Sound Generation Using Deep Convolutional Generative Adversarial Network

Shreyas Kulkarni
University of Illinois at Chicago
1200 W Harrison St
Chicago, Illinois 60607
skulka26@uic.edu

Hengbin Li
University of Illinois at Chicago
1200 W Harrison St
Chicago, Illinois 60607
hli217@uic.edu

Kruti Sharma
University of Illinois at Chicago
1200 W Harrison St
Chicago, Illinois 60607
ksharm22@uic.edu

ABSTRACT

This paper describes a novel project which was implemented to understand if DCGAN's are able to generate novel or similar music to the music samples the DCGAN is trained on. Our paper gives a detailed overview of various approaches to convert sound clips into greyscale images which are then stored in Tfrecords file format which is a native tensor flow file format to store large image datasets. These Tfrecord files are decoded back into .png images and are used to train a DGGAN machine learning model whose generator generates greyscale images. These greyscale images are then unscaled and converted back into unique sound clips generated by the GAN. The outputs observed are quiet interesting and surprisingly good in some cases. We tried three different approaches of generating sound using GAN's with three different music files of different instruments.

CCS CONCEPTS

• **Neural Networks** → *Convolutional Neural Networks*; **Generative Adversarial Network**; • **Generative Adversarial Network** → *Generator*; *Discriminator*; • **TensorFlow** → *TFRecords*;

KEYWORDS

Machine Learning, Deep Convolutional Generative Adversarial Network, Generator, Discriminator, Tfrecord. Greyscale image

1 INTRODUCTION

One of the main aims of our project is to discover and explore a novel way to generate sound/music by using an unsupervised machine learning model called Deep Convolutional Generative Adversarial Networks. Since the paper published by Ian Godfellow on Generative Adversarial Networks in 2014 there has been a rejuvenated interest in generative adversarial networks amongst the machine learning community. Research using generative adversarial networks is booming. Our approach of generating music uses one of a variety of generative adversarial networks called DCGAN which is a very successful approach in generating novel images based on the data set the model is trained on. A DCGAN consists of two networks, a generator which in this case is a de-convolutional network which is used to generate images and a discriminator which is a convolutional neural network which is used to give a probability wheatear the image generated is from the generator or from the training set. For the purpose of this project we wrote our own generator and discriminator to generate and classify greyscale images instead of the conventional DCGAN's generator and discriminator which is used to produce colored images. Our approach of generating sounds explores the idea of converting sound clips into images and training the DCGAN on these generated images which in turn can produce its own greyscale images similar to the images the DCGAN is trained on. The images produced by the DCGAN's generator are then then again converted into sound frequencies and are concatenated to produce a new sound clip which is still generated by the DCGAN. To train the GAN we stored the sound images in .tfrecord files which is a native tensor flow data format. For training the images were decoded from the tfrecord files and passed through the DCGAN model.

2 RELATED WORKS

2.1 Sound CNN

Sound CNN is a technique of classifying sounds using a convolutional network in which sounds are converted into spectrograms and passed through a convolutional neural network which is trained on various spectrograms. This sound CNN classifies sounds based on the pattern and colors of the spectrogram. Understanding this above sound CNN gave us an if converting sound into images [5]

2.2 Sound GAN

Sound Generative Adversarial Network is a paper written in Japanese and was presented at the graduate school of arts and sciences at the university of Tokyo in 2016. Sound GAN is a variation of a DCGAN which is used to generate sound by converting sound to spectrogram using a Q constant transformation to easily recover the sound back from the spectrogram. The DCGAN was then trained on these spectrogram images of sound and the images produced by the generator after training are converted back to sound using fast Fourier transform and other techniques. The results of this technique for generating sounds are encouraging for some types of sounds with low frequencies but not that encouraging for other type of sounds with high frequencies [7]

Based on the success of the above two techniques we decided to implement our version of generating sounds using deep convolutional generative networks. Our process includes the following 5 steps.

(1) Passing a sound file through a band pass filter and sampling a sound file into samples.
(2) Saving the generated samples into grey scale images.
(3) Saving the greyscale images into Tfrecord format.
(4) Train a DCGAN which is capable of generating greyscale images on the generated sound images.

(5) On the images generated by the DCGAN, decode the images back into sound clips and join the clips into a unique sound file generated entirely by a DCGAN.

    Hindrance's: Since the sound data is not of type integer there is some information loss when the sound data is saved into 8-bit greyscale images. In the greyscale image the data is stored as integers between 0 and 255 and this causes loss.

