Joachim Alfonso A. Navarro

Lesli Natasha A. Coronel

May 2018

Real-time Identification of Common and Extended Musical Chords using

Artificial Neural Networks

Philippine Science High School

Main Campus

Submitted to the Faculty of the  
Philippine Science High School – Main Campus  
in partial fulfillment of the requirements for  
Science and Technology Research 2

May 2018

Real-time Identification of Common and

Extended Musical Chords using

Artificial Neural Networks

by

Joachim Alfonso A. Navarro

Lesli Natasha A. Coronel

ABSTRACT

Musical chords are fundamental to musical harmony and are named due to their importance. For most people, accurately identifying or naming musical chords is a difficult task requiring high levels of skill. With this in mind, a neural network that aimed to identify musical chord names from their component notes in real-time was programmed in Python using the Keras framework running on the TensorFlow library. Validation accuracy data was obtained after every training session, and manual MIDI inputs were used to obtain response time data. It was found that the current number of training iterations (2,400) provided an insufficient peak validation accuracy of 5.5%, but showed steadily decreasing values of the loss function. A left-tailed T-test for one mean was carried out on 30 randomly selected chords from the dataset and showed that the neural network responded significantly faster than the generally accepted standard of 10 milliseconds for real-time use. Further development of the neural network is recommended to increase validation accuracy. Such a neural network may be implemented on devices or software for the purposes of music education.

Keywords: Chord, Chord identification, Extended chord, GPU, Neural network, Real-time

Approval Sheet

This research work entitled “**Real-time Identification of Common and Extended Musical Chords using Artificial Neural Networks**” by Joachim Alfonso A. Navarro and Lesli Natasha A. Coronel, presented to the Faculty of the Philippine Science High School – Main Campus in partial fulfillment of the requirements in Science and Technology Research 2 is hereby accepted.

|  |  |
| --- | --- |
|  |  |
|  | Kiel F. Granada  Research Adviser |

Acknowledgements

We would like to thank Ms. Kiel Granada, Mr. Edge Angeles, and Ms. Maria April Rose A. Andaca for their invaluable guidance and support, and Mr. Alejandro Alfonso Navarro and Ms. Maria Amina Amado for their financial assistance.

Table of Contents

|  |  |
| --- | --- |
|  | **Page** |

[Approval Sheet i](#_Toc513144510)

[Acknowledgements ii](#_Toc513144511)

[Table of Contents iii](#_Toc513144512)

[List of Tables vi](#_Toc513144513)

[List of Figures vii](#_Toc513144514)

[Introduction 1](#_Toc513144515)

[Background of the Study 1](#_Toc513144516)

[Objectives of the Study 3](#_Toc513144517)

[Significance of the Study 3](#_Toc513144518)

[Scope and Limitations 4](#_Toc513144519)

[Literature Review 5](#_Toc513144520)

[Chords 5](#_Toc513144521)

[Chord structure and naming 5](#_Toc513144522)

[Importance of chords amongst musicians 5](#_Toc513144523)

[Role of absolute pitch in chord identification 5](#_Toc513144524)

[Previous attempts at chord identification 6](#_Toc513144525)

[Artificial Neural Networks (ANNs) 6](#_Toc513144526)

[Artificial Neural Networks in the fields of music 7](#_Toc513144527)

[Hardware and software 7](#_Toc513144528)

[Graphics processing unit 7](#_Toc513144529)

[Software 8](#_Toc513144530)

[Methodology 9](#_Toc513144531)

[Process Flowchart 9](#_Toc513144532)

[Procurement of materials 9](#_Toc513144533)

[Test chord dataset preparation and randomization 11](#_Toc513144534)

[Neural network development 11](#_Toc513144535)

[Neural network training, testing, and data collection 11](#_Toc513144536)

[Results and Discussion 13](#_Toc513144537)

[Validation and training accuracies 13](#_Toc513144538)

[Validation and training losses 15](#_Toc513144539)

[Summary and Conclusion 19](#_Toc513144540)

[Recommendations 20](#_Toc513144541)

[Latency Testing Dataset 24](#_Toc513144542)

[Neural Network Training Source Code 26](#_Toc513144543)

[Neural Network Testing Source Code 29](#_Toc513144544)

[Accuracy Graph Source Code 33](#_Toc513144545)

[NN Output Interpreter Source Code 34](#_Toc513144546)

[Dataset Initiation Source Code 37](#_Toc513144547)

[Dataset Initiation Source Code: Note Enumeration 39](#_Toc513144548)

[Dataset Initiation Source Code: Chord Type Enumeration 41](#_Toc513144549)

[Dataset Initiation Source Code: Chord Class 44](#_Toc513144550)

[Dataset Initiation Source Code: Scale List 46](#_Toc513144551)

[Dataset Initiation Source Code: Sampler 48](#_Toc513144552)

[Dataset Interpreter Source Code 54](#_Toc513144553)

[Dataset Interpreter Source Code: Output Neuron Map Generator 59](#_Toc513144554)

[Dataset Interpreter Source Code: Output Neuron Mappings 61](#_Toc513144555)

[Key Name VK Dictionary Source Code For *pygame* 63](#_Toc513144556)

List of Tables

|  |  |  |
| --- | --- | --- |
| **Table** | **Title** | **Page** |
| 7 | Results of t-test for one mean on neural network response time | 17 |

List of Figures

|  |  |  |
| --- | --- | --- |
| **Figure** | **Title** | **Page** |
| 1  2  3  4  5  6  8 | Training (blue) and validation (orange) accuracies of neural network, epochs 0-800  Training (blue) and validation (orange) accuracies of neural network, epochs 800-1600  Training (blue) and validation (orange) accuracies of neural network, epochs 1600-2400  Training (blue) and validation (orange) losses of neural network, epochs 0-800  Training (blue) and validation (orange) losses of neural network, epochs 800-1600  Training (blue) and validation (orange) losses of neural network, epochs 1600-2400  Boxplot of total response times of neural network to MIDI inputs in milliseconds. Note: Null hypothesis is that response time is less than or equal to 10 ms. The program used to make this boxplot does not output units on the x-axis. | 13  14  14  15  16  16  17 |

Introduction

Background of the Study

Chords are collections of two or more musical notes, often played together, and are arranged in such a way that they follow the so-called “rules of harmony” (Leino, Brattico, Tervaniemi, & Vurst, 2007). These rules are recognized by humans as a response in the brain that is triggered when they are violated (Leino et al., 2007). Because of this, chords are fundamental to the harmonic integrity of any musical work.

A musical chord is commonly identified by three parameters: root note, chord type, and inversion. The root note serves as a reference point for the other notes which are played in the chord. These other notes are determined by the chord type. The inversion tells which note in the chord is played lowest. For example, a C major chord, 0th inversion has “C” as the root note, and “major” as the chord type. A major chord type includes the 1st (root), major 3rd, and 5th harmonics of the typical Western major scale, and the 0th inversion says that the root note is the bottom note. Thus, the notes of the 0th inversion of a C major chord are C (1st), E (major 3rd), and G (5th), in that order.

Humphrey, Bello, and Cho (n.d.) state that “the general music learning public places a high demand on chord-based representations of popular music” (par. 1). However, complete and accurate determination of these chords by hearing requires the use of both absolute and relative pitch, because chords utilize both an absolute reference point (root note) and a relative configuration of harmonies (chord type).

Absolute pitch is expressed when one can identify a musical note by hearing it, while relative pitch is shown when one can recognize the distances between musical notes (Zatorre, Perry, Beckett, Westbury, & Evans, 1998). While “most trained musicians” (Zatorre et al., 1998) exhibit a mastery of the latter, few of them have absolute pitch.

Absolute pitch is expressed in a low percentage of the human population and acquired through either favorable genes or music training at a young age, or both (Baharloo, Service, Risch, Gitschier, & Freimer, 2000).

Complete chord identification is thus a rare skill found in those with mastery of both absolute and relative pitch, even though chords play an important role in any musical work. An algorithm that automatically identifies chords from individual notes in real time would be a first step towards addressing this problem.

Artificial Neural Networks (ANNs) are computational models that use a layered structure of computational units called neurons in order to learn a certain task. Using mathematical functions, the neural network is able to train itself iteratively through a process called backpropagation until its error rate is significantly low (Nielsen, 2015; Sanderson, 2017). Thus, machine learning modeled through ANNs is designed to be adaptable and simple through self-organization (Daniel, 2013). The machine learns a given input dataset through by mapping an input through a probability distribution into its most probable class (Colina, Perez, & Paraan, 2017).

Objectives of the Study

This study aims to develop a neural network that quickly and correctly identifies common and extended one-root musical chords formed by playing more than two notes on a MIDI input device. Specifically, the program must identify common & extended chords and respond quickly enough to be used in live performance (Stark & Plumbley, 2009). The program must be implemented in programming languages that have MIDI input-output libraries such as pygame for Python and and neural network libraries such as Keras and TensorFlow to facilitate ease of coding.

Significance of the Study

Such application of real-time chord identification would be used in the field of music education, where a low proportion of music students have absolute pitch (Gregersen, Kowalsky, Kohn, & Marvin, 1999) despite their demand for chordal representations of music (Humphrey, Bello, & Cho, n.d., par. 1). These allow said students to learn to identify the chords they are playing more quickly and accurately, and help them develop their senses of relative and absolute pitch. They are also used in situations when musicians need to verify the chords they are playing for correctness, which usually happens when they are learning or composing a musical piece.

While automatic chord identification programs exist, they are either classical algorithm implementations that run on audio inputs in real-time (Fujishima, 1999; Stark & Plumbley, 2009), or neural network implementations that do not aim to run in real-time and do not include chords outside of major and minor triads (Perera & Kodithuwakku, 2005; Osmalskyj, Embrechts, Piérard, & Van Droogenbroeck, 2012; Zhou & Lerch, 2015). Including other chords such as 7ths and extended chords would allow identification of chords from more complex styles of music, such as jazz. The utilization of neural networks to identify extended chords from MIDI signals in real-time is largely unexplored and would provide useful data for future research.

Scope and Limitations

The study aims to create an artificial neural network that identifies both non-extended (“common”) and extended chords within a time limit of 10ms. The dataset comprises of the following chord types only: simple triads (major, minor, aug, dim), dominant extensions (7, 9, 11), major and minor extensions (M7, m7, M9, m9, M11, m11), suspended triads (sus2, sus4), major extensions with suspensions (M7sus2, M7sus4, M9sus2, M9sus4, M11sus2), dominant extensions with suspensions (7sus2, 7sus4, 9sus2, 9sus4, 11sus2), augmented and diminished extensions (aug7, dim7, ø7, aug9, dim9, aug11), and other extensions (mM7, mM9, M6, m6, M6(9), m6(9)).

MIDI commands will be used as inputs to the neural network. Audio datasets are thus not included in the scope of this study. Only 2400 epochs of training were performed due to time constraints.

Literature Review

Chords

Chord structure and naming

Chords are the basis of harmony in music. According to the The Concise Oxford Dictionary of Music, a chord is “any simultaneous combination of notes, but usually of not fewer than 3” (Chord, 2004, p. 147). The name of a chord is dictated by two significant parameters: the root note and the chord type. The root note is a “note from which a chord originates” (Root, 2004, p. 615). It is taken as a reference point from which the chord type is determined. The chord type is also determined by the distances between the notes that comprise it.

Importance of chords amongst musicians

Humphrey, Bello, and Cho (n.d.) state that “the general music learning public places a high demand on chord-based representations of popular music” (par. 1). Websites on the Internet that provide chords for many popular songs exist, but these are generated by a user base of varying levels of skill and are not guaranteed to be correct or accurate.

Role of absolute pitch in chord identification

Identification of musical chords is dependent on absolute pitch, limiting the ability of the general population to identify chords. Absolute pitch is expressed when one can identify a certain musical note by hearing it (Zatorre, Perry, Beckett, Westbury, & Evans, 1998). Absolute pitch is naturally expressed in a low percentage of the human population and can be acquired through either a combination of favorable genes, music training at a young age (Baharloo, Service, Risch, Gitschier, & Freimer, 2000), or a comprehensive ear training course. Complete and accurate determination of these chords by hearing requires the use of absolute pitch, because chords are based on an absolute reference point, the root note (Root, 2004, p. 615).

Previous attempts at chord identification

Fujishima (1999) and Stark & Plumbley (2009) independently made classical algorithms for identifying musical chords from audio inputs in real-time. Both implementations were able to identify chords with a significantly high accuracy. Relatively successful implementations of chord identification using neural networks have also been attempted before, but they do not aim to run in real-time and do not include chords outside major and minor chords (Osmalskyj, Embrechts, Piérard, & Van Droogenbroeck, 2012; Perera & Kodithuwakku, 2005; Zhou & Lerch, 2015). Perera and Kodithuwakku (2015) further note that the use of neural networks to identify chords from MIDI signals is viable; however, none of these researches explore the use of neural networks to identify more complex or extended chords in real-time.

**Artificial Neural Networks (ANNs)**

Neural networks are computational models that use sets of computational units called neurons arranged together in a layered structure. Each neuron in one layer is connected to each neuron in the next layer. Each neuron has a number called its activation, and these activations are passed down the different layers of the neural network. When a neuron receives an activation from the previous layer, it takes on an activation by manipulating the activation passed on to it using a certain mathematical function. In essence, a neural network takes numbers as inputs and gives numbers as outputs (Nielsen, 2015; Sanderson, 2017).

Neural networks are trained to learn which set of outputs are expected for every set of inputs fed to it. This is done by a process called backpropagation. Training is performed for many cycles until the error rate of the neural network becomes significantly low and the accuracy becomes significantly high (Nielsen, 2015; Sanderson, 2017).

