The Art of Transcription: From Music to Notes

Manuel Mínguez Carretero
Radboud University, Nijmegen, The Netherlands
M.MinguezCarretero@student.ru.nl

ABSTRACT

The use of deep learning is constantly growing, new complex applications are being solved thanks to the use of neural networks. For example, audio signals is a type of data that is being used in neural networks to achieve new solutions. The aim of this project is to study to what extent a neuronal network is able to transcript polyphonic musical notes from an audio signal. Using Musicnet dataset as the initial corpus, a sound signal pre-processing technique has been applied to create a new representation from the time domain to the frequency domain using Constant-Q Transforms (CQT), a variant of the well-known Fast Fourier transforms to encode a richer representation of the data for the neural network. Consequently, from the new representation of the signal, images have been produced to constitute the input data of a Convolutional Neuronal Network (CNN). Results show that the model struggles to generalise for the test set due to the complexity of the songs and the abundance of possible musical notes that can be played at the same time. Nevertheless, the model is able to achieve a 23% accuracy predicting successfully the notes that appear in the song.

1 INTRODUCTION

Music plays a mayor role in our lives and now with the global access to mobile phones, music is being played more than ever. The aim of this research is to study to what extent is it possible to transcript from a raw song to the musical notes that appear in it. The aim is not to just predict the notes that have been played in the entire song but to classify in a small time frame the songs that have been played, so the model is able to tell where and when each music note appears in a specific song.

This is an interesting study as the translation of music into its notes can be used in a variety of applications. For example, as a way of learning to play an instrument where users could scan a song to see what notes are being played and when. Also as a tool for artists to transcribe the music they are playing into musical scores in a much faster way. Or, for example, it can be further expanded to use the notes played as input for another model to predict new solutions.

This article goes through the whole process from raw data to the pre-processing of it to create a better representation of the input data to finally design a neural network that is able to accomplish such research question.

The article begins by describing different studies that have been carried out with neural networks that are important for the development of this thesis. Specifically, studies related to Automatic Speech Recognition (ASR), in which audio signals have been used as data to carry out their studies and Music Transcription, the key field of this study.

Next, it is explained how the data is used for the development of this research. How is the initial state and why it is necessary to apply transformations from the time to the frequency domain. Examples will be shown to support the claims.

The next section of this article deals with the neural network used to predict the musical notes that appear in the song. This is followed by a discussion of why the neural network has to be the way it is and what points should be avoided to prevent bad results.

Finally, the article concludes with a discussion of the results and a section summarising the whole process and the results obtained.

2 RELATED WORK

Music transcription using neural networks is a field that lacks of research, even though there are several universities and important enterprises that work on it, the number of papers published about the topic represents a huge minority comparing to other studies such as artificial vision or Natural Language Processing. Google for instance, has a team to work on developing new deep learning and reinforcement learning algorithms for generating songs [1].

Automatic Speech Recognition (ASR) is data-hungry, and using more and more data to tackle a problem tends to help performance but poses new challenges. In addition, the high number of languages on the planet is an added problem since each language may require a different model. Not all languages are structured and use the same sounds so it is sometimes necessary to do a study using a specific language. For instance, [6] presents a multidialectal corpus approach for building a text-to-speech voice for a new dialect in a language with existing resources, focusing on various South American dialects of Spanish. Using a multidialectal corpus helped them to obtain better results, being able to generalise better. Or [8] for Gujarati, Kannada, Malayalam, Marathi, Tamil and Telugu Speech Synthesis Systems.

The same philosophy has been used in this study. For the training of the neural network, songs of different genres have been used in which different musical instruments participated. The objective is to discover which musical note has been played regardless the instruments playing.

For music transcription, the paper [10] tries to entangle the "glass ceiling effect" which consist of the fact that the neural network does not learn how to generalise a musical note but to learn combination of them. Thus, when the network sees a new combination of notes, it is not able to recognise them properly. To solve it, they propose several networks to reduce this effect. Their results show that some networks are able to generalise better than others. Nevertheless, their results show that further studies should be performed to prevent this problem.

Other studies such as [2–4] also investigate in polyphonic music to solve different problems.