3 DEEP CONVOLUTIONAL GENERATIVE ADVERSARIAL NETWORK

For training the sound images we are going to use a generative adversarial model. Generative adversarial models have the capability to learn the features of a dataset by using an adversarial training technique where one network works against another network to improve itself. In a generative adversarial network one of the networks is called a generator, which is supposed to generate images and the other neural network is called as the discriminator which outputs a probability weather the image passed to it is one generated from the generator or is one of the images from the dataset. The discriminator is optimized in order to increase the likelihood of giving high probability to real data and low probability to generated data. The goal of the generator is to generate images which fool the discriminator into outputting a probability of 0.5 when it encounters an image generated by the generator. This means the generator is successful at generating images from the probability distribution of the training data. As we are training the DCGAN on greyscale images and want the generator to generate images in a greyscale format. We wrote our own generator for greyscale images instead of the conventional generator which generates colored images. We also wrote our own discriminator to out a probability between zero and one to classify greyscale images. Both the generator and the discriminator model are written in tensor flow [5] [6].

3.0.1 Generative Model. The generative model tries to learn the probability distribution of the training data. Given a input image x and its label y the generative model by learning the joint probability distribution p(x,y). In the case of a DCGAN the generator consists of a deconvolution neural network which generates a colored image of size AxBx3 from an input vector of random numbers of fixed length, where A and B are width and height of an image and 3 is the color axis of a colored image [1].

3.0.2 Our Greyscale Image Generator. Since we are saving sound frequency data as greyscale images we wrote our own generator to generate greyscale images with a color axis of 1. Our generator consists of 4 de convolutional layers and each layer has a step size of 2 and a window filter size of 5x5. The depth of our de convolutional layers are 1024, 512, 256, 128 respectively. Our generator generates images of size AxBx1 where A and B are the width and height of the image and one is the color axis. Our generator generates random vector of size 100 having values between 0 and 1. This vector of is reshaped to a shape of (size x size x1024) (depth of 0) before starting deconvolution process. This reshaped vector is then passed through the 4 convolutional layers to produce the greyscale image [3].

3.1 Discriminative Model

The main aim of the discriminative model is to output a probabilistic value to determine if the image it received is from the data distribution of the training set or if the image is generated by the generator. Given an image x and its class label y the discriminator tries to learn the probabilistic distribution of y|x that is p(y|x). The discriminator consists of 4 convolutional layers and outputs a probability of whether the image is a generated by the generator or not. The discriminator gives a high output for real data and a lower probability for generated data [1].

3.2 Our Greyscale Image Discriminator

The discriminator we wrote is based on a discriminator which classifies colored images. Since the images sound samples are stored as greyscale images and the generator we have written we have written generates greyscale images, we wrote a discriminator which is capable of classifying greyscale images. The discriminator is written in tensor flow and python and produces a high probability output if the image sample it sees is from the true training data and a lower probabilistic output if the image is generated by the greyscale generator. Our discriminator consists of 4 convolutional layers which take an image of size AxBx1 as input where A and B are the width and the height of the image and 1 is the color axis of the greyscale image and tries to learn the probability distribution of y(label of the image) given x(the image) p(y|x). The depth of the 4 convolutional layers is 54,128,256,512 respectively. The stride for each layer is 2x2 and the window filter size is 5x5 which is the same as the greyscale generator we have written. The final layer of the discriminator consists of fully connected layer and a soft max activation layer which provides a probability if the image is true image or generated by the generator [3].

4 DATA SET

For this project we did not rely on any dataset instead we generated our own images from sound clips. For our project we took sample clips from the website www.looperman.com. We used three clips [2].

4.1 Piano Sound Clip

This clip was recorded at 44100 frequencies per second. This clip was first limited into 21 seconds in length and then sampled into samples of 1 second. Since the clip was recorded at 44100 frequencies per second the length of each sample is 44100. These frequency values were extracted from each sample and saved into a greyscale image of size 210 x 210. The piano sample produced 21 images each of size 210 x 210. Using the 21 image samples we tried to generate music by training the DCGAN in 2 different ways: Training the greyscale DCGAN on single second of sound which is a single image and training the greyscale DCGAN on all the sample images with a label of piano.

4.2 Drums Beat Sound Clip

This clip was recorded at 44100 frequencies per second at 92 beats per minute. For the drum beat sound music samples were generated in a different way than the piano clip. First this sound clip was limited for length of 21 seconds. This clip of 21 seconds was then passed through a band pass filter to limit the low and high frequencies. As this clip is recorded at 92 beats per minute it will have 32 beats in 21 seconds. This 21 seconds of music has 44100 x 21 frequency data points. These 926100 data of frequencies are sampled into samples of length of each beat. 926100/32. The length of frequencies data for each beat is 28940.625. To convert this data of length 28940.625 into an image the image size will be sqrt(28940) which is 170.11. As the image size has to be an integer we will round of the image size to 170 x170. Thus the length of data frequencies for sample should be 170 x 170 = 28900. Since there are 32 beats in 21 seconds of music the length sound data should be 32 x 28900 =924800. These 924800 data pints of music are sampled into 32 samples each of length 28900 and each sample is then saved into a greyscale image of size 170 x170. The greyscale DCGAN is then trained on these 32 images which are given a label of drum music.

