self.combined.summary()
105
106     def encode(self):
```

```
107         def layer(x, filter, x_stride, y_stride):
108             x = Conv2D(filter, (5, 5), strides = (x_stride,
                    y_stride), padding = 'same')(x)
109             x = InstanceNormalization()(x)
110             x = Activation('relu')(x)
111             print(x.shape)
112             return x
113         input = Input(shape = self.shape)
114         d = layer(input, 4, 2, 2)
115         d = layer(d, 8, 2, 2)
116         d = layer(d, 16, 2, 2)
117         d = layer(d, 4, 2, 2)
118         d = layer(d, 1, 2, 2)
119         return Model(input, d)
120
121     def decode(self):
122         def layer(x, filter, x_stride, y_stride):
123             x = Conv2D(filter, (5,5), strides = (1, 1),
124                 padding = 'same')(UpSampling2D(size = (
                      x_stride, y_stride))(x))
125             x = InstanceNormalization()(x)
126             x = Activation('relu')(x)
127             print(x.shape)
128             return x
129         input = Input(shape = (64, 64, 1))
130         d = layer(input, 4, 2, 2)
131         d = layer(d, 16, 2, 2)
132         d = layer(d, 8, 2, 2)
133         d = layer(d, 4, 2, 2)
```

```
134         d = layer(d, 1, 2, 2)
135         return Model(input, d)
136
137 class GAN():
138     def __init__(self):
139         self.rows = 64
140         self.cols = 64
141         self.dim = 1
142         self.shape = (self.rows, self.cols, self.dim)
143         self.dis_shape = (8, 8, 1)
144         self.filternum = 32
145         self.dlayer_num = 3
146         optimizer = Adam(0.0002, 0.5)
147         self.lambda_cycle = 10.0
148         self.lambda_id = 0.1 * self.lambda_cycle
149
150         self.d_A = self.build_D()
151         self.d_B = self.build_D()
152         self.d_A.compile(loss='mse', optimizer = optimizer,
                metrics = ['accuracy'])
153         self.d_B.compile(loss='mse', optimizer = optimizer,
                metrics = ['accuracy'])
154
155         self.g_AB = self.build_resG()
156         self.g_BA = self.build_resG()
157
158         origin_A = Input(shape=self.shape)
159         origin_B = Input(shape=self.shape)
160
```

```
161         fake_B = self.g_AB(origin_A)
162         fake_A = self.g_BA(origin_B)
163
164         reconstr_A = self.g_BA(fake_B)
165         reconstr_B = self.g_AB(fake_A)
166
167         oorigin_A = self.g_BA(origin_A)
168         oorigin_B = self.g_AB(origin_B)
169
170         self.d_A.trainable = False
171         self.d_B.trainable = False
172
173         valid_A = self.d_A(fake_A)
174         valid_B = self.d_B(fake_B)
175
176         self.d_A.summary()
177
178         self.combined = Model(inputs = [origin_A, origin_B
                ], outputs = [ valid_A, valid_B, reconstr_A,
                reconstr_B, oorigin_A, oorigin_B ])
179         self.combined.compile(loss = ['mse', 'mse','mae', '
                mae', 'mae', 'mae'], loss_weights = [ 1, 1, self
                .lambda_cycle, self.lambda_cycle,self.lambda_id,
                 self.lambda_id ], optimizer=optimizer)
180         self.combined.summary()
181
182     def build_D(self):
183         def conv(input, filternum, function = LeakyReLU(
                alpha = 0.2), size = 5, normalization = True):
```