### *Artificial Neural Networks in the fields of music*

Osmalskyj, Embrechts, Piérard, and Van Droogenbroeck (2012) proposed a feed-forward neural network for identification of recorded guitar chords and other instruments. A twelve-number vector called the Pitch Class Profile served as the input and represented each note in the chord, which the neural network would use to predict the chord. Another previous study suggested the use of deep neural networks to identify cover songs (Stamenovic, 2015). The method aimed to remedy the inability of other methods to study time and audio spectra by using the neural network to extract certain features of the input audio. Artificial Neural Networks have also been used to study patterns in Carnatic Raga classical music (Srimania & Parimalab, 2011).

**Hardware and software**

### *Graphics processing unit*

Various pieces of hardware and software were utilized in this research. A GPU or graphics processing unit is a parallel processor (Nickolls, Buck, Garland, & Skadron, 2008) useful for parallel computing applications such as artificial neural network (ANN) simulations and training and is empirically found to be as much as 30 times faster than a regular CPU in these tasks (Colina, Perez & Paraan, 2017).

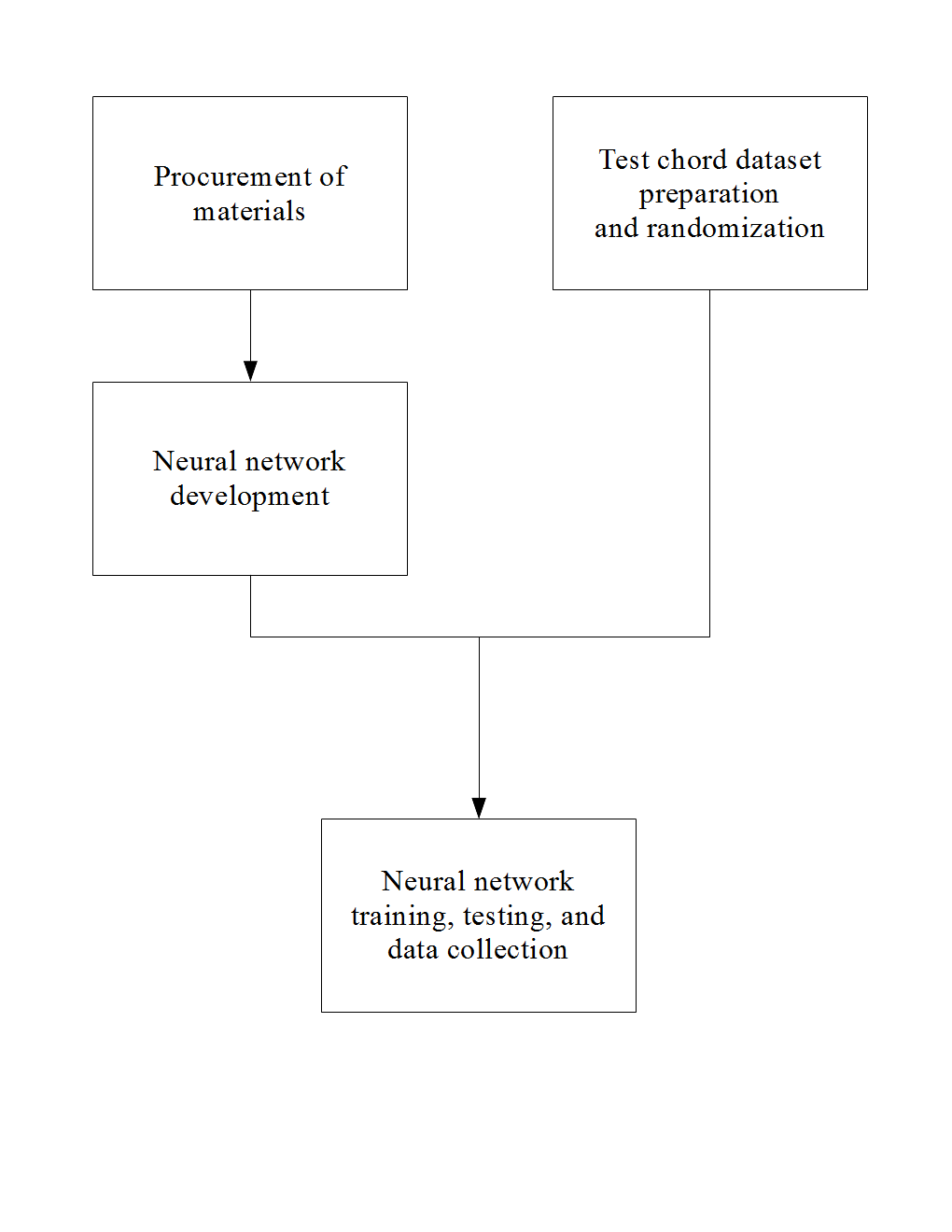
The NVIDIA GeForce GTX 1070 (“GTX 1070” or “1070”) is a GPU is used by many ANN researches (Colina, Perez, & Paraan, 2017; Kang, Hong, & Park, 2017; Kim, Hong, Nam, & Park, 2017, Zorrilla et al., 2017) and was thus selected for use in this research.

### *Software*

Certain pieces of software will be necessary to define the structure of, train, and test the neural network. The neural network libraries necessary include Keras, a library that allows for the prototyping of the neural network on Python; as used by Xie, Zhang, Sapkota, and Yang (2016) in their research; and TensorFlow, as used by Covington, Adams, and Sargin (2016) which allows GPU parallelization (Colina, Perez, & Paraan, 2017) and serves as a backbone to Keras. Use of the GPU depends on NVIDIA CUDA which will be necessary for GPU computations and neural network simulation (Nickolls, Buck, Garland, & Skadron, 2008). Anaconda is a specific distribution of the Python and R programming languages specializing in scientific computing and data processing and will be used in programming the neural network. Finally, pygame (Shinners, 2017) will be installed as it contains facilities for real-time MIDI input. The pygame library is used in the successfully-implemented music platform developed by Cerny, Suthar, and Geselowitz (2013) for user-input MIDI and is therefore viable for use in research work.

Methodology

Process Flowchart



Procurement of materials

In order to facilitate computations that the neural network will perform, a graphics processing unit or GPU was utilized (Nickolls, Buck, Garland, & Skadron, 2008).

It is useful for parallel computing applications such as artificial neural network (ANN) simulations and training. Colina, Perez, and Paraan (2017) found that the use of a consumer GPU accelerated ANN computation by approximately 30 times when compared to a regular central processing unit. The minimum recommended GPU for research work is the NVIDIA GeForce GTX 1070 (“GTX 1070” or “1070”) (Dettmers, 2017). This GPU is used in many neural network researches and papers (Colina, Perez, & Paraan, 2017; Kang, Hong, & Park, 2017; Kim, Hong, Nam, & Park, 2017, Zorrilla et al., 2017) and was ordered online.

The necessary pieces of software to define the structure of, train, and test the neural network were acquired and installed in one of the researchers’ computers using the methods in mustgoplay (2016), which uses the following components.

Keras (Keras Team, 2018) is a Python-specific neural network library that allows for swift prototyping of ANN structures. Its relative simplicity is well-suited to research. It runs on TensorFlow, a neural networks library for Python.

NVIDIA CUDA & cuDNN are libraries that allow computation on an NVIDIA GPU and ANN simulation on a NVIDIA GPU, respectively.

Anaconda is a specific distribution of the Python and R programming languages for scientific computing and data processing.

PyCharm is a piece of software known as an independent development environment or IDE, which allows the user to write and run code.

Finally, pygame, a Python library for game development, will be installed as it contains provisions for real-time MIDI signal input to Python. This input can be fed to the neural network for chord identification.

Test chord dataset preparation and randomization

A program that makes a full list of chords from the 12 possible root notes and the chosen chord types was then written. The program outputs these chords to a text file that will serve as the input dataset of the ANN. Its training and validation datasets were randomized using a Python script to be detailed in Neural network training, testing, and data collection. The input dataset was split by this script into training and validation datasets with 80% and 20% of the chords respectively (Kolassa, 2015).

Neural network development

An algorithm that takes a current MIDI input and converts it into a format that can be interpreted by the neural network was written. The chords are interpreted by the neural network on a per-note basis (that is, one neuron per note spanning two octaves of 12 notes each for a total of 24 notes). Using the appropriate programming tools in Procurement of materials: Acquisition of programming tools, an artificial neural network whose goal is to identify chords was written in Python using Keras (Keras Team, 2018) and TensorFlow.

Neural network training, testing, and data collection

These processes are iterative by nature and thus proceeded once for every training epoch (one training “pass” through the dataset) of the neural network.

The neural network was trained using the built-in functions of Keras (Keras Team, 2018). During training, the neural network was tested with the chords in the training dataset output by the script in Test chord dataset preparation and randomization. The appropriate data such as training and validation accuracy and losses were recorded. Testing proceeded in an identical manner to training using the validation dataset. After the testing phase, adjustments were automatically made by Keras to the neural network depending on the desired output. The network was trained repeatedly until its validation accuracy stopped significantly increasing.

After training, the response time of the neural network was tested against a generally accepted standard of 10 ms for real-time use using a left-tailed T-test for one mean carried out on 30 randomly selected chords from the dataset. These chords were played manually on a MIDI controller, and response time was recorded using another Python script. The statistical T-test was carried out using R (R Core Team, 2017) and the Rcmdr package (Fox & Bouchet-Valat, 2017).

Results and Discussion

**Validation and training accuracies**

Every neural network is trained by traversing through a training dataset, and each pass through the training dataset is called an “epoch”. After every epoch, the accuracy of the network is tested separately using a different dataset called the validation dataset (Nielsen 2015).

After three training sessions of 800 epochs each, the neural network showed a markedly low peak validation accuracy, or rate of correct predictions on the test dataset, of under 5.5%. Every session of 800 epochs showed stepwise and irregular increases and decreases of validation accuracy before consistently dipping to 0% towards the end of the training session. Below are graphs showing the validation (orange) and training (blue) accuracies of the neural network.

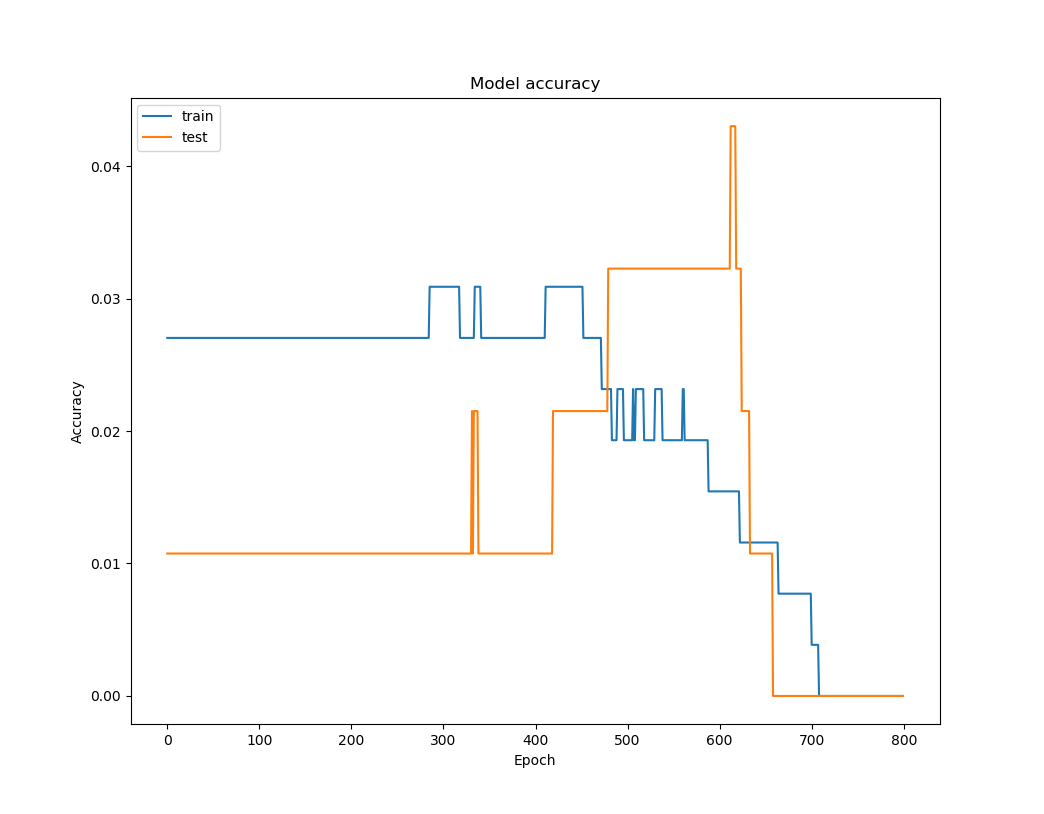


Figure 1: Training (blue) and validation (orange) accuracies of neural network, epochs 0-800.

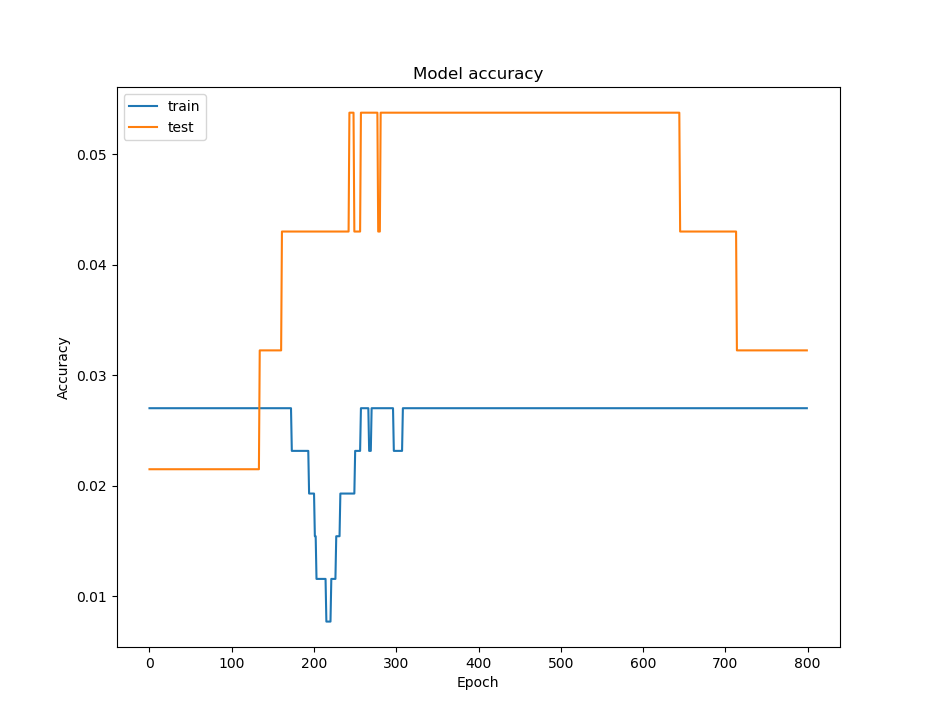


Figure 2: Training (blue) and validation (orange) accuracies of neural network, epochs 800-1600.