4.3 String Instrument Sound Clip

For the string instrument music, the samples were taken in a similar way to the drum beat music with the length of the sample being nearly equal to the number of frequency data in one beat of music. However, in this case we wanted to have images of size 164 x 164. Since the drum is an instrument with low frequency of sound, we wanted to sample frequency data of a sound file into a greyscale images for an instrument with mostly high frequency of data like a string instrument. We sampled a string instrument clip taken from looperman.com and saved it into 16 greyscale images in which each image was of size 164 x 164 which is less than the size for a single beat. Following are the details of converting the string instrument sound clip into greyscale images of size length 164 x164. This clip was recorded at 44100 frequencies per second at 80 beats per minute. First this sound clip was limited for length of 10 seconds. This clip of 10 seconds was then passed through a band pass filter to limit the low and high frequencies. These 10 seconds of music has 44100 x 10 frequency data points. In this case the we wanted 16 greyscale images size to be 164 x 164. So we clipped the sound into of 10 seconds having 441000 data pints into size of 164 x164 x16= 434666 data points. The length of each sample is 26896. Each sample was saved into a greyscale image each representing length of sound less than one beat.

5 SAVING SOUND IMAGES INTO TFRECORDS DATA FORMAT

Tfrecords is a simple way to prepare a large scale image data set. Tfrecords is a native tensor flow format in which we have saved the greyscale images of sound data mentioned in the data section. Before passing the data to the DCGAN for training data stored in the Tfrecords format is decoded back into the greyscale images [4]. To convert the images into the Tfrecords we have used an already existing repository. This repository stores the following data of each image in the Tfrecord file.

(1) Encoded Image: The field stores the encoded image data. These are the frequency values of the sound.
(2) Image Format: This field stores the image format in our case the images have been stored in .png data format.
(3) Height: The height of the image, for the piano music sample in which the data has been samples in for one second the height is 210. For the drum beat sample music which is sampled per beat time length, the height is 170 and for the string music sample which is also sampled per beat length its 164.
(4) Width: The width of the image, for the piano music sample in which the data has been samples in for one second the width is 210. For the drum beat sample music which is sampled per beat time length, the height is 170 and for the string music sample which is also sampled per beat length its 164.
(5) Image label: label assigned to the image.

When creating a Tfrecord file from the image we provide certain parameters like number of shards to spilt the data into and the output file name. For the purpose of our experiments we spilt the data into 2 shards and the output file name was given according to the data being converted. Each music file data is converted into its own Tfrecord file.

6 GENERATING MUSIC IN DIFFERENT WAYS FROM DIFFERENT MUISC FILES

We have tried to generate sound using DCGAN's in different approaches based on the way the sound data is sampled and stored in greyscale images. As mentioned above we have used 3 music files and the data from each of these files is used to generate music/sound in a different way. Each music file is a .wav file downloaded from looperman.com. For each file we extracted the frequency data from the .wav file. The frequency data was sampled per second or per beat of time depending on the frequency and beats per minute of the .wav file and each sample which contains frequency data is then saved into a greyscale image. Since the frequency data from the music file are floating point values and a greyscale image stores data in from the range 0 to 255 integer values there was some data loss when the sound frequency data is normalized and saved as greyscale image. While normalizing the data for normalization the domain is taken as maximum and minimum frequency data values present in the sample. Following are the different approaches we have followed to generate music from the three music files.

6.1 Approach 1 To Generate Music

For approach 1 to generate music we used the piano music data to generate samples. The piano music sound file is a .wav sound file. To sample the music file, the frequency data from the .wav file is sampled into contiguous samples of one second each. These one second samples of music then were saved as greyscale .png images. Figure: 1 shows the greyscale image generated from the first second of the music file.

In this approach we first tried to train the DCGAN on the same one second sample of music to see if the GAN was able to generate the same or similar sample of music.

For training we used the image generated from the first second of data and made 21 copies of the image. Each image had the same label. These 21 greyscale images all representing the first second of piano music data was saved into 2 Tfrecord files splitting them into two shards. Each file had its own validation file. Since the piano music was recorded at 44100 frequencies per second the size of each image is 210 x 210. These images were used to train our

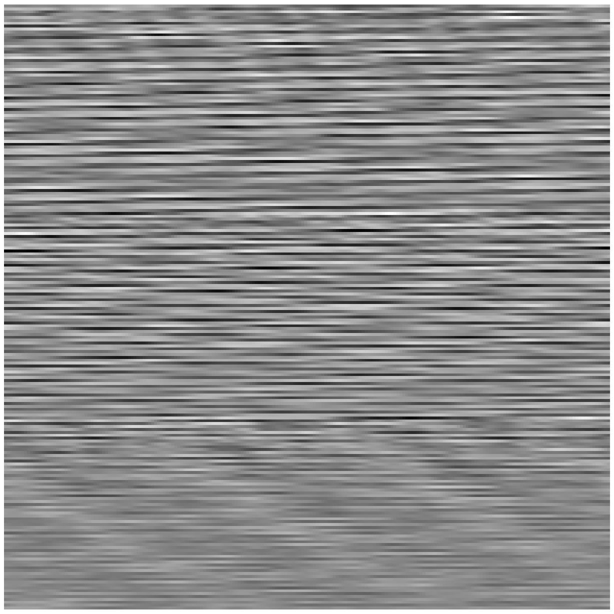
Figure 1: Greyscale Image Generated from Sampling the First Second of the Piano Music Data. Size: 210x210