```
184             temp = Conv2D(filternum, kernel_size = size,
                    strides = 2, padding = 'same')(input)
185             temp = function(temp);
186             if normalization:
187                 temp = InstanceNormalization()(temp)
188             return temp
189         data = Input(shape = self.shape)
190         temp = conv(input = data, filternum = self.
                filternum, normalization = False)
191         filters = self.filternum*2
192         for i in range(1,self.dlayer_num):
193             temp = conv(input = temp, filternum = filters)
194             filters = filters*2
195         val = Conv2D(1, (1,1), strides = (1,1), padding = '
                valid')(temp)
196         print("D shape")
197         print(val.shape)
198         return Model(data,val)
199
200     def build_resG(self):
201         def resblock(x, filternum, layernum):
202             shortcut = x;
203             for i in range(layernum):
204                 x = Conv2D(filternum, (1, 1), strides = (1,
                        1), padding = 'valid')(x)
205                 x = InstanceNormalization()(x)
206                 x = Activation('relu')(x)
207                 x = Conv2D(filternum, (3, 3), strides = (1,
                        1), padding = 'same')(x)
```

```
208                 x = InstanceNormalization()(x)
209                 x = Activation('relu')(x)
210                 x = Conv2D(filternum*2, (1, 1), strides=(1,
                        1), padding = 'valid')(x)
211                 if i==0:
212                     shortcut = Conv2D(filternum*2, (1, 1),
                            strides = (1, 1), padding = 'valid')
                            (shortcut)
213                     x = InstanceNormalization()(x)
214                     shortcut = InstanceNormalization()(
                            shortcut)
215
216                 x = Add()([x, shortcut])
217                 x = Activation('relu')(x)
218                 shortcut = x
219             return x
220
221         input = Input(shape = self.shape)
222         x = ZeroPadding2D(padding = (3, 3))(input)
223         x = Conv2D(64, (7, 7), strides = (1, 1))(x)
224         x = InstanceNormalization()(x)
225         x = Activation('relu')(x)
226         x = resblock(x, 32, 3)
227         x = resblock(x, 64, 4)
228         x = resblock(x, 64, 6)
229         x = resblock(x, 32, 3)
230         x = Conv2D(1, (1,1), strides = (1,1), padding = '
              valid')(x)
231         print("G shape")
```

```
232         print(x.shape)
233         return Model(input,x)
234
235 def convshape(t_in):
236     t_v = []
237     for i in t_in:
238         temp = preprocessing.sequence.pad_sequences(np.
                array(i), maxlen = 2048, padding = 'post')
239         t_v.append(temp)
240     pad = preprocessing.sequence.pad_sequences(np.array(t_v
            ), maxlen = 2048, padding = 'post')
241     return pad
242
243 class DataLoader:
244     def __init__(self, full_dir, melod_dir, batch_size,
            shuffle = True):
245         self.batch_size = batch_size
246         self.full_dir = full_dir
247         self.melod_dir = melod_dir
248         self.full_list = os.listdir(full_dir)
249         self.full_list = self.full_list[start_batch:
                end_batch]
250         self.melod_list = os.listdir(melod_dir)
251         self.full_i = 0
252         self.melod_i = 0
253         self.shuffle = shuffle
254
255     def load(self):
256         t_full = []
```

```
257         t_melod = []
258         for i in range(0,batch_size):
259 
260             if self.full_i >= len(self.full_list):
261                 self.full_i = 0
262                 if self.shuffle == True:
263                     np.random.shuffle(self.full_list)
264             if self.melod_i >= len(self.melod_list):
265                 self.melod_i = 0
266                 if self.shuffle == True:
267                     np.random.shuffle(self.melod_list)
268 
269             full_in = np.load(os.path.join(self.full_dir,
                    self.full_list[self.full_i]))
270             melod_in = np.load(os.path.join(self.melod_dir,
                    self.melod_list[self.melod_i]))
271 
272             t_full.append(full_in)
273             t_melod.append(melod_in)
274 
275             self.full_i = self.full_i + 1
276             self.melod_i = self.melod_i + 1
277 
278         return np.array(convshape(t_full)), np.array(
                convshape(t_melod))
279 
280 class DataGenerator(Sequence):
281     def __init__(self, direction, namelist, batch_size,
            shuffle = True):
```