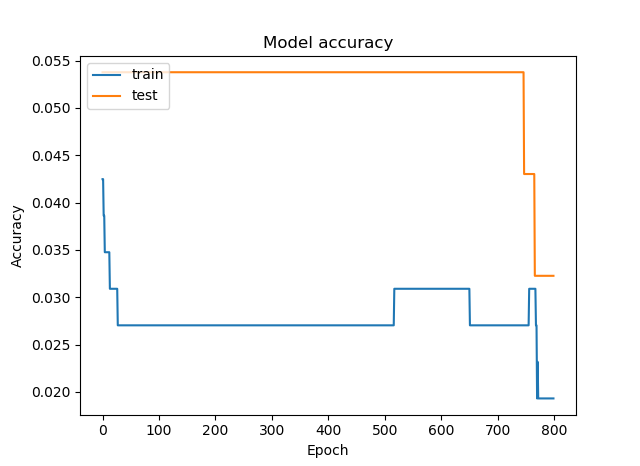


Figure 3: Training (blue) and validation (orange) accuracies of neural network, epochs 1600-2400.

Validation and training losses

The loss function is a special series of mathematical operators that shows how far the neural network is from the correct prediction. As such, lower values are considered favorable (Nielsen 2015).

The same training sessions returned decreasing values of the loss function as the training session proceeded. The training and validation losses consistently decreased in a smooth fashion as expected. Below are graphs showing the validation (orange) and training (blue) losses of the neural network.

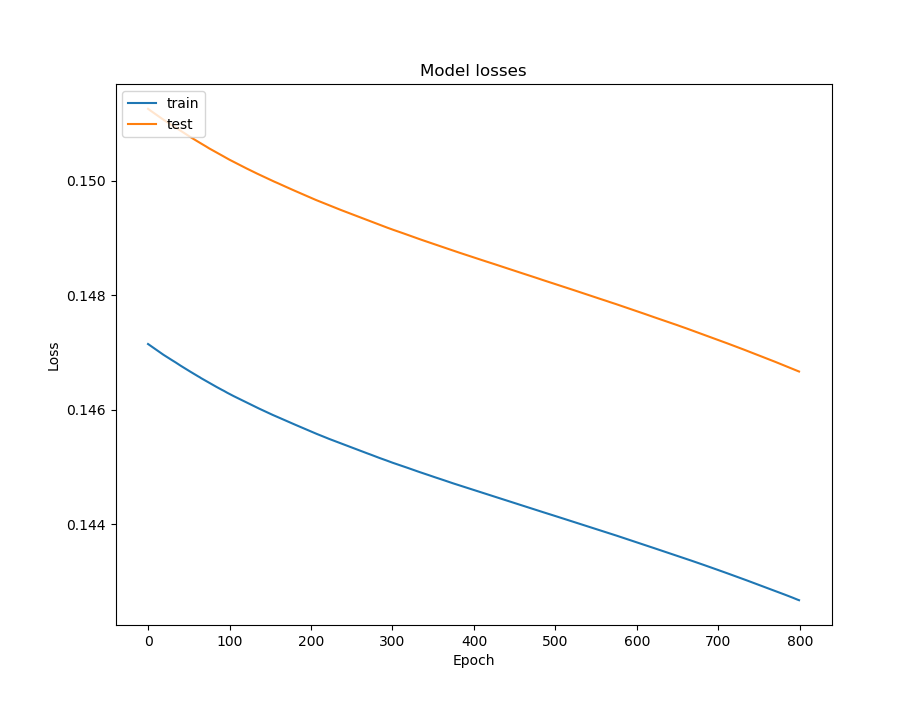


Figure 4: Training (blue) and validation (orange) losses of neural network, epochs 0-800.

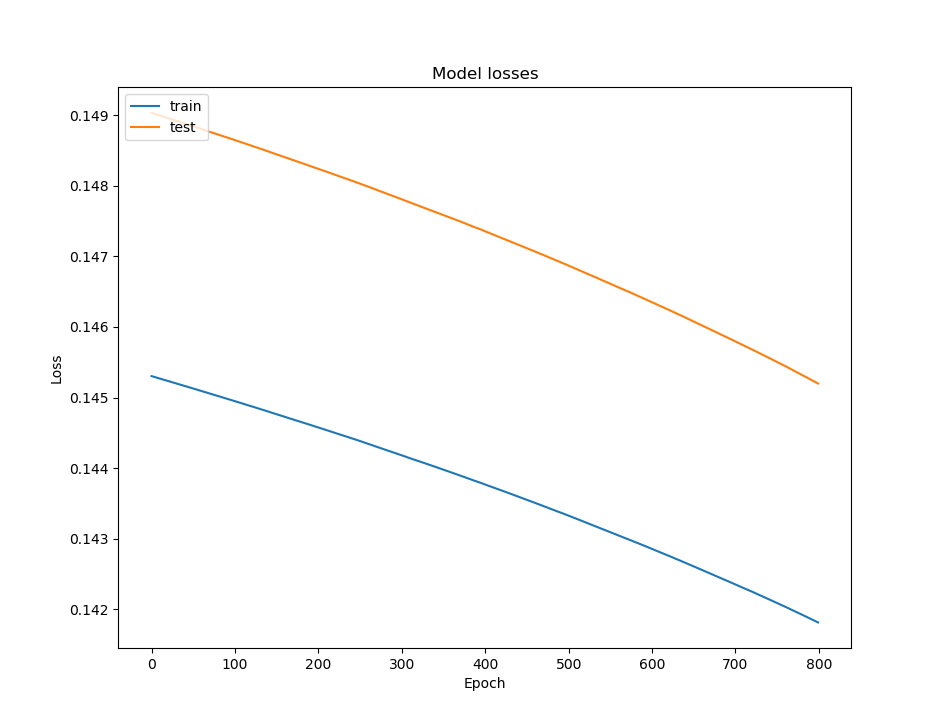


Figure 5: Training (blue) and validation (orange) losses of neural network, epochs 800-1600.

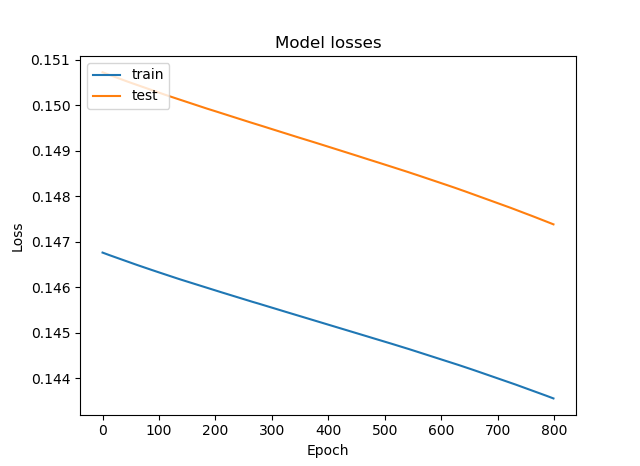


Figure 6: Training (blue) and validation (orange) losses of neural network, epochs 1600-2400.

Table 7: Results of t-test for one mean on neural network response time

|  |  |
| --- | --- |
| Summary measure | Value |
| Mean, ms | 2.95466667 |
| Mean t-statistic | -17.19 |
| Critical t | -1.699 |
| Null hypothesis outcome | Reject |

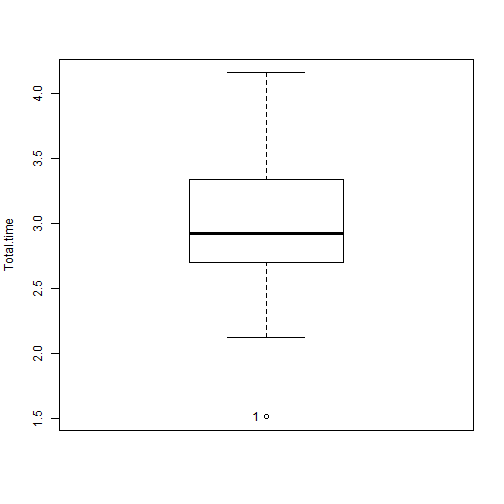


Figure 8: Boxplot of total response times of neural network to MIDI inputs in milliseconds. Note: Null hypothesis is that response time is less than or equal to 10 ms. The program used to make this boxplot does not output units on the x-axis.

The neural network performs atypically in terms of validation and training accuracies. It is expected that the training and validation accuracies would increase as the training proceeds. However, it is behaving as expected in terms of validation and training losses, as they decrease during training (Nielsen, 2015).

Bodik (2018) suggests that this behavior is caused by the input and output formats of the neural network being too complex for the neural network to properly classify input data. This can cause the neural network to underfit, i.e. not be able to adapt to the characteristics of the training data. This is manifested by consistently unfavorable values of validation accuracy. (Nielsen, 2015; Bodik, 2018). It is therefore recommended that steps to reduce this underfitting be taken in the future.

Summary and Conclusion

A neural network that aimed to identify common and extended musical chords in real-time was programmed and did so with a markedly low accuracy. However, its satisfactory response time shows that neural networks may be used in real-time tasks.

Recommendations

The use of a simpler neural network structure and an increase in training time are recommended. Other machine learning algorithms like linear regression may be considered, as the breadth of the dataset is limited (i.e. there are limited combinations of notes for one certain chord).

BIBLIOGRAPHY

Baharloo, S., Service, S., Risch, N., Gitschier, J. & Freimer, N. (2000). Familial aggregation of absolute pitch. *American Journal of Human Genetics, 67,* 755-758. doi:10.1086/303057.

Bhabra, K. (2013). *pyMidi* [GitHub repository]. Retrieved November 25, 2017, from https://github.com/kushalbhabra/pyMidi.

Cerny, D., Suthar, J., & Geselowitz, I. (2013). *U-AMP: User input based algorithmic music platform.* Retrieved from https://pdfs.semanticscholar.org/3901/eda9bafbc0e831f2e743380469284f2ac78e.pdf.

Chord. (2004). In *The concise Oxford dictionary of music* (4th ed.) Oxford, UK: Oxford University Press.

Colina, N. C. A., Perez, C. E., & Paraan, F. N. C. (2017). Simple techniques for improving deep neural network outcomes on commodity hardware. *AIP Conference Proceedings*, *1871*, 040001. doi:10.1063/1.4996523.

Covington, P., Adams, J., & Sargin, E. (2016). Deep neural networks for YouTube recommendations. *Proceedings of the 10th ACM Conference on Recommender Systems*, 191-198. doi:10.1145/2959100.2959190.

Daniel, G. (2013). *Principles of artificial neural networks* (3rd ed.) Chicago, IL: World Scientific.

Dettmers, T. (2017). *Which GPU(s) to get for deep learning: My experience and advice for using GPUs in deep learning*. Retrieved from http://timdettmers.com/2017/04/09/which-gpu-for-deep-learning/.

Fox, J., & Bouchet-Valat, M. (2017). Rcmdr: R Commander. R package version 2.3-2.

Fujishima, T. (1999). *Realtime chord recognition of musical sound: A system using common Lisp music.* Retrieved from http://www.music.mcgill.ca/~jason/mumt621/papers5/fujishima\_1999.pdf.

Gregersen, P., Kowalsky, E., Kohn, N., & Marvin, E. (1999). Absolute pitch: prevalence, ethnic variation, and estimation of the genetic component [Letter to the Editor]. *American Journal of Human Genetics, 65,* 911-913. doi:10.1086/302541.

Humphrey, E., Bello, J. P., & Cho, T. (n.d.). Chord Recognition. Retrieved from http://steinhardt.nyu.edu/marl/research/chord\_recognition.

Kang, J. K., Hong, H. G., Park, K. R. (2017). Pedestrian detection based on adaptive selection of visible light or far-infrared light camera image by fuzzy inference system and convolutional neural network-based verification. *Sensors*, *17*(7), 1598. doi:10.3390/s17071598.

Keras Team. (2018). *Keras* [GitHub repository]. Retrieved January 25, 2018, from https://github.com/keras-team/keras.

Kim, K. W., Hong, H. G., Nam, G. P., & Park K. R. (2017). A study of deep CNN-based classification of open and closed eyes using a visible light camera sensor. *Sensors*, *17*(7), 1534. doi:10.3390/s17071534.

Kolassa, S. (2015). *Dividing percentage* [Online forum comment.] Retrieved from https://datascience.stackexchange.com/questions/5113/dividing-percentage.

Leino, S., Brattico, E., Tervaniemi, M., & Vurst, P. (2007). Representation of harmony rules in the human brain: Further evidence from event-related potentials. *Brain Research, 1142,* 169-177. doi:10.1016/j.brainres.2007.01.049.

*mustgoplay* (2016). *Anaconda Python, CUDA, cuDNN and Theano with PyCharm install for CPU and GPU on Windows 10* [Online forum comment.] Retrieved from https://github.com/Theano/Theano/issues/5348.

Nickolls, J., Buck, I., Garland, M., & Skadron, K. (2008). Scalable parallel programming with CUDA. *ACM Queue, 6*(2), 40-53.