DCGAN which we wrote for generating greyscale images. Since our GAN is unable to generate images of size 210 x 210 we set up our DCGAN to generate image of size 208 x 208 which is approximately equal to one second of music when sound is recovered from the greyscale image. The GAN was trained on the images of first second for over 8000 iterations. The images produced by the GAN were then reshaped into a 1d tensor and then reconverted into sound by after descaling the image values ranging from 0 to 255 into a range ranging from sample minimum value to sample maximum frequency value. Figure: 2 shows the image generated by the GAN after 100 iterations and Figure: 3 shows the image generated by the GAN after 10000 iterations after being trained on images generated from the first second of piano music.

Once this image was converted back into sound we got pretty impressive results. The sound recovered from the image was very similar to the sound of the first second of piano music. The recovered sound did have some noise ingrained in it.

Similarly, we trained our DCGAN for images generated from second 2, second 3 and second 4 respectively, that is the GAN was trained on images of second 2 and then images of second 3 and images of second 4 separately to produce sounds similar to second 2, second 3 and second 4. Since the piano music is a loop of 4 seconds we only trained the GAN separately for the images generated from first four seconds of the piano music.

Figure: 4 shows greyscale image generated from second 3 of piano music.

Figure: 5 shows the image generated by the DCGAN after training for 100 iterations on images generated from second

Figure: 6 shows the image generated by the DCGAN after training for 10000 iterations on the image generated from second 3 of piano music data.

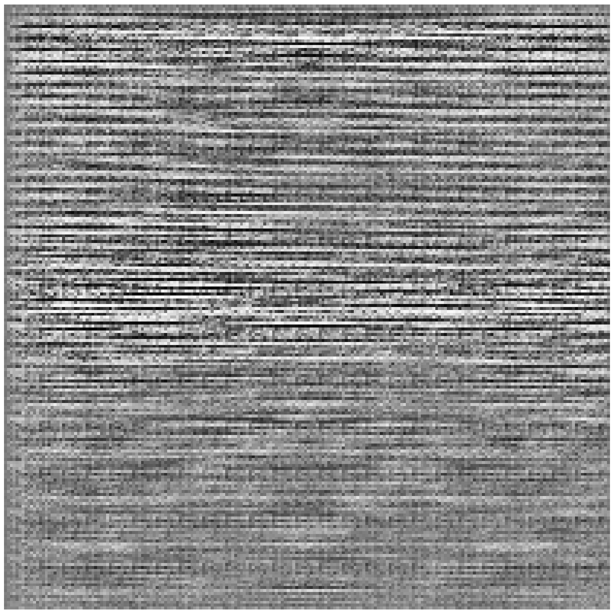
Figure 2: Image generated by the DCGAN after being trained on image generated from 1st second of piano music after 100 iterations. Size 208 x 208

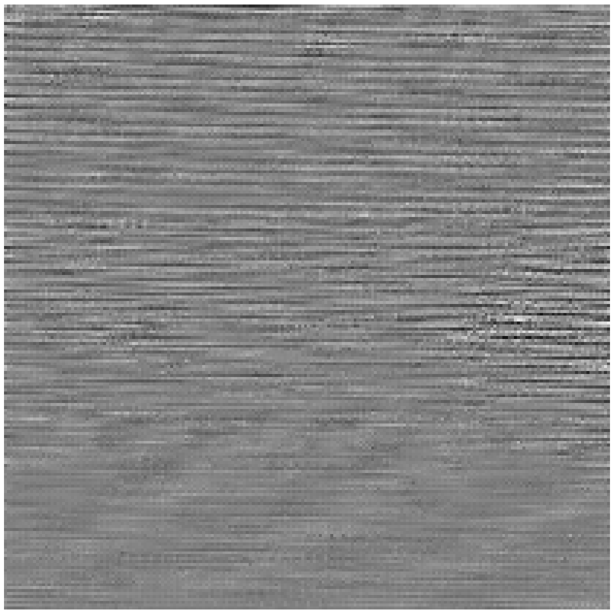
Figure 3: Image generated by the DCGAN after being trained on image generated from 1st second of piano music after 9300 iterations. Size: 208 x 208

On training the DCGAN for 10000 iterations for images of second 2, images of second 3 and images of second 4 separately the DCGAN produced the images, when unscaled and converted back into 1 second sound clips produced sounds similar to sounds herd in