ASR 2020, Radboud University, Nijmegen, Netherlands

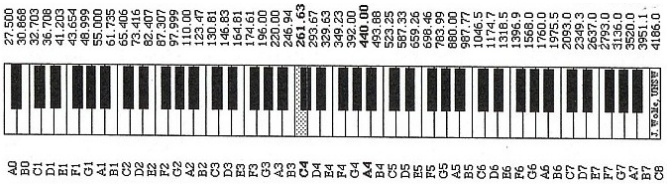

Figure 1: A piano keyboard with its 88 keys. It is shown the wave frequency length that each note produces and its correspondent name.

start_time	end_time	instrument	note	start_beat	end_beat	note_value
9182	90078	43	53	4.0	1.5	Dotted Quarter
9182	33758	42	65	4.0	0.5	Eighth
9182	62430	1	69	4.0	1.0	Quarter

Figure 2: Sample of the information that can be found in a label file.

3 DATA

This section describes the data used perform the music transcription. The data can be found in [1]. MusicNet is a collection of 330 labelled songs with varying lengths. The labels indicate 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. As we can see in the Figure 2, there is multiple data information that can be used but for the sake of efficiency only the necessary columns where used to predict the musical notes. Specifically, the information regarding to the start time and end time of the note was used to know the windows of the spectrogram where that specific note was present. Furthermore, the id of the music note played represents its value in a Musical Instrument Digital Interface (MIDI). The MIDI is a standard protocol that allows computers, musical instruments and other hardware to communicate [7]. In this case, it is used as a way to represent all the possible musical notes that can be played by any instrument. A MIDI in a piano keyboard has 88 keys where contiguous keys share similar frequencies. As we can see in the picture 1.

For the sake of simplicity, it was decided to leave out the instrument ids since the goal of this study is to observe to what extent a neural network is able to transcript from sound signals to musical notes regarding the instrument.

The dataset can be downloaded on three different formats, raw data stored in CSV files, data stored in numpy arrays or a HDF5 file. All of them containing the same information. For this project, even though the use of HDF5 files is recommended due to their good memory handling, for this project has been decided using the CSV formats due to the fact that can be highly paralleled using Pytorch Dataloaders [12]. The Section 3.2, describes in depth how the data is handled for the proposed model.

[1] https://homes.cs.washington.edu/ thickstn/musicnet.html

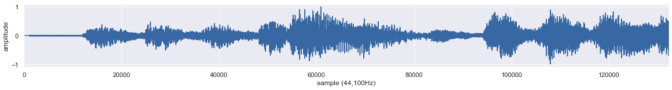

Figure 3: Plot of the first three seconds of the song 2416 from MusicNet represented in its raw form.

3.1 Preprocessing of the data

The initial dataset contains 330 songs, 320 of them constituting the train set and the ten remaining used as a test set (selected a priori by the creators of the dataset). The reason why those 10 songs where selected as the test set is because they contain a balanced features and labels.

Each song, stored in a WAV format, has its corresponding labels presented in the Figure 2. The songs are stored using their raw sound signal represented in the time domain with a sampling frequency of 44.100 Hz. The Figure3 sheds light on the evolution in time of the song, where the Y axis represents the amplitude and the X the time. This representation of the sound signal lacks of information for the neural network since it is not possible to distinguish the music that is actually playing, what music notes are present or other relevant information. If this raw data were used as an input for a neural network, it is very likely that it would never converge and thus, it would never be able to obtain positive results.

The state of the art results in automatic speech recognition (ASR) and other fields that deal with sound signals as the input data for neural networks proved that converting the signals from the time domain to the frequency domain has a much better representation of the sound and thus, the network is able to extract important features from such data, obtaining better results [5]. In this study it has been succesfully implemented the following transformations:

- Fast Fourier Transforms (FFT)
- Filter Banks
- Mel-frequency cepstral coefficients (MFCCs)
- Constant-Q transforms (CQT)

The recent paper [9] makes a comparison of time-frequency representations for two different classification problems. Even though it shows that MFCC outperforms the other possible representations this is not the case for this problem. Using Filter Banks and MFCC the network was not able to generalise the important features from the text, leading to a poor performance due to the fact that the network was getting lazy predicting all zeros since it is the most common label (representing that a note is not being played).

Both FFT and CQT obtained similar behaviour when training the model. The difference is that the CQT instead of using 2048 values to encode a window, it uses 384 which reduces considerably the size to represent a similar information. This leads to faster data processing, data loading and most importantly, faster training. That is why CQT was chosen among the time-frequency representations candidates. Because the network is able to interpret the input data to learn how to classify the musical notes and because it reduces the spatial complexity compared to the FFT.