Nielsen, M. A. (2015). *Neural networks and deep learning*. Determination Press.

Osmalskyj, J., Embrechts, J-J., Piérard, S., & Van Droogenbroeck, M. (2012, May 9). *Neural networks for musical chords recognition*. Retrieved at http://jim.afim-asso.org/jim12/pdf/jim2012\_08\_p\_osmalskyj.pdf.

Perera, N., & Kodithuwakku, S. R. (2005, December 15). Music chord recognition using artificial neural networks. *1st Proceedings of the International Conference on Information and Automation*, 304-308.

R Core Team (2017). *R: A language and environment for statistical computing*. R Foundation for Statistical Computing, Vienna, Austria. Retrieved from https://www.R-project.org/.

Root. (2004). In *The concise Oxford dictionary of music* (4th ed.) Oxford, UK: Oxford University Press.

Sanderson, G. [3Blue1Brown]. (2017, October 5). *But what \*is\* a Neural Network? | Chapter 1, deep learning*. Retrieved from https://www.youtube.com/watch?v=aircAruvnKk.

Shinners, P. (2017).  *pygame* - Python game development. Retrieved January 25, 2018, from http://www.pygame.org.

Srimania, P.K., & Parimalab, Y.G. (2011). Artificial neural network approach to develop unique classification and Raga identification tools for pattern recognition in Carnatic music. *AIP Conference Proceedings 2011, 1414(1),* 227-231. doi:https://doi.org/10.1063/1.3669961

Stamenovic, M. (2015). *Identifying cover songs using deep neural networks* [PDF file]. Retrieved at https://pdfs.semanticscholar.org/9b51/62790afbe733e777a9352732ccb5cc30559d.pdf?\_ga=2.51066401.901537269.1523544799-1147536875.1523544799.

Stark, A. M., & Plumbley, M. D. (2009). *Real-time chord recognition for live performance* [PDF file]*.* Retrieved at https://www.eecs.qmul.ac.uk/~markp/2009/StarkPlumbley09-icmc.pdf.

Xie, Y., Zhang, Z., Sapkota, M., & Yang, L. (2016). Spatial clockwork recurrent neural network for muscle perimysium segmentation. In S. Ourselin, L. Joskowicz, M. Sabuncu, G. Unal, W. Wells (Eds.), *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2016*. *MICCAI 2016: Lecture notes in computer science, vol 9901.* Cham: Springer.

Zatorre, R. J., Perry, D. W., Beckett, C. A., Westbury, C. F., & Evans, A. C. (1998). Functional anatomy of musical processing in listeners with absolute pitch and relative pitch. *Proceedings of the National Academy of Sciences, 95*, 3172-3177. Retrieved at http://www.pnas.org/content/95/6/3172.full.

Zhou, X., & Lerch, A. (2015). Chord detection using deep learning. *16th International Society for Music Information Retrieval Conference*, 52-58. Retrieved at http://ismir2015.uma.es/articles/96\_Paper.pdf.

Zorrilla, A. L., de Velasco-Vázquez, M., Serradilla-Casado, O., Roa-Barco, L., Graña, M., Chyzhyk, D., & Price, C. C. (2017). Brain white matter lesion segmentation with 2D/3D CNN. *International Work-Conference on the Interplay Between Natural and Artificial Computation 2017*, *10334*, 394-403. doi:10.1007/978-3-319-59740-9\_39.

**APPENDIX A**

# LATENCY TESTING DATASET

|  |  |
| --- | --- |
| *Chord played* | *Total time (ms)* |
| A | **1.51** |
| A#m9 | **2.12** |
| Daug | **2.19** |
| Em | **3.05** |
| Dm7 | **2.37** |
| C#M7 | **2.4** |
| G#sus4 | **2.96** |
| G#7 | **2.25** |
| Csus4 | **2.98** |
| Cdim | **3.18** |
| C9sus4 | **2.63** |
| BM9 | **2.7** |
| F#9sus4 | **2.71** |
| B11sus2 | **2.74** |
| FmM7 | **2.77** |
| Am6 | **3.06** |
| D#M6 | **2.8** |
| A#m6 | **2.83** |
| C#dim9 | **2.86** |
| C#11sus2 | **2.88** |
| EM6 | **3.34** |
| F9 | **3.13** |
| F#sus2 | **3.91** |
| GM7sus2 | **3.24** |
| A9sus2 | **3.34** |
| BM7sus2 | **3.72** |
| Bm | **4.16** |
| B7sus4 | **3.72** |
| Baug9 | **3.48** |
| FM11 | **3.61** |

**APPENDIX B**

# NEURAL NETWORK TRAINING SOURCE CODE

'''

Neural Network Trainer

Initializes and trains the neural network

author: R2\_B09

'''

from keras.models import Sequential, load\_model

from keras.layers import Dense

import keras.callbacks

import numpy as NP

import h5py

import soleil as sol

FIRST\_RUN = False

INPUT\_VALS = NP.load("input\_dataset/samples/database/NP\_INPUT\_NEURON\_VALUES.npy")

OUTPUT\_VALS = NP.load("input\_dataset/samples/database/NP\_OUTPUT\_NEURON\_VALUES.npy")

VALIDATION\_INPUT\_VALS = NP.load("input\_dataset/samples/database/NP\_INPUT\_NEURON\_VALUES\_NOT\_SAMPLE.npy")

VALIDATION\_OUTPUT\_VALS = NP.load("input\_dataset/samples/database/NP\_OUTPUT\_NEURON\_VALUES\_NOT\_SAMPLE.npy")

def splitter():

   VI = VALIDATION\_INPUT\_VALS.tolist()

   VO = VALIDATION\_OUTPUT\_VALS.tolist()

   print(VI, len(VI))

   print(VO, len(VO))

   VI\_TEST = []

   VO\_TEST = []

   VI\_VALI = []

   VO\_VALI = []

   for i in range(len(VI)):

       if i % 2 == 0:

           VI\_VALI.append(VI[i])

           VO\_VALI.append(VO[i])

       else:

           VI\_TEST.append(VI[i])

           VO\_TEST.append(VO[i])

   NP.save("input\_dataset/samples/database/NP\_INPUT\_VALIDATION.npy", NP.array(VI\_VALI))

   NP.save("input\_dataset/samples/database/NP\_INPUT\_TEST.npy", NP.array(VI\_TEST))

   NP.save("input\_dataset/samples/database/NP\_OUTPUT\_VALIDATION.npy", NP.array(VO\_VALI))

   NP.save("input\_dataset/samples/database/NP\_OUTPUT\_TEST.npy", NP.array(VO\_TEST))

def save\_history(mh):

   MODE = 'w' if FIRST\_RUN else 'a'

   with open("models/history/mse.txt", MODE) as ModelHistoryMSE:

        ModelHistoryMSE.write(str(mh.history['mean\_squared\_error']))

   with open("models/history/val\_acc.txt", MODE) as ModelHistoryVA:

       ModelHistoryVA.write(str(mh.history['val\_acc']))

   with open("models/history/val\_loss.txt", MODE) as ModelHistoryVL:

       ModelHistoryVL.write(str(mh.history['val\_loss']))

def checkpoint(verb = 1):

   return keras.callbacks.ModelCheckpoint("models/chord\_identifier.h5", verbose=verb, monitor='val\_acc', save\_best\_only=True, mode='max')

if \_\_name\_\_ == "\_\_main\_\_":

   VALIDATION\_INPUT = NP.load("input\_dataset/samples/database/NP\_INPUT\_VALIDATION.npy")

   VALIDATION\_OUTPUT = NP.load("input\_dataset/samples/database/NP\_OUTPUT\_VALIDATION.npy")

   print('INPUT\_VALS shape: ' + str(NP.shape(INPUT\_VALS)))

   print('INPUT\_VALS shape of each element: ' + str(NP.shape(INPUT\_VALS[0])))

   print('OUTPUT\_VALS shape: ' + str(NP.shape(OUTPUT\_VALS)))

   print('OUTPUT\_VALS shape of each element: ' + str(NP.shape(OUTPUT\_VALS[0])))

   if FIRST\_RUN: # Make a new NN if FIRST\_RUN is true

       chord\_identifier = Sequential()

       chord\_identifier.add(Dense(14, input\_shape = (24,), activation = 'sigmoid'))

       chord\_identifier.add(Dense(2))

       chord\_identifier.add(Dense(2))

       chord\_identifier.add(Dense(14, activation = 'softmax'))

       chord\_identifier.compile(optimizer = 'sgd', loss = 'mean\_squared\_error', metrics = ['mse', 'accuracy'])

       checkpointer = checkpoint()

       callbacks\_list = [checkpointer]

       h = chord\_identifier.fit(INPUT\_VALS, OUTPUT\_VALS, epochs = 800, verbose = 1, validation\_data = (VALIDATION\_INPUT, VALIDATION\_OUTPUT), callbacks=callbacks\_list)

       save\_history(h)

       sol.graph\_from\_History(things\_to\_graph=['acc', 'val\_acc'], MHObject=h, title="Model accuracy", ylabel="Accuracy", xlabel="Epoch", legendlist=['train', 'test'], legendloc = 'upper left')

       sol.graph\_from\_History(things\_to\_graph=['loss', 'val\_loss'], MHObject=h, title="Model losses", ylabel="Loss", xlabel="Epoch", legendlist=['train', 'test'], legendloc = 'upper left')

   else:

       # recall and train NN if FIRST\_RUN is false

       chord\_identifier = keras.models.load\_model("models/chord\_identifier.h5")

       chord\_identifier.compile(optimizer='sgd', loss='mean\_squared\_error', metrics=['mse', 'accuracy'])

       checkpointer = checkpoint()

       h = chord\_identifier.fit(INPUT\_VALS, OUTPUT\_VALS, epochs=1000, verbose=1, validation\_data=(VALIDATION\_INPUT, VALIDATION\_OUTPUT), callbacks=callbacks\_list)

       save\_history(h)

       sol.graph\_from\_History(things\_to\_graph=['acc', 'val\_acc'], MHObject=h, title="Model accuracy", ylabel="Accuracy", xlabel="Epoch", legendlist=['train', 'test'], legendloc = 'upper left')

**APPENDIX C**

# NEURAL NETWORK TESTING SOURCE CODE

'''

Neural Network Tester

Tests the neural network and gives a graph of the results

author: R2\_B09

'''

import timeit

import keras

import numpy as NP

import h5py

import loc\_note\_MIDI

import pygame

from pygame import midi

from pygame.locals import \*

import moira as moi

def check\_gpu():

   from keras import backend as K

   K.tensorflow\_backend.\_get\_available\_gpus()

def NNtest(INPUT, EXPECTED\_OUTPUT, IDENTIFIER):

   '''

   Runs a single NN test pass given input and expected outs as numpy arrays.

   :param EXPECTED\_INPUT: Numpy array for expected input.

   :param EXPECTED\_OUTPUT: Numpy array for expected output.

   :return:

   '''

   # SAFETY NET: If nothing is running through realtime input then pass

   if INPUT is None or len(INPUT) < 24:

       pass

   else:

       \_IN\_ = NP.array(INPUT)

       #print(str(NP.shape(\_IN\_)))

       tick = timeit.default\_timer()

       guess\_np = IDENTIFIER.predict(\_IN\_.reshape(1, 24))

       tock = timeit.default\_timer()

       guess = guess\_np.flatten().tolist()

       #print(guess)

       print("===============================")

       outertick = timeit.default\_timer()

       print("Guessed chord: " + moi.determineChordName(guess, INPUT))

       outertock = timeit.default\_timer()

       print("Time elapsed: " + str(tock - tick))

       print("Time elapsed, including display chord: " + str(tock - tick + outertock - outertick))

       print("===============================")

def realTimeTest(ID):

   '''

   Displays the mappings of the notes currently being played on MIDI and runs them through the NN,

   (kushalbhabra, 2013; Tang, n.d.)