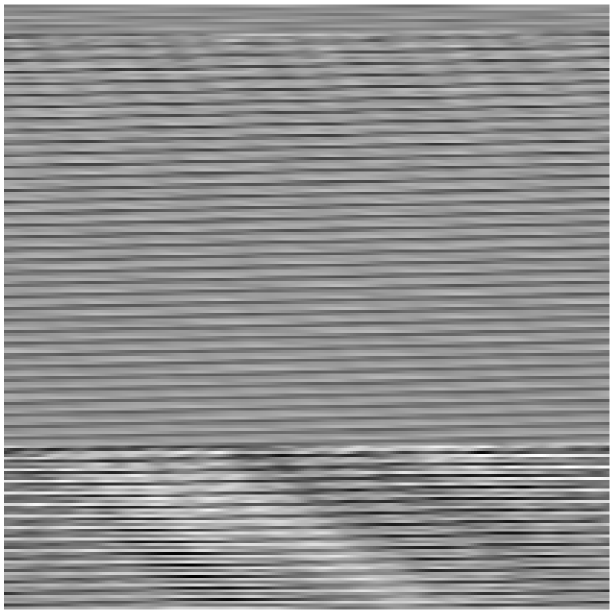

Figure 4: Greyscale Image Generated from Sampling the Third Second of the Piano Music Data. Size: 210 x 210

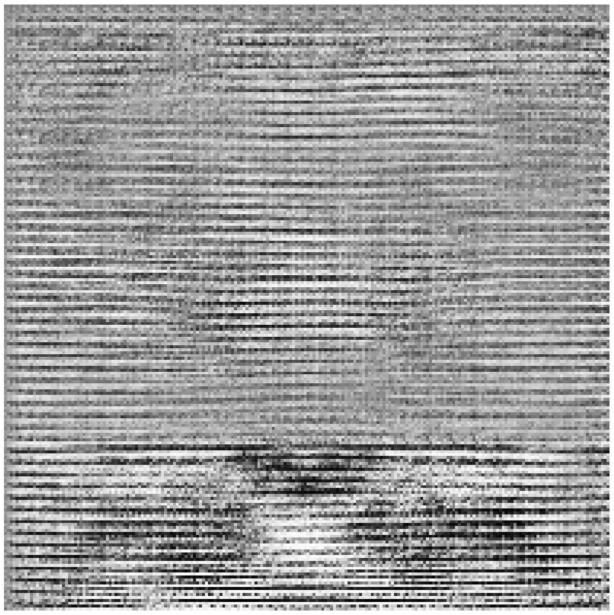

Figure 5: Image generated by the DCGAN after being trained on image generated from 3rd second of piano music after 100 iterations. Size: 208 x 208

second 2, second 3 and second 4. The recovered sound from the images was pretty accurate and had little noise due to the loss of data when scaling frequency into a greyscale image values of 0 to 255. The sound produced by the DCGAN while training on each second was then concatenated into a single clip of approximately 4

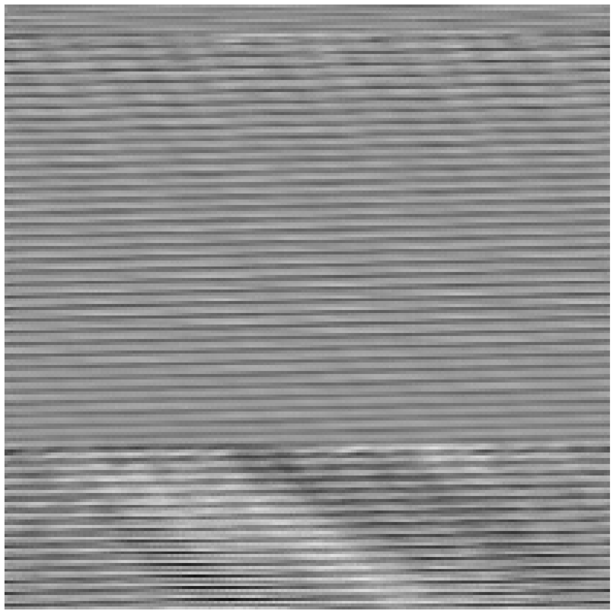

Figure 6: Image generated by the DCGAN after being trained on image generated from third second of piano music after 10000 iterations. Size: 208 x 208

seconds. This 4 second sound clip mimics the original 4 seconds of the piano sound clip and has some noise associated with it.

6.2 Approach 2 to Generate Music

In approach 2 of generating music we used drum beat music data file. The main reason for using drum beats is because drums produce lower frequency sounds than string instruments and we wanted to understand if the frequency range makes any difference to a DC-GAN while producing sound. Another difference in this approach as compared to the first one is the drum beat music is, the music is clipped into time lengths equal to time of one beat. We first clip the drum beat music to 21 seconds. Since the music is recorded at 92 beats per minute the 21 seconds of drum beat music is spilt into 32 beats. The frequency data for each of these 32 beat samples or clips is then saved as a greyscale image. Also unlike approach 1 the GAN is now trained on all 32 image having labels drums instead of on a single second sound image as in approach one. These 32 images which represent 32 beats are then saves as two Tfrecord files and are used to train our DCGAN. Each image is of size 170 x 170 pixels. We clip some frequency data to fit the frequency data into the image which has integer length and height. This reason for the clipping is explained in the Dataset section: 4.2.