The Figure 4 shows an example of how a spectrogram looks like when transformed into its CQT form. This picture shows very valuable information. We can see that it is sort of a 3D image where

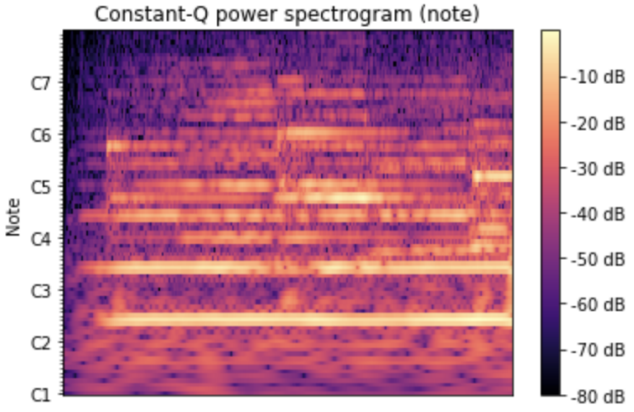

Figure 4: Spectrogram of the first 15 seconds of the song 1727 from Musicnet. Note the colours in the spectogram show the intensity in - DB, the brighter it is, the more intensity represents.

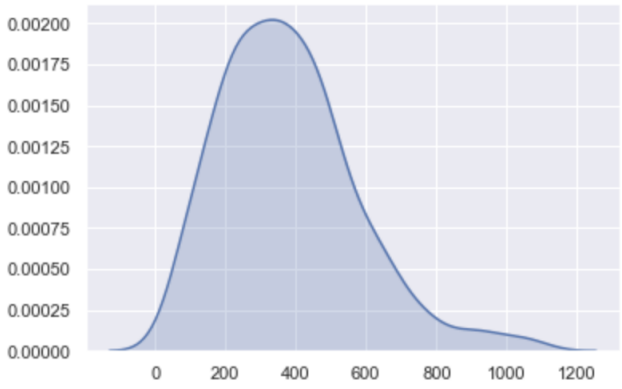

Figure 5: The X axis represent the time length in seconds.

it is encoded three different types of information. Along the X axis we can see how the song varies in time while the Y axis tells use the frequencies. Finally, the colours in the spectrogram itself reveals the maginiture of a Y_i frequency at a X_j time.

The figure 5 shows the distribution of the song lengths. Even though it is not possible to know the exact value from the Figure, the mean length is 379 seconds and σ 194 seconds. This information has been taken into account to build the Pytorch Dataloader, since it takes slides of each song to produce the input data of the CNN. This information helps us to know how many times we should iterate trough the Dataloader to create images using as much new data as possible.

3.2 Data Loader

Once the data has been pre-processed transforming it from the time to the frequency domain we obtain the so-called spectrograms with an output shape of (N, 384, 1) being N the number of windows that

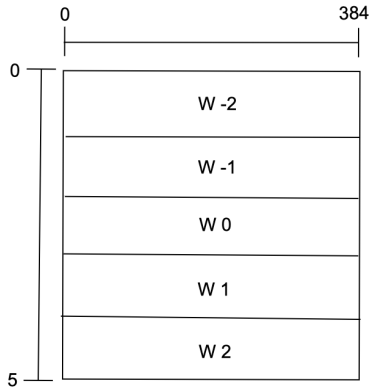

Figure 6: Image created from the CQT spectrogram. There are 5 windows staked together where each window contains 384 values. The image size is (5x384).

the song has encoded. One second is encoded in 86 windows so each window represents to 0.0012 seconds of a song.

In order to show come context to the neural network, it has been stacked 5 windows together creating images of shape (5, 384,1) being 5 the number of windows, 384 the information encoded per window and 1 the number of channels. As we can see, this data has a similar representation as an image. Therefore, the use of Convolutional Neural Networks are highly recommended for tasks that require this type of input data.

The Figure 6 shows an example of how the input data looks like. The main window of the image is the central one, so the labels that correspond to the image are the musical notes that appear in the Window W_0. The idea behind using front and back windows from the main one is to give some context to the neural network. After some training, the CNN should be able to focus the activations that appear in the middle part of the image, discarding unnecessary features from the other windows and using the information that is useful to classify the possible musical notes.

The problem of creating images stacking 5 windows together from the spectrogram is that the memory required to load all the dataset is unfeasible. Two efficient methods have been used to prevent this issue. The first way of saving memory is by referencing the location of the windows from other images. In other words, each image is created by stacking 5 windows together, then the algorithm shifts one position and creates a new image. By shifting only one window, the new image instead of creating new copies of the windows, it references to the ones already assigned in the previous image, saving a significant amount of memory to be loaded.