   '''

   # display a list of MIDI devices connected to the computer

   # this function written by Stephen Tang (n.d.), stephentang.io

   def print\_device\_info():

       for i in range(pygame.midi.get\_count()):

           r = pygame.midi.get\_device\_info(i)

           (interf, name, input, output, opened) = r

           in\_out = ""

           if input:

               in\_out = "(input)"

           if output:

               in\_out = "(output)"

           print("%2i: interface: %s, name: %s, opened: %s %s" %

                 (i, interf, name, opened, in\_out))

   pygame.init()

   pygame.fastevent.init()

   event\_get = pygame.fastevent.get

   event\_post = pygame.fastevent.post

   pygame.midi.init()

   print("Available MIDI devices:")

   print\_device\_info()

   MI = int(input("Select device ID: "))  # 2 the default for Project-X99

   '''

   Examples for Project-X99:

   INPUT 2: Komplete Kontrol 1

   INPUT 3: Komplete Kontrol EXT

   INPUT 4: Komplete Kontrol DAW

   '''

   input\_id = MI

   midi\_in = pygame.midi.Input(input\_id)

   pygame.display.set\_caption("midi test")

   screen = pygame.display.set\_mode((400, 300), RESIZABLE, 32)

   print("Initializing realtime poll process.")

   realtime = True

   currentNoteNumbersInChord = []  # Current MIDI note #s in chord

   currentNotesInChord = []        # Binary mapping of notes in chord

   currentNoteNamesInChord = []    # Current note names in chord

   for i in range(24):

       currentNotesInChord.append(0)

   print("initialize:\n", currentNotesInChord, "length:" + str(len(currentNotesInChord)))

   while realtime:

       # pygame event handling

       events = event\_get()

       for e in events:

           if e.type in [QUIT]:

               realtime = False

           if e.type in [KEYDOWN]:

               realtime = False

       if midi\_in.poll():

           midi\_events = midi\_in.read(10)

           # MIDI NOTE ON

           if 0x90 <= midi\_events[0][0][0] <= 0x9F:

               mod = 1

               if len(currentNotesInChord) != 24:

                   print("Error - length of array is not 24! Emergency stop.")

                   realtime = False

                   break

               # Add note number to current note numbers in chord

               currentNoteNumbersInChord.append(midi\_events[0][0][1])

               # Add note to binary list

               root = currentNoteNumbersInChord[0] % 12

               # print("root:", str(root))

               # print("last element of currentNoteNumbersInChord:", currentNoteNumbersInChord[-1])

               # print("current note added:", midi\_events[0][0][1])

               try:

                   currentNotesInChord[(root) + (currentNoteNumbersInChord[-1] - currentNoteNumbersInChord[0])] = 1

               except IndexError:

                   currentNotesInChord[midi\_events[0][0][1] % 12] = 1

               # Add note being played

               currentNoteNamesInChord.append(loc\_note\_MIDI.Loc\_Note\_MIDI.midiNumToNote(midi\_events[0][0][1]))

               #print(currentNotesInChord)

               #print(currentNoteNumbersInChord)

               print(currentNoteNamesInChord)

           # MIDI NOTE OFF

           elif 0x80 <= midi\_events[0][0][0] <= 0x8F:

               print("Purging variables.")

               currentNotesInChord = []

               for i in range(24): currentNotesInChord.append(0)

               currentNoteNumbersInChord = []

               currentNoteNamesInChord = []

           # Aftertouch

           elif 0xD0 <= midi\_events[0][0][0] <= 0xDF:

               pass

           # Something else except aftertouch

           else:

               print("Something else was tinkered.")

               print(midi\_events[0][0][0])

           # Execute NN testing only if there are more than 2 notes playing

           if currentNotesInChord.count(1) > 2:

               NNtest(currentNotesInChord, [], ID)

   print("Exiting.")

   midi\_in.close()

   pygame.midi.quit()

   pygame.quit()

   exit()

if \_\_name\_\_ == "\_\_main\_\_":

   chord\_identifier = keras.models.load\_model("models/chord\_identifier.h5")

   chord\_identifier.compile(optimizer='sgd', loss='mean\_squared\_error', metrics=['mse', 'accuracy'])

   realTimeTest(chord\_identifier)

**APPENDIX D**

# ACCURACY GRAPH SOURCE CODE

'''

Summative Output Landscaper

Outputs a graph of the parameter and training / validation data passed to it.

author: R2\_B09

'''

import matplotlib.pyplot as plt

def graph\_from\_History(things\_to\_graph, MHObject, title=None, ylabel=None, xlabel=None, legendlist=None, legendloc='upper left'):

   for i in range(len(things\_to\_graph)):

       plt.plot(MHObject.history[things\_to\_graph[i]])

   plt.title(title)

   plt.ylabel(ylabel)

   plt.xlabel(xlabel)

   plt.legend(legendlist, loc=legendloc)

   plt.show()

**APPENDIX E**

# NN OUTPUT INTERPRETER SOURCE CODE

'''

Mediating Output Interpreter for Real-time Analysis

Converts the numerical outputs of the neural network into a chord name.

author: R2\_B09

'''

import math

import numpy as NP

UPPER\_BOUNDS = [0.08, 0.16, 0.24, 0.32, 0.40, 0.48, 0.56, 0.64, 0.72, 0.80, 0.88, 0.96]

NOTES        = ['C', 'C#', 'D',  'D#', 'E', 'F', 'F#', 'G',  'G#', 'A', 'A#', 'B']

def softmax(x, verbose=False):

   x\_exp = [math.exp(i) for i in x]

   #print([round(i, 2) for i in x\_exp])

   sum\_x\_exp = sum(x\_exp)

   #print(round(sum\_x\_exp, 2))

   soft\_max = [round(i / sum\_x\_exp, 3) for i in x\_exp]

   if verbose:

       print(soft\_max)

   return soft\_max

def determineNote(NEURON\_0\_VALUE):

   #print(NEURON\_0\_VALUE)

   if NEURON\_0\_VALUE < 0.08:

       return 'C'

   else:

       for i in range(1, 12):

           if UPPER\_BOUNDS[i - 1] < NEURON\_0\_VALUE <= UPPER\_BOUNDS[i]:

               return NOTES[i]

           elif NEURON\_0\_VALUE >= 0.96:

               return "B"

def determineChordName(outputs, inputs, verbose=False) -> str:

   # Access the expected outputs for the inputs

   possible\_expected\_outputs = NP.load("input\_dataset/samples/database/NP\_ALL\_EXPECTED\_OUTPUTS.npy")

   possible\_expected\_inputs = NP.load("input\_dataset/samples/database/NP\_ALL\_EXPECTED\_INPUTS.npy")

   comparator\_of\_expected\_inputs\_index = -1

   ips = NP.array(inputs)

   if verbose:

       print("possible\_expected\_inputs: " + str(possible\_expected\_inputs))

   # Check which index of inputs corresponds to the given inputs

   for i in range(len(possible\_expected\_inputs)):

       if possible\_expected\_inputs[i][0] == ips[0]:

           if NP.array\_equal(possible\_expected\_inputs[i], ips):

               comparator\_of\_expected\_inputs\_index = i

               break

   # Access the corresponding outputs

   expected\_outputs = possible\_expected\_outputs[comparator\_of\_expected\_inputs\_index]

   if verbose:

       print(expected\_outputs)

   # Main determine chord name routine

   rn = float(outputs[0])

   softmax\_in = []

   for i in range(1, len(expected\_outputs)):

       softmax\_in.append(expected\_outputs[i])

   ROOT\_NOTE = determineNote(rn)

   if verbose:

       print(ROOT\_NOTE)

   chord\_modifiers = ""

   soft\_max\_expected = [0]

   soft\_max\_expected.extend(softmax(softmax\_in, verbose))

   for output\_neuron in range(1, len(outputs)):

       if outputs[output\_neuron] >= soft\_max\_expected[output\_neuron]:             # The number contained is 1

           if output\_neuron == 1:                  # The index is some value

               chord\_modifiers += "maj"

           elif output\_neuron == 2:

               chord\_modifiers += "min"

           elif output\_neuron == 7 and outputs[2] == 1 and outputs[9] == 1:

               chord\_modifiers += "o"

           elif output\_neuron == 3:

               chord\_modifiers += "aug"

           elif output\_neuron == 4:

               chord\_modifiers += "dim"

           else:

               pass

   for output\_neuron in range(1, len(outputs)):

       if outputs[output\_neuron] >= soft\_max\_expected[output\_neuron]:

           if output\_neuron == 8:

               chord\_modifiers += "6"

           elif output\_neuron == 9:

               chord\_modifiers += "7"

           elif output\_neuron == 10:

               chord\_modifiers += "9"

           elif output\_neuron == 11:

               chord\_modifiers += "11"

           elif output\_neuron == 12:

               chord\_modifiers += "maj7"

           else:

               pass

   for output\_neuron in range(1, len(outputs)):

       if outputs[output\_neuron] >= soft\_max\_expected[output\_neuron]:

           if output\_neuron == 13:

               chord\_modifiers += "(9)"

           elif output\_neuron == 5:

               chord\_modifiers += "sus2"

           elif output\_neuron == 6:

               chord\_modifiers += "sus4"

           elif output\_neuron == 7 and outputs[2] == 1 and outputs[9] == 1:

               chord\_modifiers = "o7"

           else:

               pass

   return ROOT\_NOTE + chord\_modifiers

**APPENDIX F**

# DATASET INITIATION SOURCE CODE

'''

Dataset Initiation Algorithm for Nominal Note Extrapolation

MAIN FILE:

Initializes a dataset from a given set of possible notes and chord types.

author: R2\_B09

'''

import chord

import ChordTypes

import Note

import scale

chordsFile = None

chordObjectsList = []

chordsList = []

chordAbsolutePositionsList = []

subsetChordObjectsList = []

subsetChordsList = []

def generateChords():

   '''

   Constructs all possible chord objects as defined in ChordTypes.py and Note.py

   :return: Void

   '''

   for rootNote in scale.TWELVE\_NOTE\_SCALE:

       for chordType in scale.CHORD\_TYPES:

           chordObjectsList.append(chord.Chord(rootNote, chordType))

   for a in range(len(chordObjectsList)):

       chordname = chordObjectsList[a].\_\_repr\_\_() + " "

       chordpat = chordObjectsList[a].getPattern()

       chordap = chordObjectsList[a].getAbsoluteNotePattern()

       chordsList.append(chordname + chordpat + "  " + chordap)

       #chordsList.append(chordname)

def writeToChordsFile():

   '''

   Writes list of chords to an output text file.

   :return: Void

   '''

   for i in chordsList:

       print(i)

   with open("../input\_dataset.txt", 'w') as chordsFile:

       for i in chordsList:

           chordsFile.write(i)

           chordsFile.write("\n")

       chordsFile.write("\n\n\n" + "Number of chords: " + str(len(chordsList)))

   print("Written " + str(len(chordsList)) + " chords to file.")

# Main thread

if \_\_name\_\_ == "\_\_main\_\_":

   generateChords()

   writeToChordsFile()

**APPENDIX G**

# DATASET INITIATION SOURCE CODE: NOTE ENUMERATION

'''

Dataset Initiation Algorithm for Nominal Note Extrapolation

NOTE ENUMERATION:

Contains representations of the 12 possible notes in the Western musical scale.

author: R2\_B09

'''

from enum import Enum

class Note(Enum):

   C    = 1

   C\_SHARP = 2

   D    = 3

   D\_SHARP = 4

   E    = 5

   F    = 6

   F\_SHARP = 7

   G    = 8

   G\_SHARP = 9

   A    = 10

   A\_SHARP = 11

   B    = 12

   def noteToString(n):

       if n == Note.C:

           return "C"

       elif n == Note.C\_SHARP:

           return "C#"

       elif n == Note.D:

           return "D"

       elif n == Note.D\_SHARP:

           return "D#"

       elif n == Note.E:

           return "E"

       elif n == Note.F:

           return "F"

       elif n == Note.F\_SHARP:

           return "F#"

       elif n == Note.G:

           return "G"

       elif n == Note.G\_SHARP:

           return "G#"

       elif n == Note.A:

           return "A"

       elif n == Note.A\_SHARP:

           return "A#"

       elif n == Note.B:

           return "B"

       else:

           return "X"

   def stringToNote(n):

       if n == 'C':

           return Note.C

       elif n == 'C#':

           return Note.C\_SHARP

       elif n == 'D':

           return Note.D

       elif n == 'D#':

           return Note.D\_SHARP

       elif n == 'E':

           return Note.E

       elif n == 'F':

           return Note.F

       elif n == 'F#':

           return Note.F\_SHARP

       elif n == 'G':

           return Note.G

       elif n == 'G#':

           return Note.G\_SHARP

       elif n == 'A':

           return Note.A

       elif n == 'A#':

           return Note.A\_SHARP

       elif n == 'B':

           return Note.B

       else:

           return None

**APPENDIX H**

# DATASET INITIATION SOURCE CODE: CHORD TYPE ENUMERATION

'''

Dataset Initiation Algorithm for Nominal Note Extrapolation

CHORD TYPE ENUMERATION:

Contains enumeration of chord types with relative chromatic mappings. (Root = 1). Also contains a representation method.

author: R2\_B09

'''

from enum import Enum

NUMBER\_OF\_CHORD\_TYPES = 37

NUMBER\_OF\_CHORDS = 12 \* NUMBER\_OF\_CHORD\_TYPES

'''

NOTE    C C#   D D# E    F F# G G#    A A# B

#NUM    1 2   3 4 5    6 7 8 9     10 11 12

       13 14   15 16 17   18 19 20 21    22 23 24

'''

class ChordTypes(Enum):

   # Triads

   MAJOR           = (1, 5, 8)

   MINOR           = (1, 4, 8)

   AUGMENTED       = (1, 5, 9)

   DIMINISHED      = (1, 4, 7)

   SUS2            = (1, 3, 8)

   SUS4            = (1, 6, 8)

   # 7-based

   MAJOR\_7         = (1, 5, 8, 12)

   MINOR\_7         = (1, 4, 8, 11)

   DOMINANT\_7      = (1, 5, 8, 11)

   AUG\_7           = (1, 5, 9, 11)

   DIM\_7           = (1, 4, 7, 10)

   HALF\_DIM        = (1, 4, 7, 11)

   MAJOR\_7\_SUS2    = (1, 3, 8, 12)

   MAJOR\_7\_SUS4    = (1, 6, 8, 12)

   DOM\_7\_SUS2      = (1, 3, 8, 11)

   DOM\_7\_SUS4      = (1, 6, 8, 11)

   # 9-based

   MAJOR\_9         = (1, 5, 8, 12, 15)

   MINOR\_9         = (1, 4, 8, 11, 15)

   DOMINANT\_9      = (1, 5, 8, 11, 15)

   AUG\_9           = (1, 5, 9, 11, 15)

   DIM\_9           = (1, 4, 7, 10, 15)

   MAJOR\_9\_SUS2    = (1, 3, 8, 12, 15)

   MAJOR\_9\_SUS4    = (1, 6, 8, 12, 15)

   DOM\_9\_SUS2      = (1, 3, 8, 11, 15)

   DOM\_9\_SUS4      = (1, 6, 8, 11, 15)

   # 11-based

   MAJOR\_11        = (1, 5, 8, 12, 15, 18)

   MINOR\_11        = (1, 4, 8, 11, 15, 18)

   DOMINANT\_11     = (1, 5, 8, 11, 15, 18)

   AUG\_11          = (1, 5, 9, 11, 15, 18)

   MAJOR\_11\_SUS2   = (1, 3, 8, 12, 15, 18)

   DOM\_11\_SUS2     = (1, 3, 8, 11, 15, 18)

   # Special

   MIN\_MAJ\_7       = (1, 4 ,8, 12)

   MIN\_MAJ\_7\_9     = (1, 4, 8, 12, 15)

   MAJ\_6           = (1, 5, 8, 10)