```
282         self.batch_size = batch_size
283         self.direction = direction
284         self.namelist = namelist
285         self.shuffle = shuffle
286         self.on_epoch_end()
287 
288     def __len__(self):
289         return int(np.floor(len(self.namelist) / self.
                batch_size))
290 
291     def __getitem__(self, index):
292         numlist = self.numlist[index*self.batch_size:(index
                +1)*self.batch_size]
293         temp = [self.namelist[k] for k in numlist]
294         X, y = self.__data_generation(temp)
295         return X, y
296 
297     def on_epoch_end(self):
298         self.numlist = np.arange(len(self.namelist))
299         if self.shuffle == True:
300             np.random.shuffle(self.numlist)
301 
302 
303     def __data_generation(self, temp):
304         X = []
305         for i, ID in enumerate(temp):
306             mf = music21.midi.MidiFile()
307             mf.open(os.path.join(self.direction, ID))
308             mf.read()
```

```
309             mf.close()
310             pattern = mido.MidiFile(os.path.join(self.
                    direction, ID))
311             div = pattern.ticks_per_beat
312             arr, mx, mus = getMidiData(mf, div)
313             X.append(arr)
314         return np.array(X), np.array(X)
315
316 def train():
317
318     melod_path = "/content/gdrive/My Drive/MIDIdataset/
            melodyMIDI"
319     full_path = "/content/gdrive/MyDrive/MIDIdataset/
            fullMIDI"
320
321     if os.path.isfile(os.path.join(model_path,'EnD.h5')):
322         EnD = load_model(os.path.join(model_path, 'EnD.h5')
                )
323     else:
324         EnD = Pre()
325         data_list = os.listdir(full_path)
326         data_list = data_list[start_batch:end_batch]
327         data_generator = DataGenerator(full_path, data_list
                , batch_size)
328
329         batch_num = int(len(os.listdir(full_path)) /
                batch_size)
330
```

```
331         hist = EnD.combined.fit_generator(generator =
                data_generator, epochs = pre_epoch,
                use_multiprocessing=False)
332         np.save(os.path.join(model_path,'result.npy'),hist.
                history['loss'])
333         EnD.combined.save(os.path.join(model_path,'EnD.h5')
                )
334
335     Gan = GAN()
336     loader = DataLoader(full_path, melod_path, batch_size)
337     valid = np.ones((batch_size,) + (8, 8, 1))
338     fake = np.zeros((batch_size,) + (8, 8, 1))
339
340     g_result = []
341     d_result = []
342
343     for i in range(epoch):
344         for j in range(batch_num):
345             (A, B) = loader.load()
346
347             A = A[..., np.newaxis]
348             B = B[..., np.newaxis]
349
350             A = EnD.encoder.predict(A)
351             B = EnD.encoder.predict(B)
352
353             fake_B = Gan.g_AB.predict(A)
354             fake_A = Gan.g_BA.predict(B)
355
```

```
356             dA_loss_real = Gan.d_A.train_on_batch(A, valid)
357             dA_loss_fake = Gan.d_A.train_on_batch(fake_A,
                    fake)
358             dA_loss = 0.5 * np.add(dA_loss_real,
                    dA_loss_fake)
359
360             dB_loss_real = Gan.d_B.train_on_batch(B, valid)
361             dB_loss_fake = Gan.d_B.train_on_batch(fake_B,
                    fake)
362             dB_loss = 0.5 * np.add(dB_loss_real,
                    dB_loss_fake)
363
364             d_loss = 0.5 * np.add(dA_loss, dB_loss)
365
366             g_loss = Gan.combined.train_on_batch([A, B],
367                 [valid, valid, A, B, A, B])
368             g_result.append(g_loss)
369             d_result.append(d_loss)
370
371     Gan.combined.save(os.path.join(model_path,'GAN.h5'))
372     np.save(os.path.join(model_path,'d_loss.npy'),d_result)
373     np.save(os.path.join(model_path,'G_loss.npy'),g_result)
374
375 train()
```