The second and more useful approach is using Pytorch Dataloaders, which are used to create the images from the spectrograms on the fly to train the CNN. So the algorithm will create a batch of images that will be used as the input data of the CNN and once the model has been run using this data, it is removed from the main memory to load a new batch of new images. This process not only saves memory since we only use the necessary data to train the model at a time, but also saves time since it is possible to parallelise

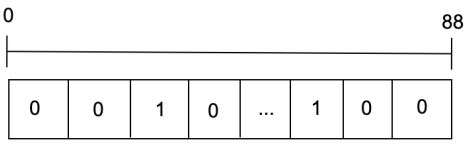

Figure 7: A multi-hot encoding vector representing the appearance of the 88 possible musical notes. Note that there can be multiple ones representing various notes being played in the same window.

using N number of workers who will create batches of images in parallel, that is why the dataset has been produced on separated CSV files per song.

Each song has a variable length thus the Dataloader takes slices from the spectrograms to create always the same number of images per song each epoch which constitutes the batch size.The slices are taken randomly depending on the size of each song. Therefore, it is required multiple number of epochs to show the whole available data to the CNN while training. For instance, imagine the song "Wish You Were Here" had 300 windows in total, if the slices had a length of 100 windows, it would be necessary at least 3 epochs to use all the data.

Other approaches where considered but rejected since this approach is able to train the model using the same batch size of images (100 in the previous example) and it loads only one song per thread which as it has just been mentioned, it is possible to create the images from the espectrograms in parallel.

The labels correspond to the central window (W_0) and the possible musical notes that can be played are encoded using a multi-hot encoding which means that each possible note has a value in a vector where 0 represents the absence in the image and 1 that the note is being played. Let be V the possible musical notes that exist, x_i a note played in the the song where $x_i \in \{0,1\}^{|V|}$. The output multi-hot vector consists of a vector $x = [x_1, x_2, ..., x_n]$ where n is the length of V. In this case, as there are 88 possible musical notes at the same time, the label that corresponds to a input image is a vector of 88 binary values (See Figure 7).

The fact that the labels are represented in multi-hot vectors means that the complexity of the neural network is affected. First of all, the large number of possible musical notes makes most of the numbers encoded in the vector zeros, but not necessarily. Which leads us to the next problem, as there are so many possible musical notes, certain notes appear more often than others, and most notes do not appear enough, causing underfitting. In this case underfitting can be controlled using class weights, but after making some tests it did not yielded very promising results since the labels are too sparse.

4 MODEL

This section describes the model that was finally used to transcript polyphonic musical notes. The model architecture is described in the Table 1. The network has been named *LongCNN* and it is an extension of the network SmallConvNet presented in [10]. The network consist of two differentiated parts, the convolutional layers and the linear layers. First, the network convolutes over the images trying to keep the time dimensionality, which is the third dimension. The reason why it is desired to keep the time dimensionality is to be able to extract the context information as much as possible. If we shrink the data too quickly, we would lose the possibility of generalising using that information. That is why, the first convolutional layer uses a "same" padding and the MaxPool layers are 1x2, so they only shrink the frequency dimension by 2.

After the convolutions, the features of the musical notes should be encoded into tensors. These tensors have to be flattened into a 1-Dimension vectors in order to use them as input for the linear layers. To prevent overfitting, Batchnormalization and Dropout layers have been used.

Finally, the last layer contains 88 nodes, representing each possible music note that can appear from the input data, followed by a Sigmoid function, which gives the probability between 0 and 1 of each possible musical note.

Other approaches have been tested. For instance, a model with just Linear layers. This approach could not capture enough information since the input data were not longer images but just the different windows of the spectrogram.

Moreover, a simpler version of the proposed architecture was also tested, obtaining similar but slightly worse results.

The following section presents the results obtained and discusses whether these results can be considered satisfactory or not.

5 RESULTS

The final model was run using Ponyland Server, most specially, using Twist server thanks to their two Cuda GPUs. To run the experiment, the batch size selected was 2048 images, rather a large number but it is necessary to take into account that each song produces an average of 32.594 windows (See Figure 5).

The optimiser chosen is Adam [11] due to its self learning rate management. This optimiser has proven many times to be the most effective for multiple applications.

The loss selected is the BCE Loss. It is a criterion that measures the Binary Cross Entropy between the target and the output and it can be described as:

$l(x, y) = L = \{l_1, ..., l_n\}^T, l_n = -w_n[y_n \cdot \log x_n + (1 - y_n) \cdot \log 1 - x_n)]$

Where N is the batch size.