   MIN\_6           = (1, 4, 8, 10)

   MAJ\_6\_9         = (1, 5, 8, 10, 15)

   MIN\_6\_9         = (1, 4, 8, 10, 15)

   def chordTypeToString(ct):

       if ct == ChordTypes.MAJOR:

           return "maj"

       elif ct == ChordTypes.MINOR:

           return "min"

       elif ct == ChordTypes.AUGMENTED:

           return "aug"

       elif ct == ChordTypes.DIMINISHED:

           return "dim"

       elif ct == ChordTypes.SUS2:

           return "sus2"

       elif ct == ChordTypes.SUS4:

           return "sus4"

       elif ct == ChordTypes.MAJOR\_7:

           return "M7"

       elif ct == ChordTypes.MINOR\_7:

           return "m7"

       elif ct == ChordTypes.DOMINANT\_7:

           return "7"

       elif ct == ChordTypes.AUG\_7:

           return "aug7"

       elif ct == ChordTypes.DIM\_7:

           return "dim7"

       elif ct == ChordTypes.HALF\_DIM:

           return "o7"

       elif ct == ChordTypes.MAJOR\_7\_SUS2:

           return "M7sus2"

       elif ct == ChordTypes.MAJOR\_7\_SUS4:

           return "M7sus4"

       elif ct == ChordTypes.DOM\_7\_SUS2:

           return "7sus2"

       elif ct == ChordTypes.DOM\_7\_SUS4:

           return "7sus4"

       elif ct == ChordTypes.MAJOR\_9:

           return "M9"

       elif ct == ChordTypes.MINOR\_9:

           return "m9"

       elif ct == ChordTypes.DOMINANT\_9:

           return "9"

       elif ct == ChordTypes.AUG\_9:

           return "aug9"

       elif ct == ChordTypes.DIM\_9:

           return "dim9"

       elif ct == ChordTypes.MAJOR\_9\_SUS2:

           return "M9sus2"

       elif ct == ChordTypes.MAJOR\_9\_SUS4:

           return "M9sus4"

       elif ct == ChordTypes.DOM\_9\_SUS2:

           return "9sus2"

       elif ct == ChordTypes.DOM\_9\_SUS4:

           return "9sus4"

       elif ct == ChordTypes.MAJOR\_11:

           return "M11"

       elif ct == ChordTypes.MINOR\_11:

           return "m11"

       elif ct == ChordTypes.DOMINANT\_11:

           return "11"

       elif ct == ChordTypes.AUG\_11:

           return "aug11"

       elif ct == ChordTypes.MAJOR\_11\_SUS2:

           return "M11sus2"

       elif ct == ChordTypes.DOM\_11\_SUS2:

           return "11sus2"

       elif ct == ChordTypes.MIN\_MAJ\_7:

           return "mM7"

       elif ct == ChordTypes.MIN\_MAJ\_7\_9:

           return "mM7(9)"

       elif ct == ChordTypes.MAJ\_6:

           return "M6"

       elif ct == ChordTypes.MIN\_6:

           return "m6"

       elif ct == ChordTypes.MAJ\_6\_9:

           return "M6(9)"

       elif ct == ChordTypes.MIN\_6\_9:

           return "m6(9)"

       else:

           return "XC"

**APPENDIX I**

# DATASET INITIATION SOURCE CODE: CHORD CLASS

'''

Dataset Initiation Algorithm for Nominal Note Extrapolation

CHORD CLASS: Python class representing a chord.

author: R2\_B09

'''

import Note

import ChordTypes

import scale

class Chord():

   rootNote = None

   type = None

   scaleNotes = []

   pattern = []

   def \_\_init\_\_(self, r, t):

       '''

       Constructor for chord class.

       :param r: Note n.

       :param t: ChordTypes t from enum.

       '''

       self.rootNote = r

       self.type = t

       self.scaleNotes = []

       self.pattern = []

       self.absolutePositionNotes = []

       noteNum = r.value

       # Notes of the scale this chord is in

       for note in range(noteNum - 1, noteNum + 23):

           self.scaleNotes.append(Note.Note.noteToString(scale.TWELVE\_NOTE\_SCALE[note % 12]))

       for i in range(len(self.type.value)):

           self.pattern.append(self.scaleNotes[self.type.value[i] - 1])

           #print("Added " + self.scaleNotes[self.type.value[i] - 1] + " to pattern")

           #self.absolutePositionNotes.append(scale.NOTE\_ABSOLUTE\_POSITIONS[Note.Note.stringToNote(self.pattern[i])])

       # Append the indeces of the notes in the chord from self.scaleNotes

       self.scaleNotesSelection = 0

       self.patternSelection = 0

       while self.scaleNotesSelection < 24 and self.patternSelection < len(self.pattern):

           #print('Checking self.pattern', self.patternSelection)

           if self.scaleNotes[self.scaleNotesSelection] == self.pattern[self.patternSelection]:

               self.absolutePositionNotes.append(24 if (self.scaleNotesSelection + noteNum == 24) else (self.scaleNotesSelection + noteNum) % 24)

               self.patternSelection += 1

           self.scaleNotesSelection += 1

   def getPattern(self):

       '''

       Returns a list of notes in the chord.

       '''

       s = ""

       for n in self.pattern:

           s += (n + " ")

       return s

   def getAbsoluteNotePattern(self):

       '''

       Returns the numerical representations of the notes in the chord

       (C = 1, B = 12)

       '''

       s = ""

       for n in self.absolutePositionNotes:

           s += (str(n) + " ")

       return s

   def \_\_repr\_\_(self):

       return Note.Note.noteToString(self.rootNote) + ChordTypes.ChordTypes.chordTypeToString(self.type)

**APPENDIX J**

# DATASET INITIATION SOURCE CODE: SCALE LIST

'''

Dataset Initiation Algorithm for Nominal Note Extrapolation

SCALE LIST: Contains the scale, a configuration of notes. Also contains a list of defined ChordTypes.

author: R2\_B09

'''

import Note

import ChordTypes

TWELVE\_NOTE\_SCALE = (Note.Note.C, Note.Note.C\_SHARP, Note.Note.D, Note.Note.D\_SHARP, Note.Note.E, Note.Note.F, Note.Note.F\_SHARP, Note.Note.G, Note.Note.G\_SHARP, Note.Note.A, Note.Note.A\_SHARP, Note.Note.B)

NOTE\_ABSOLUTE\_POSITIONS = {Note.Note.C: 1, Note.Note.C\_SHARP: 2, Note.Note.D: 3, Note.Note.D\_SHARP: 4, Note.Note.E: 5, Note.Note.F: 6, Note.Note.F\_SHARP: 7, Note.Note.G: 8, Note.Note.G\_SHARP: 9, Note.Note.A: 10, Note.Note.A\_SHARP: 11, Note.Note.B: 12}

CHORD\_TYPES = (

ChordTypes.ChordTypes.MAJOR,

ChordTypes.ChordTypes.MINOR,

ChordTypes.ChordTypes.AUGMENTED,

ChordTypes.ChordTypes.DIMINISHED,

ChordTypes.ChordTypes.SUS2,

ChordTypes.ChordTypes.SUS4,

ChordTypes.ChordTypes.MAJOR\_7,

ChordTypes.ChordTypes.MINOR\_7,

ChordTypes.ChordTypes.DOMINANT\_7,

ChordTypes.ChordTypes.AUG\_7,

ChordTypes.ChordTypes.DIM\_7,

ChordTypes.ChordTypes.HALF\_DIM,

ChordTypes.ChordTypes.MAJOR\_7\_SUS2,

ChordTypes.ChordTypes.MAJOR\_7\_SUS4,

ChordTypes.ChordTypes.DOM\_7\_SUS2,

ChordTypes.ChordTypes.DOM\_7\_SUS4,

ChordTypes.ChordTypes.MAJOR\_9,

ChordTypes.ChordTypes.MINOR\_9,

ChordTypes.ChordTypes.DOMINANT\_9,

ChordTypes.ChordTypes.AUG\_9,

ChordTypes.ChordTypes.DIM\_9,

ChordTypes.ChordTypes.MAJOR\_9\_SUS2,

ChordTypes.ChordTypes.MAJOR\_9\_SUS4,

ChordTypes.ChordTypes.DOM\_9\_SUS2,

ChordTypes.ChordTypes.DOM\_9\_SUS4,

ChordTypes.ChordTypes.MAJOR\_11,

ChordTypes.ChordTypes.MINOR\_11,

ChordTypes.ChordTypes.DOMINANT\_11,

ChordTypes.ChordTypes.AUG\_11,

ChordTypes.ChordTypes.MAJOR\_11\_SUS2,

ChordTypes.ChordTypes.DOM\_11\_SUS2,

ChordTypes.ChordTypes.MIN\_MAJ\_7,

ChordTypes.ChordTypes.MIN\_MAJ\_7\_9,

ChordTypes.ChordTypes.MAJ\_6,

ChordTypes.ChordTypes.MIN\_6,

ChordTypes.ChordTypes.MAJ\_6\_9,

ChordTypes.ChordTypes.MIN\_6\_9

**APPENDIX K**

# DATASET INITIATION SOURCE CODE: SAMPLER

'''

Dataset Initiation Algorithm for Nominal Note Extrapolation

SAMPLER: Makes a stratified random sample from the initialized dataset.

author: R2\_B09

'''

import scale

import chord

import ChordTypes

import random

import numpy as NP

listOfChordsOfChordTypes = []

listOfChords = []

sampleChords = []

flatSampleChords = []

notInTheSample = []

ct = []

SAMPLE\_SIZE\_G = 7

INPUTS = "../input\_dataset\_binaries.txt"

OUTPUTS = "../input\_dataset\_output\_binaries.txt"

SAMPLE\_FILE = "../samples/sample\_one.txt"

def chordgen():

   for rootNote in scale.TWELVE\_NOTE\_SCALE:

       for chordType in scale.CHORD\_TYPES:

           listOfChords.append(chord.Chord(rootNote, chordType))

   print(listOfChords)

def output\_array\_to\_file(in\_sample, out\_sample, in\_notsample, out\_notsample):

   IS\_FP = "../samples/database/NP\_INPUT\_NEURON\_VALUES.npy"

   OS\_FP = "../samples/database/NP\_OUTPUT\_NEURON\_VALUES.npy"

   INS\_FP = "../samples/database/NP\_INPUT\_NEURON\_VALUES\_NOT\_SAMPLE.npy"

   ONS\_FP = "../samples/database/NP\_OUTPUT\_NEURON\_VALUES\_NOT\_SAMPLE.npy"

   NP.save(IS\_FP, in\_sample)

   NP.save(OS\_FP, out\_sample)

   NP.save(INS\_FP, in\_notsample)

   NP.save(ONS\_FP, out\_notsample)

   print("numpy array save complete.")

def sample(SAMPLE\_SIZE):

   # Makes list of list of chords. The top level list is broken down into lists of chords grouped by type

   for type in range(0, ChordTypes.NUMBER\_OF\_CHORD\_TYPES):

       listOfChordsOfChordTypes.append([])

       for chord in range(0, ChordTypes.NUMBER\_OF\_CHORDS, ChordTypes.NUMBER\_OF\_CHORD\_TYPES):

           listOfChordsOfChordTypes[type].append(listOfChords[chord + type])

   print(listOfChordsOfChordTypes)

   print(len(listOfChordsOfChordTypes))

   for ctListIndex in range(len(listOfChordsOfChordTypes)):

       #for r in range(SAMPLE\_SIZE):

       sampleChords.append(random.sample(listOfChordsOfChordTypes[ctListIndex], SAMPLE\_SIZE))

   flatSampleChords = [\_chord\_ for \_chordList\_ in sampleChords for \_chord\_ in \_chordList\_]

   for c in listOfChords:

       if c not in flatSampleChords:

           notInTheSample.append(c)

   print("sample chords:", sampleChords)

   print("flat list:", flatSampleChords)

   print("not in the sample:", notInTheSample)

   printEqualElements(flatSampleChords, notInTheSample)

   print("Sample size:", str(len(flatSampleChords)), "(" , str(ChordTypes.NUMBER\_OF\_CHORD\_TYPES), "chord types \*", SAMPLE\_SIZE, "chords per type)")

   print(4 \* '\n')

   # ===============================================================================

   # WRITING SUBROUTINE

   inputs\_file = open(INPUTS, 'r')

   outputs\_file = open(OUTPUTS, 'r')

   inputs\_file\_contents = [i.strip().split() for i in inputs\_file]

   outputs\_file\_contents = [i.strip().split() for i in outputs\_file]

   INPUT\_NEURON\_VALUES = []

   OUTPUT\_NEURON\_VALUES = []

   INPUT\_NEURON\_VALUES\_NOT\_SAMPLE = []

   OUTPUT\_NEURON\_VALUES\_NOT\_SAMPLE = []

   temp1 = []

   temp2 = []

   print(inputs\_file\_contents)

   print(len(inputs\_file\_contents))

   print(outputs\_file\_contents)

   print(len(outputs\_file\_contents))

   print()

   with open(SAMPLE\_FILE, 'w') as sampleFile:

       counter = 0

       print(len(flatSampleChords))

       for sampleChord in range(len(flatSampleChords)):

           for inputs\_index in range(len(inputs\_file\_contents)):

               # print(inputs\_file\_contents[inputs\_index][0])

               # print(flatSampleChords[sampleChord].\_\_repr\_\_())

               if inputs\_file\_contents[inputs\_index][0] == flatSampleChords[sampleChord].\_\_repr\_\_():

                   counter += 1

                   print(counter)

                   # Name of the chord

                   print(flatSampleChords[sampleChord], end=" ")

                   sampleFile.write(flatSampleChords[sampleChord].\_\_repr\_\_() + " ")

                   # Input representation of the chord as a 24-number list

                   print("INPUTS:", end=" ")

                   sampleFile.write("INPUTS: ")

                   for i in range(1, len(inputs\_file\_contents[inputs\_index])):