In order to train these images on our DCGAN model we adjust the parameters of DC GAN to produce images of 160 x 160 which is slightly lesser than the image size for one beat of 170 x 170. We do this because our GAN is unable to generate images of size 170 x 170 and the closest size the GAN can generate is 160 x 160. Since the GAN is trained on the entire drum beat sound clip the aim was to see if the GAN can produce unique sounds in the style of the drum beats. This GAN was trained on for 10000 iterations on the

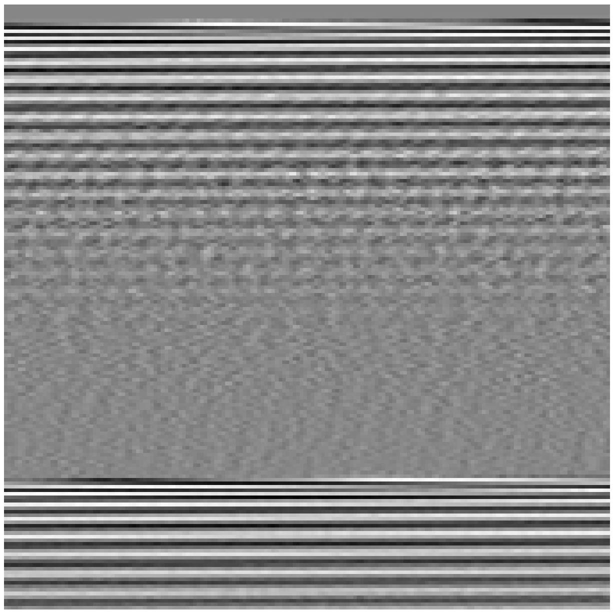

Figure 7: Image of the drum beat sound clip converted into a greyscale image (First Beat) Size: 170 x 170

images of the drum beats sound clips. We took 4 images generated by the DCGAN of size 160 x 160 and then converted the image data into sound clips. These sound clips were concatenated to produce a sound file approximately equal the sound of length equal to 4 beats. We observed that the sound generated was noisy and was not as clear as the outputs of the approach 1. However, the sound did have certain characteristics of the beats of the drum present in the drum beats sound clip. Figure: 7 shows an image of the drum beat sound clip converted into a greyscale image. Figure: 8 shows the image generated by the DCGAN after training on the images generated of the drum beat sound file for 8300 iterations.

6.3 Approach 3 to Generat Music

In approach 3 we wanted to try sound generation on sound clips having a high frequency data and see if the DCGAN is can provide better results on sound having a higher frequency. For this approach we took a string instrument sound clip form looperman.com. The sound clip was 12 seconds long. This sound clip was clipped to 10 seconds and then passed through a band pass filter to limit the frequencies. For this approach we wanted to save the frequency data of the sound into 16 images of size 164 x 164. Thus the sound file was converted into 16 greyscale images. In order to do this the frequency data of the sound file was clipped from 0th index to 430336 index which is equal to 164 x 164 x 16. This clipped data was the saved as 16 greyscale images. Figure: 9 shows an image saved after converting one of the clips of the string instrument sound file into a greyscale image.

These 16 images were then saved into two Tfrecord files. Each file having 8 images. Each image is labeled with a 'strings 'label. Our DCGAN is then trained on these images generated from the string instrument sound clip for 20000 iterations. The DCGAN's

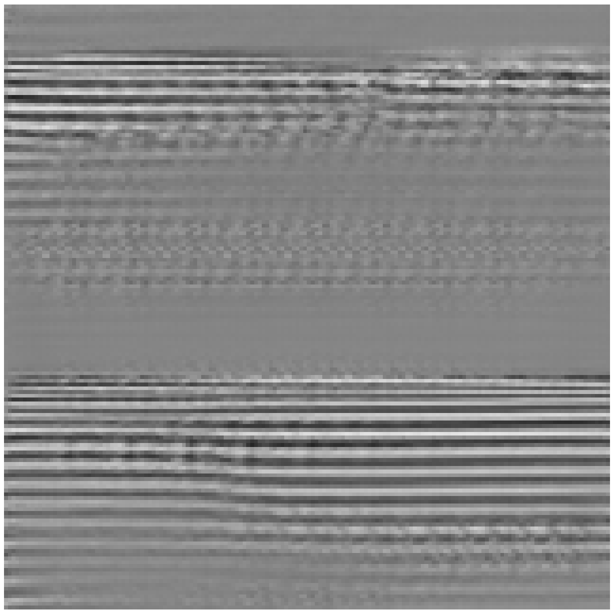

Figure 8: Image generated by the DCGAN after training on the images generated of the drum beat sound file after 8200 iterations. Size 160 x 160