The loss decreases consistently after backpropagating each batch. The Figure 8 shows the evolution of the loss in the training process. Which shows that the network is able to learn how to transcript musical notes for the train set.

Nevertheless, similar conclusions as the paper [10] has been reached. The model struggles to generalise properly for the test set. It is not entirely a problem of everfitting or underfitting, the problem relies in the fact that the network learn how to predict combination of music notes detected from the spectrogram. Consequently, in the test set there can be new combinations of contiguous notes confusing the network in the test phase.

Nevertheless, the network generalises a bit more after each epoch until it reaches around 20% accuracy. The Figure 9 shows the accuracy obtained after each epoch in the test set. We can see that at the

Layer	Output shape	Parameters
Input	(Batch, 1, 5, 384)	
Conv2D	(Batch,16, 5, 384)	Padding = 1, Kernel = 3x3
Relu	(Batch,16, 5, 384)	
Batchnorm2D	(Batch,16, 5, 384)	
MaxPool2D	(Batch,16, 5, 192)	Size = 1x2
Dropout	(Batch,16, 5, 192)	p = 0.25
Conv2D	(Batch,16, 3, 190)	Kernel = 3x3
Relu	(Batch,16, 3, 190)	
Batchnorm2D	(Batch,16, 3, 190)	
MaxPool2D	(Batch,16, 3, 95)	Size = 1x2
Dropout	(Batch,16, 3, 95)	p = 0.25
Conv2D	(Batch,32, 3, 93)	Kernel = 1x3
Relu	(Batch,32, 3, 93)	
Batchnorm2D	(Batch,32, 3, 93)	
MaxPool2D	(Batch,32, 3, 46)	Size = 1x2
Dropout	(Batch,32, 3, 46)	p = 0.25
Flatten	(Batch, 4416)	
Linear	(Batch, 64)	
BatchNorm1d	(Batch, 64)	
Relu	(Batch, 64)	
Dropout	(Batch, 64)	p = 0.25
Linear	(Batch, 128)	
BatchNorm1d	(Batch, 128)	
Dropout	(Batch, 128)	p = 0.5
Linear	(Batch, 88)	
Sigmoid	(Batch, 88)	

Table 1: LongCNN Architecture.

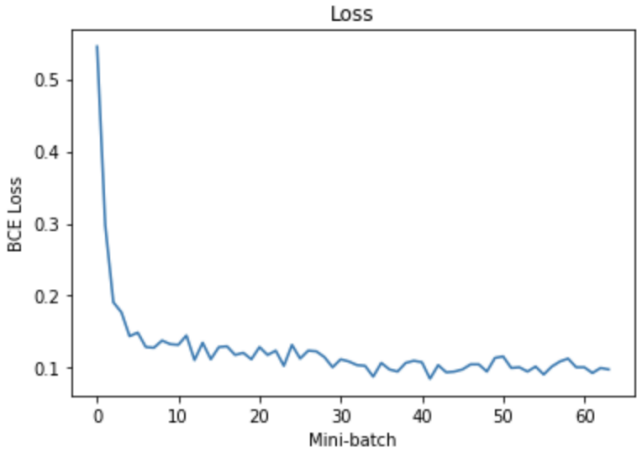

Figure 8: The loss obtained after training each batch of images.

beginning the model obtains almost 0% accuracy, this is because predicting the possible musical notes randomly, there is a $2 * N$ possible combinations, where N is the number of ones in the ground truth, in this specific test set around 45.000. So random guessing

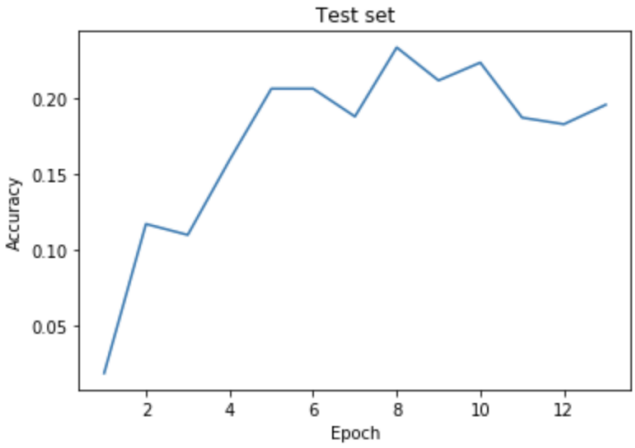

Figure 9: Accuracy obtained after each epoch in the test set. This is the predicted accuracy only among the labels that should be predicted as 1.

obtains a very low accuracy. It is also necessary to remark that this accuracy obtained is way lower than the real accuracy obtained. This values represent the accuracy of the musical notes that should be predicted as 1. So this value does not increase when the network predicts 0 and it was indeed 0.