                       # Print the value for bug fixing

                       print(inputs\_file\_contents[inputs\_index][i], end=" ")

                       # Append the value to be written to the file to the numpy array

                       temp1.append(inputs\_file\_contents[inputs\_index][i])

                       # Write the value to the file

                       sampleFile.write(inputs\_file\_contents[inputs\_index][i] + " ")

                   INPUT\_NEURON\_VALUES.append(temp1)

                   temp1 = []

                   # Output representation of the chord as a 14-number list

                   print("OUTPUTS:", end=" ")

                   sampleFile.write("OUTPUTS: ")

                   for k in range(1, len(outputs\_file\_contents[inputs\_index])):

                       # Print the value for bug fixing

                       print(outputs\_file\_contents[inputs\_index][k], end=" ")

                       # Append the value to be written to the file to the numpy array

                       temp2.append(outputs\_file\_contents[inputs\_index][k])

                       # Write the value to the file

                       sampleFile.write(outputs\_file\_contents[inputs\_index][k] + " ")

                   OUTPUT\_NEURON\_VALUES.append(temp2)

                   temp2 = []

                   print()

                   sampleFile.write("\n")

       print("\nNOT IN THE SAMPLE\n")

       sampleFile.write("\nNOT IN THE SAMPLE\n")

       for sampleChord in range(len(notInTheSample)):

           for inputs\_index in range(len(inputs\_file\_contents)):

               # print(inputs\_file\_contents[inputs\_index][0])

               # print(flatSampleChords[sampleChord].\_\_repr\_\_())

               if inputs\_file\_contents[inputs\_index][0] == notInTheSample[sampleChord].\_\_repr\_\_():

                   counter += 1

                   print(counter)

                   print(notInTheSample[sampleChord], end=" ")

                   sampleFile.write(notInTheSample[sampleChord].\_\_repr\_\_() + " ")

                   print("INPUTS:", end=" ")

                   sampleFile.write("INPUTS: ")

                   for i in range(1, len(inputs\_file\_contents[inputs\_index])):

                       print(inputs\_file\_contents[inputs\_index][i], end=" ")

                       # Append the value to be written to the file to the numpy array

                       temp1.append(inputs\_file\_contents[inputs\_index][i])

                       sampleFile.write(inputs\_file\_contents[inputs\_index][i] + " ")

                   INPUT\_NEURON\_VALUES\_NOT\_SAMPLE.append(temp1)

                   temp1 = []

                   print("OUTPUTS:", end=" ")

                   sampleFile.write("OUTPUTS: ")

                   for k in range(1, len(outputs\_file\_contents[inputs\_index])):

                       print(outputs\_file\_contents[inputs\_index][k], end=" ")

                       # Append the value to be written to the file to the numpy array

                       temp2.append(outputs\_file\_contents[inputs\_index][k])

                       sampleFile.write(outputs\_file\_contents[inputs\_index][k] + " ")

                   OUTPUT\_NEURON\_VALUES\_NOT\_SAMPLE.append(temp2)

                   temp2 = []

                   print()

                   sampleFile.write("\n")

       # print(INPUT\_NEURON\_VALUES)

       # print(OUTPUT\_NEURON\_VALUES)

       # print(INPUT\_NEURON\_VALUES\_NOT\_SAMPLE)

       # print(OUTPUT\_NEURON\_VALUES\_NOT\_SAMPLE)

       # Make numpy arrays of the sample

       # print(len(INPUT\_NEURON\_VALUES))

       # print(len(OUTPUT\_NEURON\_VALUES))

       # print(len(INPUT\_NEURON\_VALUES\_NOT\_SAMPLE))

       # print(len(OUTPUT\_NEURON\_VALUES\_NOT\_SAMPLE))

       NP\_INPUT\_NEURON\_VALUES = NP.asarray(INPUT\_NEURON\_VALUES)

       NP\_OUTPUT\_NEURON\_VALUES = NP.asarray(OUTPUT\_NEURON\_VALUES)

       NP\_INPUT\_NEURON\_VALUES\_NOT\_SAMPLE = NP.asarray(INPUT\_NEURON\_VALUES\_NOT\_SAMPLE)

       NP\_OUTPUT\_NEURON\_VALUES\_NOT\_SAMPLE = NP.asarray(OUTPUT\_NEURON\_VALUES\_NOT\_SAMPLE)

       print("-------------------------------------")

       print("NUMPY ARRAYS")

       print("-------------------------------------")

       print()

       print("NP\_INPUT\_NEURON\_VALUES")

       print(NP\_INPUT\_NEURON\_VALUES)

       print()

       print("NP\_OUTPUT\_NEURON\_VALUES")

       print(NP\_OUTPUT\_NEURON\_VALUES)

       print()

       print("NP\_INPUT\_NEURON\_VALUES\_NOT\_SAMPLE")

       print(NP\_INPUT\_NEURON\_VALUES\_NOT\_SAMPLE)

       print()

       print("NP\_OUTPUT\_NEURON\_VALUES\_NOT\_SAMPLE")

       print(NP\_OUTPUT\_NEURON\_VALUES\_NOT\_SAMPLE)

       print("-------------------------------------")

       # Outputs sample arrays to h5

       output\_array\_to\_file(NP\_INPUT\_NEURON\_VALUES.astype(float), NP\_OUTPUT\_NEURON\_VALUES.astype(float), NP\_INPUT\_NEURON\_VALUES\_NOT\_SAMPLE.astype(float), NP\_OUTPUT\_NEURON\_VALUES\_NOT\_SAMPLE.astype(float))

def anyTwoElementsEqual(CBL, CL):

   for i in range(len(CBL)):

       for j in range(i + 1, len(CBL)):

           if CBL[i] == CBL[j]:

               print(CL[i][0], "and", CL[j][0])

               return True

   return False

def printEqualElements(CBL, CL):

   if not anyTwoElementsEqual(CBL, CL):

       print("No elements of the two lists are the same.")

   else:

       ee = []

       for i in range(len(CBL)):

           for j in range(i + 1, len(CBL)):

               if CBL[i] == CBL[j]:

                   print(CL[i][0], "and", CL[j][0])

                   ee.append(CL[i][0])

                   ee.append(CL[j][0])

       print(len(ee) // 2)

# =============== MAIN THREAD ===============

chordgen()

if SAMPLE\_SIZE\_G < 12:

   sample(SAMPLE\_SIZE\_G)

   #writeSampleToFile(flatSampleChords, notInTheSample)

else:

   print("Sample invalid.")

**APPENDIX L**

# DATASET INTERPRETER SOURCE CODE

'''

Neural Array Dataset Interpreter for Note Extrapolates

MAIN: Converts chord input file to data that can be interpreted by the ANN.

author: R2\_B09

'''

import sys

import loc\_note\_MIDI

import pygame

from pygame import midi

from pygame.locals import \*

sys.path.insert(0, "../Dianne/")

INPUT\_FILE = "../input\_dataset.txt"

NUM\_CHORDS = 444

outputFile = None

chordsList = []

chordTypesList = []

chordObjectsList = []

chordPossList = []

chordBoolsList = []

def chordmap(input\_file, WRITE\_TO\_OUTPUT\_FILE):

   # STEP 1: MAKE LIST OF CHORDS

   input\_notes = open(INPUT\_FILE)

   #chordsList, chordTypesList = [i.strip().split(' ') for i in input\_notes]

   chordsList = ([i.strip().split() for i in input\_notes])

   for i in range(4):

       chordsList.pop()

   print(chordsList)

   print(len(chordsList))

   # STEP 2: MAKE LIST OF POSITIONS

   for chord in chordsList:

       # Numbers at which chord notes are (1 = C, 2 = C#...)

       chord\_notes = []

       for s in chord:

           try:

               # Try converting the string to a number

               num = int(s)

               # Append the positions

               chord\_notes.append(num)

           except:

               # Go to next element if the string can't be converted to a number

               pass

       chordPossList.append(chord\_notes)

   print(chordPossList)

   print(len(chordPossList))

   # STEP 3: MAKE LIST OF BINARY POSITIONS

   for \_set\_ in range(len(chordPossList)):

       poss = []

       # Put a 1 or 0 in the position

       for i in range(24):

           if (i + 1) in chordPossList[\_set\_]:

               poss.append(1)

           else:

               poss.append(0)

       chordBoolsList.append(poss)

   print(chordBoolsList)

   print(len(chordBoolsList))

   if WRITE\_TO\_OUTPUT\_FILE:

       writeToChordsFile(chordsList, chordBoolsList)

def writeToChordsFile(CL, CBL):

   SPACE\_LENGTH = 10

   try:

       with open("../input\_dataset\_binaries.txt", 'w') as chordsFile:

           for i in range(len(CL)):

               chordsFile.write(CL[i][0] + ((SPACE\_LENGTH - len(CL[i][0])) \* " "))

               for j in range(24):

                   chordsFile.write(str(CBL[i][j]) + " ")

               chordsFile.write("\n")

   except Exception as e:

       print(e)

   printEqualElements(CBL, CL)

def anyTwoElementsEqual(CBL, CL):

   for i in range(len(CBL)):

       for j in range(i + 1, len(CBL)):

           if CBL[i] == CBL[j]:

               print(CL[i][0], "and", CL[j][0])

               return True

   return False

def printEqualElements(CBL, CL):

   if not anyTwoElementsEqual(CBL, CL):

       print("No two chords have equal mappings.")

   else:

       ee = []

       for i in range(len(CBL)):

           for j in range(i + 1, len(CBL)):

               if CBL[i] == CBL[j]:

                   print(CL[i][0], "and", CL[j][0])

                   ee.append(CL[i][0])

                   ee.append(CL[j][0])

       print(len(ee) // 2)

def realTimeMIDIChords():

   '''

   Displays the mappings of the notes currently being played on MIDI.

   (kushalbhabra, 2013; Tang, n.d.)

   '''

   # display a list of MIDI devices connected to the computer

   # this function written by Stephen Tang (n.d.), stephentang.io

   def print\_device\_info():

       for i in range(pygame.midi.get\_count()):

           r = pygame.midi.get\_device\_info(i)

           (interf, name, input, output, opened) = r

           in\_out = ""

           if input:

               in\_out = "(input)"

           if output:

               in\_out = "(output)"

           print("%2i: interface: %s, name: %s, opened: %s %s" %

                 (i, interf, name, opened, in\_out))

   pygame.init()

   pygame.fastevent.init()

   event\_get = pygame.fastevent.get

   event\_post = pygame.fastevent.post

   pygame.midi.init()

   print("Available MIDI devices:")

   print\_device\_info()

   MI = int(input("Select device ID: "))  # 2 the default for Project-X99

   '''

   Examples for Project-X99:

   INPUT 2: Komplete Kontrol 1

   INPUT 3: Komplete Kontrol EXT

   INPUT 4: Komplete Kontrol DAW

   '''

   input\_id = MI

   midi\_in = pygame.midi.Input(input\_id)

   pygame.display.set\_caption("midi test")

   screen = pygame.display.set\_mode((400, 300), RESIZABLE, 32)

   print("Initializing realtime poll process.")

   realtime = True

   currentNoteNumbersInChord = []  # Current MIDI note #s in chord

   currentNotesInChord = []        # Binary mapping of notes in chord

   currentNoteNamesInChord = []    # Current note names in chord

   for i in range(24):

       currentNotesInChord.append(0)

   print("initialize:\n", currentNotesInChord, "length:" + str(len(currentNotesInChord)))

   while realtime:

       # pygame event handling

       events = event\_get()

       for e in events:

           if e.type in [QUIT]:

               realtime = False

           if e.type in [KEYDOWN]:

               realtime = False

       if midi\_in.poll():

           midi\_events = midi\_in.read(10)

           # MIDI NOTE ON

           if 0x90 <= midi\_events[0][0][0] <= 0x9F:

               mod = 1

               if len(currentNotesInChord) != 24:

                   print("Error - length of array is not 24! Emergency stop.")

                   realtime = False

                   break

               # Add note number to current note numbers in chord

               currentNoteNumbersInChord.append(midi\_events[0][0][1])

               # Add note to binary list

               root = currentNoteNumbersInChord[0] % 12

               # print("root:", str(root))

               # print("last element of currentNoteNumbersInChord:", currentNoteNumbersInChord[-1])

               # print("current note added:", midi\_events[0][0][1])

               try:

                   currentNotesInChord[(root) + (currentNoteNumbersInChord[-1] - currentNoteNumbersInChord[0])] = 1

               except IndexError:

                   currentNotesInChord[midi\_events[0][0][1] % 12] = 1

               # Add note being played

               currentNoteNamesInChord.append(loc\_note\_MIDI.Loc\_Note\_MIDI.midiNumToNote(midi\_events[0][0][1]))

               print(currentNotesInChord)

               print(currentNoteNumbersInChord)

               print(currentNoteNamesInChord)

           # MIDI NOTE OFF

           elif 0x80 <= midi\_events[0][0][0] <= 0x8F:

               print("Purging variables.")

               currentNotesInChord = []

               for i in range(24): currentNotesInChord.append(0)

               currentNoteNumbersInChord = []

               currentNoteNamesInChord = []

           # Aftertouch

           elif 0xD0 <= midi\_events[0][0][0] <= 0xDF:

               pass

           # Something else except aftertouch

           else:

               print("Something else was tinkered.")

               print(midi\_events[0][0][0])

   print("Exiting.")

   midi\_in.close()

   pygame.midi.quit()

   pygame.quit()

   exit()

if \_\_name\_\_ == "\_\_main\_\_":

   print("Real-time MIDI to ANN Input Interpreter")

   choice = input("Generate chord file? [y/n]: ")

   if choice in ['y', 'Y']:

       chordmap(INPUT\_FILE, True)

   choice2 = input("Enter real-time MIDI-ANN input mode? ")

   if choice2 in ['y', 'Y']:

       realTimeMIDIChords()

**APPENDIX M**

# DATASET INTERPRETER SOURCE CODE: OUTPUT NEURON MAP GENERATOR

'''