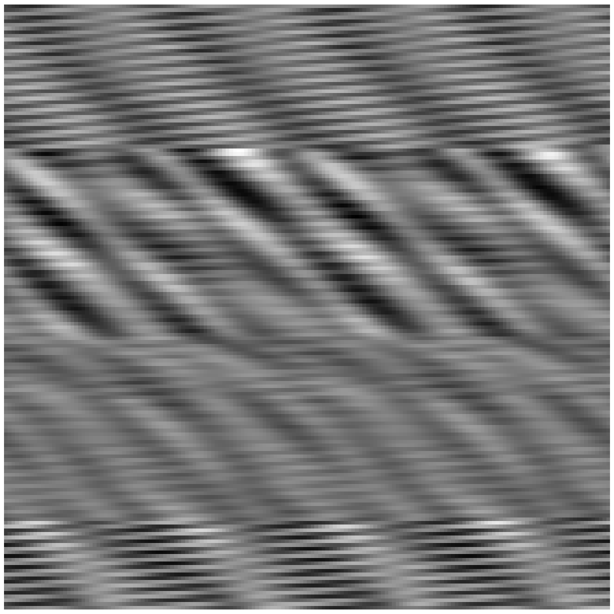

Figure 9: An example of an image saved after converting the string instrument music file into greyscale images. Size: 164 x 164

parameters are adjusted such that the generator produces images of size 160 x 160 which is the closest size the generator can generate to 164 x 164. Few of the images generated from the generator are taken and reshaped into a 1D tensor and saved as sound clips.

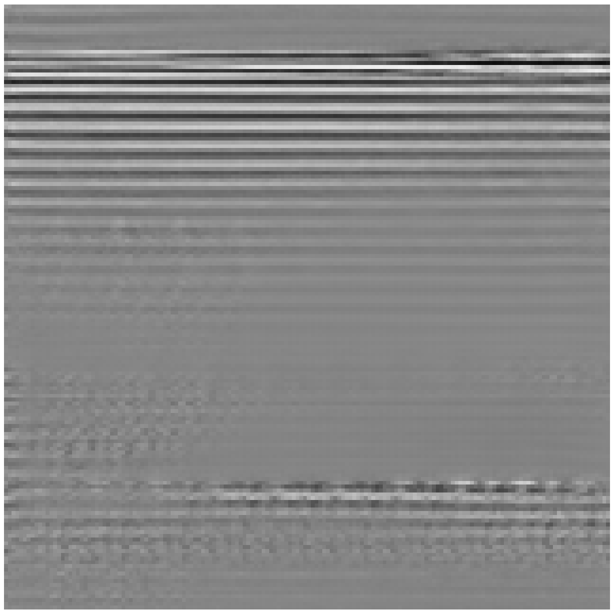

Figure 10: An example output of the generator after being trained on string music sound file after 8300 iterations. Size: 160 x 160

These sound clips are concatenated into a larger sound clip. The sound generated from the generator when it is trained on the string instrument music file does have noise associated with it and is a little unclear. However, the recovered sound is unique and does have some underlying characteristics of the string instrument sound clip in the way the notes are played. Figure: 10 shows an output of the generator after being trained on the string instrument music sound file.

7 DIFFERENT METHODS TRIED TO REMOVE THE NOISE ASSOCIATED WITH GENERATED SOUND

One of the issues we faced when sound was recovered from the images produced from the DCGAN was noise being associated with the sound. We tried various ways to reduce the noise associated with the recovered sound.

(1) One of the approaches to remove noise was to take the image values which represented frequencies of the sound and convert the data into a spectrogram. Sound was then recovered from the spectrogram. This technique did not result in any improvement of sound quality.
(2) This way of reducing noise was tried for approach 2 and approach 3 since the DCGAN is trained on the entire music file in these two approaches. After extracting the frequency data from the sound file, the frequency data are floating point values. When converted into greyscale images the floating point values are scaled into a range of 0 to 255 from the domain of highest and lowest frequency values from the clip. Since the DCGAN is trained on the images generated from the entire sound file. The images were scaled into a range of 0 to 255 from the domain of highest and lowest frequency values of the entire sound file instead of highest and lowest frequency values of the clip itself. The values of the image generated by the DCGAN is then rescaled back to the range of highest and lowest frequency values of the sound file. The sound is recovered from these rescaled values. This approach also produced noisy sound for approach 2 and approach 3 and did not de noise the produced music.
(3) For Approach 3 we tried to train the DCGAN for a higher number of iterations from 10000 to 50000. However, there was no significant improvement and the GAN ended up overfitting.