The network had a high precision but low recall. Among the ones predicted, almost all of them were correctly classified. The problem, and the reason why the accuracy of the model is not so high is because the model predicts more zeros that it is supposed to.

6 CONCLUSIONS

The aim of this thesis was to research to what extent a neural network is able to transcript music. In order to do so, it was necessary to apply Fourier transformations to obtain a representation of the data in the frequency domain. Filter Bank and MFCC approaches were discarded since they were not encoding enough information, leading the model to get lazy and predict all zeros to reduce the loss, causing the model to end up in a local minimum.

By using CQT this problem could be solved. After trying multiple architectures and hyperparameters, the final architecture is described in the Table 1. The network consist of convolutional layers followed by some linear layers to perform the classification.

Results can be considered satisfactory, taking into account that the accuracy that can be obtained by random guessing is very low, the model was able to obtain a 23% accuracy in the musical notes that should be predicted as 1. And most of the ones predicted were correctly classified.

ACKNOWLEDGEMENTS

This work is supported by Equestria (Ponyland), their servers were used to run the experiments of this study using their 2 Nvidia Tesla M40 using their "Twist" server.

REFERENCES

[1] [n.d.]. Magenta. https://research.google/teams/brain/magenta/

[2] Emmanouil Benetos, Sebastian Ewert, and Tillman Weyde. 2014. Automatic transcription of pitched and unpitched sounds from polyphonic music. In *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 3107–3111.

[3] Nancy Bertin, Roland Badeau, and Gaël Richard. 2007. Blind signal decompositions for automatic transcription of polyphonic music: NMF and K-SVD on the benchmark. In *2007 IEEE International Conference on Acoustics, Speech and Signal Processing-ICASSP'07*, Vol. 1. IEEE, I–65.

[4] Nancy Bertin, Roland Badeau, and Emmanuel Vincent. 2009. Fast Bayesian NMF algorithms enforcing harmonicity and temporal continuity in polyphonic music transcription. In *2009 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics*. IEEE, 29–32.

[5] Ronald Newbold Bracewell and Ronald N Bracewell. 1986. *The Fourier transform and its applications*. Vol. 31999. McGraw-Hill New York.

[6] Adriana Guevara-Rukoz, Isin Demirsahin, Fei He, Shan Hui Cathy Chu, Supheakmungkol Sarin, Knot Pipatsrisawat, Alexander Gutkin, Alena Butryna, and Oddur Kjartansson. 2020. Crowdsourcing Latin American Spanish for Low-Resource Text-to-Speech. In *Proc. 12th Language Resources and Evaluation Conference (LREC 2020)*. 11–16 May, Marseille, France, 6504–6513. http://www.lrec-conf.org/proceedings/lrec2020/pdf/2020.lrec-1.801.pdf

[7] Michael Hahn. 2020. What Is MIDI? How To Use the Most Powerful Tool in Music. https://blog.landr.com/what-is-midi/#MIDI-definition

[8] Fei He, Shan Hui Cathy Chu, Oddur Kjartansson, Clara E. Rivera, Anna Katanova, Alexander Gutkin, Isin Demirsahin, Cibu C Johny, Martin Jansche, Supheakmungkol Sarin, and Knot Pipatsrisawat. 2020. Open-source Multi-speaker Speech Corpora for Building Gujarati, Kannada, Malayalam, Marathi, Tamil and Telugu Speech Synthesis Systems. In *Proc. 12th Language Resources and Evaluation Conference (LREC 2020)*. 11–16 May, Marseille, France, 6494-–6503. http://www.lrec-conf.org/proceedings/lrec2020/pdf/2020.lrec-1.800.pdf

[9] Muhammad Huzaifah. 2017. Comparison of time-frequency representations for environmental sound classification using convolutional neural networks. *arXiv preprint arXiv:1706.07156* (2017).

[10] Rainer Kelz and Gerhard Widmer. 2017. An experimental analysis of the entanglement problem in neural-network-based music transcription systems. *arXiv preprint arXiv:1702.00025* (2017).

[11] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[12] PyTorch. 2017. Writing Custom Datasets, DataLoaders and Transforms¶. https://pytorch.org/tutorials/beginner/data_loading_tutorial.html