Since the methods of reducing noise did not produce any additional benefits following is the method followed for saving sound frequency data as images and recovering the sound from the images generated from the generator. While saving the frequency data into greyscale images the frequency data is scaled into a range of 0 to 255 from a domain of minimum and maximum frequency values of the particular sample clip being converted into the image. When recovering the sound back from the greyscale image the image values are read and converted back into sound without rescaling into a range of minimum and maximum frequency values of the sound file or particular sound clip.

8 RESULTS

We tried to generate sound using deep convolutional generative adversarial networks using three different approaches. Our results were encouraging for all the three approaches but we did get the best results using approach 1. Approach 1: The sound produced almost mimicked the original training data sound for approach 1 and had little noise associated with the generated sound. For Approach 2 and 3: The sound generated from the generator when it is trained on the string instrument and drum beat file does have noise associated with it and is a little unclear. However, the recovered sound is unique and does have some underlying characteristics of the sound in the training data in the way the notes are played.

9 FUTURE WORK

Our project has a lot of scope for future work. We have only tried to generate sound using DCGAN on a single music file of data having one label. We can try and train our DCGAN on multiple music files have various labels. Since there is a lot of noise associated with the outputs when the model is trained on the entire music file there is a large scope to look into demonising techniques to reduce the noise in the sound generated by the DCGAN. Another area where there is scope for improvement is scaling and descaling the frequency data when it is stored into a greyscale image and recovering the frequency data from the greyscale image without loss of decimal values. Also We have tried to generate sound by saving frequency data as greyscale images, one can look into other ways of generating sounds using DCGAN by converting sound into spectrograms and training the DCGAN on spectrogram to generate spectrograms from which sound can be recovered.

10 CONCLUSIONS

Our various approaches in generating sound from music files gave us mixed yet encouraging results. The outputs recovered from approach one are clear and the sound mimics the input sound and has little noise. While the sound produced from approach 2 and 3 have more noise associated with it and are unclear. However, the outputs from approach 3 do have some characteristics of the training data in the way beats are played. Our various approaches for reducing noise from the outputs were not very encouraging but we are sure there are a lot of other ways to de noise the outputs from approach 2 and approach 3 that we have not explored yet. The outputs from approach 3 are better than those of approach 2 which may mean the DCGAN performs better on data with high frequency. Since the outputs from approach 1 are really good and approach 3 are encouraging we are hopeful future work on this technique or similar technique will produce better results.

A APPENDIX

A.1 Introduction

A.2 Related Works

A.2.1 Sound CNN.

A.2.2 Sound GAN.

A.3 Deep Convolutional Generative Adversarial Network

A.3.1 Generative Model.

A.3.2 Our Greyscale Image Generator.

A.3.3 Discriminative Model.

A.3.4 Our Greyscale Image Discriminator.

A.4 Data Set

A.4.1 Piano Sound Clip.

A.4.2 Drum Beat Sound Clip.

A.4.3 String Instrument Sound Clip.

A.5 Saving Sound Images into Tfrecords Data Format

A.6 Generating Music in Different Ways from Different Music Files

A.6.1 Approach 1 to Generate Music.

A.6.2 Approach 2 to Generate Music.

A.6.3 Approach 3 to Generate Music.

A.7 Different Methods Tried to Remove the Noise Associated with Generated Sound

A.8 Results

A.9 Future Work

A.10 Conclusions

A.11 References

ACKNOWLEDGMENTS

We would like to thank Professor Angus Forbes for contributing significantly to the success of this project and for giving an interesting challenging and inspiring topic to work on. His guidance and encouragement to allow us experiment with such a unique machine learning model on a very interesting idea such as sound generation using DCGAN proved extremely valuable while implementing our idea.

REFERENCES

[1] bamos.github.io 2016. *Image Completion with Deep Learning in TensorFlow*. bamos.github.io. https://bamos.github.io/2016/08/09/deep-completion/.

[2] Future Web Services 2014. *LOOPERMAN Pro Audio Resources and Musicians Community*. Future Web Services. https://www.looperman.com.

[3] GitHub 2016. *Sugyan: Face Generator*. GitHub. https://github.com/sugyan/face-generator.

[4] GitHub 2017. *kwotsin : create_tfrecords*. GitHub. https://github.com/kwotsin/create$_t$frecords.

[5] Medium Corporation 2016. *Generative Adversarial Networks Explained with a Classic Spongebob Squarepants Episode*. Medium Corporation. medium.com/@awjuliani/generative-adversarial-networks-explained-with-a-classic-spongebob-squarepants-episode-54deab2fce39.

[6] Alec Radford, Luke Metz, and Soumith Chintala. 2015. Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. *CoRR* abs/1511.06434 (2015), 1–301. http://arxiv.org/abs/1511.06434

[7] The 30th Annual Conference of the Japanese Society for Artificial Intelligence 2016. *SNDGAN: The generative model of sounds using Generative Adversarial Networks*. The 30th Annual Conference of the Japanese Society for Artificial Intelligence. https://kaigi.org/jsai/webprogram/2016/pdf/951.pdf.