Neural Array Dataset Interpreter for Note Extrapolates

OUTPUT NEURON MAP GENERATOR: Generates a set of output numbers for a certain chord type.

author: R2\_B09

'''

import output\_mappings

NOTES = [0.08, 0.16, 0.24, 0.32, 0.40, 0.48, 0.56, 0.64, 0.72, 0.80, 0.88, 0.96]

CHORD\_TYPES = [

   output\_mappings.MAJOR,

   output\_mappings.MINOR,

   output\_mappings.AUGMENTED,

   output\_mappings.DIMINISHED,

   output\_mappings.SUS2,

   output\_mappings.SUS4,

   output\_mappings.MAJOR\_7,

   output\_mappings.MINOR\_7,

   output\_mappings.DOMINANT\_7,

   output\_mappings.AUG\_7,

   output\_mappings.DIM\_7,

   output\_mappings.HALF\_DIM,

   output\_mappings.MAJOR\_7\_SUS2,

   output\_mappings.MAJOR\_7\_SUS4,

   output\_mappings.DOM\_7\_SUS2,

   output\_mappings.DOM\_7\_SUS4,

   output\_mappings.MAJOR\_9,

   output\_mappings.MINOR\_9,

   output\_mappings.DOMINANT\_9,

   output\_mappings.AUG\_9,

   output\_mappings.DIM\_9,

   output\_mappings.MAJOR\_9\_SUS2,

   output\_mappings.MAJOR\_9\_SUS4,

   output\_mappings.DOM\_9\_SUS2,

   output\_mappings.DOM\_9\_SUS4,

   output\_mappings.MAJOR\_11,

   output\_mappings.MINOR\_11,

   output\_mappings.DOMINANT\_11,

   output\_mappings.AUG\_11,

   output\_mappings.MAJOR\_11\_SUS2,

   output\_mappings.DOM\_11\_SUS2,

   output\_mappings.MIN\_MAJ\_7,

   output\_mappings.MIN\_MAJ\_7\_9,

   output\_mappings.MAJ\_6,

   output\_mappings.MIN\_6,

   output\_mappings.MAJ\_6\_9,

   output\_mappings.MIN\_6\_9,

]

INPUT\_FILE = "../input\_dataset.txt"

OUTPUT\_FILE = "../input\_dataset\_output\_binaries.txt"

listOfOutputVectors = []

def chordmap(N, CT):

   i = 0

   while i < len(N):

       for chord in CT:

           # chord[0] = N[i]

           # listOfOutputVectors.append(chord)

           listOfOutputVectors.append((N[i], chord[1], chord[2], chord[3], chord[4], chord[5], chord[6], chord[7], chord[8], chord[9], chord[10], chord[11], chord[12], chord[13]))

       i += 1

   for i in listOfOutputVectors: print(i)

   print(len(listOfOutputVectors))

def writeToFile(LOOV):

   SPACE\_LENGTH = 10

   f = open(INPUT\_FILE)

   rawListChordFile = [i.strip().split() for i in f]

   for i in range(4):

       rawListChordFile.pop()

   chordsList = [rawListChordFile[i][0] for i in range(len(rawListChordFile))]

   try:

       with open(OUTPUT\_FILE, 'w') as of:

           of.write("CHORD      note ma mi au di s2 s4 b5 ~6 ~7 ~9 11 +M7 +9\n")

           for i in range(len(chordsList)):

               of.write(chordsList[i] + ((SPACE\_LENGTH - len(chordsList[i])) \* " "))

               of.write(str(LOOV[i]))

               of.write("\n")

   except Exception as e:

       print(e)

   print("File write complete.")

if \_\_name\_\_ == "\_\_main\_\_":

   chordmap(NOTES, CHORD\_TYPES)

   writeToFile(listOfOutputVectors)

**APPENDIX N**

# DATASET INTERPRETER SOURCE CODE: OUTPUT NEURON MAPPINGS

'''

Neural Array Dataset Interpreter for Note Extrapolates

OUTPUT NEURON MAPPINGS: A list of expected neuron outputs for each chord type.

author: R2\_B09

'''

# Notes

C               = 0.08

C\_SHARP         = 0.16

D               = 0.24

D\_SHARP         = 0.32

E               = 0.40

F               = 0.48

F\_SHARP         = 0.56

G               = 0.64

G\_SHARP         = 0.72

A               = 0.80

A\_SHARP         = 0.88

B               = 0.96

# Triads

#                  n ma mi a d s2 s4 b5 6  7 9 11 +M7+9

MAJOR           = [0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

MINOR           = [0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

AUGMENTED       = [0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

DIMINISHED      = [0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]

SUS2            = [0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0]

SUS4            = [0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0]

# 7-based

#                  n ma mi a d s2 s4 b5 6  7 9 11 +M7+9

MAJOR\_7         = [0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0]

MINOR\_7         = [0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0]

DOMINANT\_7      = [0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0]

AUG\_7           = [0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0]

DIM\_7           = [0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0]

HALF\_DIM        = [0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0]

MAJOR\_7\_SUS2    = [0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0]

MAJOR\_7\_SUS4    = [0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0]

DOM\_7\_SUS2      = [0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0]

DOM\_7\_SUS4      = [0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0]

# 9-based

#                  n ma mi a d s2 s4 b5 6  7 9 11 +M7+9

MAJOR\_9         = [0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0]

MINOR\_9         = [0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0]

DOMINANT\_9      = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0]

AUG\_9           = [0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0]

DIM\_9           = [0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0]

MAJOR\_9\_SUS2    = [0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0]

MAJOR\_9\_SUS4    = [0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0]

DOM\_9\_SUS2      = [0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0]

DOM\_9\_SUS4      = [0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0]

# 11-based

#                  n ma mi a d s2 s4 b5 6  7 9 11 +M7+9

MAJOR\_11        = [0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0]

MINOR\_11        = [0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0]

DOMINANT\_11     = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0]

AUG\_11          = [0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0]

MAJOR\_11\_SUS2   = [0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0]

DOM\_11\_SUS2     = [0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0]

# Special

#                  n ma mi a d s2 s4 b5 6  7 9 11 +M7+9

MIN\_MAJ\_7       = [0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0]

MIN\_MAJ\_7\_9     = [0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1]

MAJ\_6           = [0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0]

MIN\_6           = [0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0]

MAJ\_6\_9         = [0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1]

MIN\_6\_9         = [0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1]

**APPENDIX O**

# KEY NAME VK DICTIONARY SOURCE CODE FOR *PYGAME*

'''

giantwin32.py

Giant dictionary to hold key name and VK value

author: Bhabra, Kushal (2013)

'''

import win32api

import win32con

import time

# Giant dictionary to hold key name and VK value

VK\_CODE = {'backspace': 0x08,

          'tab': 0x09,

          'clear': 0x0C,

          'enter': 0x0D,

          'shift': 0x10,

          'ctrl': 0x11,

          'alt': 0x12,

          'pause': 0x13,

          'caps\_lock': 0x14,

          'esc': 0x1B,

          'spacebar': 0x20,

          'page\_up': 0x21,

          'page\_down': 0x22,

          'end': 0x23,

          'home': 0x24,

          'left\_arrow': 0x25,

          'up\_arrow': 0x26,

          'right\_arrow': 0x27,

          'down\_arrow': 0x28,

          'select': 0x29,

          'print': 0x2A,

          'execute': 0x2B,

          'print\_screen': 0x2C,

          'ins': 0x2D,

          'del': 0x2E,

          'help': 0x2F,

          '0': 0x30,

          '1': 0x31,

          '2': 0x32,

          '3': 0x33,

          '4': 0x34,

          '5': 0x35,

          '6': 0x36,

          '7': 0x37,

          '8': 0x38,

          '9': 0x39,

          'a': 0x41,

          'b': 0x42,

          'c': 0x43,

          'd': 0x44,

          'e': 0x45,

          'f': 0x46,

          'g': 0x47,

          'h': 0x48,

          'i': 0x49,

          'j': 0x4A,

          'k': 0x4B,

          'l': 0x4C,

          'm': 0x4D,

          'n': 0x4E,

          'o': 0x4F,

          'p': 0x50,

          'q': 0x51,

          'r': 0x52,

          's': 0x53,

          't': 0x54,

          'u': 0x55,

          'v': 0x56,

          'w': 0x57,

          'x': 0x58,

          'y': 0x59,

          'z': 0x5A,

          'numpad\_0': 0x60,

          'numpad\_1': 0x61,

          'numpad\_2': 0x62,

          'numpad\_3': 0x63,

          'numpad\_4': 0x64,

          'numpad\_5': 0x65,

          'numpad\_6': 0x66,

          'numpad\_7': 0x67,

          'numpad\_8': 0x68,

          'numpad\_9': 0x69,

          'multiply\_key': 0x6A,

          'add\_key': 0x6B,

          'separator\_key': 0x6C,

          'subtract\_key': 0x6D,

          'decimal\_key': 0x6E,

          'divide\_key': 0x6F,

          'F1': 0x70,

          'F2': 0x71,

          'F3': 0x72,

          'F4': 0x73,

          'F5': 0x74,

          'F6': 0x75,

          'F7': 0x76,

          'F8': 0x77,

          'F9': 0x78,

          'F10': 0x79,

          'F11': 0x7A,

          'F12': 0x7B,

          'F13': 0x7C,

          'F14': 0x7D,

          'F15': 0x7E,

          'F16': 0x7F,

          'F17': 0x80,

          'F18': 0x81,

          'F19': 0x82,

          'F20': 0x83,

          'F21': 0x84,

          'F22': 0x85,

          'F23': 0x86,

          'F24': 0x87,

          'num\_lock': 0x90,

          'scroll\_lock': 0x91,

          'left\_shift': 0xA0,

          'right\_shift ': 0xA1,

          'left\_control': 0xA2,

          'right\_control': 0xA3,

          'left\_menu': 0xA4,

          'right\_menu': 0xA5,

          'browser\_back': 0xA6,

          'browser\_forward': 0xA7,

          'browser\_refresh': 0xA8,

          'browser\_stop': 0xA9,

          'browser\_search': 0xAA,

          'browser\_favorites': 0xAB,

          'browser\_start\_and\_home': 0xAC,

          'volume\_mute': 0xAD,

          'volume\_Down': 0xAE,

          'volume\_up': 0xAF,

          'next\_track': 0xB0,

          'previous\_track': 0xB1,

          'stop\_media': 0xB2,

          'play/pause\_media': 0xB3,

          'start\_mail': 0xB4,

          'select\_media': 0xB5,

          'start\_application\_1': 0xB6,

          'start\_application\_2': 0xB7,

          'attn\_key': 0xF6,

          'crsel\_key': 0xF7,

          'exsel\_key': 0xF8,

          'play\_key': 0xFA,

          'zoom\_key': 0xFB,

          'clear\_key': 0xFE,

          '+': 0xBB,

          ',': 0xBC,

          '-': 0xBD,

          '.': 0xBE,

          '/': 0xBF,

          '`': 0xC0,

          ';': 0xBA,

          '[': 0xDB,

          '\\': 0xDC,

          ']': 0xDD,

          "'": 0xDE,

          '`': 0xC0}

def press(\*args):

   '''

   one press, one release.

   accepts as many arguments as you want. e.g. press('left\_arrow', 'a','b').

   '''

   for i in args:

       win32api.keybd\_event(VK\_CODE[i], 0, 0, 0)

       win32api.keybd\_event(VK\_CODE[i], 0, win32con.KEYEVENTF\_KEYUP, 0)

       time.sleep(0.02)

def pressAndHold(\*args):

   '''

   press and hold. Do NOT release.

   accepts as many arguments as you want.

   e.g. pressAndHold('left\_arrow', 'a','b').

   '''

   for i in args:

       win32api.keybd\_event(VK\_CODE[i], 0, 0, 0)

       time.sleep(.02)

def pressHoldRelease(\*args):

   '''

   press and hold passed in strings. Once held, release

   accepts as many arguments as you want.

   e.g. pressAndHold('left\_arrow', 'a','b').

   this is useful for issuing shortcut command or shift commands.

   e.g. pressHoldRelease('ctrl', 'alt', 'del'), pressHoldRelease('shift','a')

   '''

   for i in args:

       win32api.keybd\_event(VK\_CODE[i], 0, 0, 0)

       time.sleep(.05)

   for i in args:

       win32api.keybd\_event(VK\_CODE[i], 0, win32con.KEYEVENTF\_KEYUP, 0)

       time.sleep(.1)

def release(\*args):

   '''

   release depressed keys

   accepts as many arguments as you want.

   e.g. release('left\_arrow', 'a','b').

   '''

   for i in args:

       win32api.keybd\_event(VK\_CODE[i], 0, win32con.KEYEVENTF\_KEYUP, 0)

def event(code1):

   '''

       this function will call the win32 api keyboard event,

       basically this method will reduce the LOC's by

       generalizing the calls to simplify typer(xx,xx) function.

   '''

   win32api.keybd\_event(VK\_CODE[code1], 0, 0, 0)

   time.sleep(.05)

   win32api.keybd\_event(VK\_CODE[code1], 0, win32con.KEYEVENTF\_KEYUP, 0)

def event(code1, code2):

   '''

       This is overloaded event() funtion - where in there are two event calls instead of one !!!

       this function will call the win32 api keyboard event,

       basically this method will reduce the LOC's by

       generalizing the calls to simplify typer(xx,xx) function.

   '''

   win32api.keybd\_event(VK\_CODE[code1], 0, 0, 0)

   win32api.keybd\_event(VK\_CODE[code2], 0, 0, 0)

   time.sleep(.05)

   win32api.keybd\_event(VK\_CODE[code1], 0, win32con.KEYEVENTF\_KEYUP, 0)

   win32api.keybd\_event(VK\_CODE[code2], 0, win32con.KEYEVENTF\_KEYUP, 0)

def typer(string=None, \*args):

   ##  time.sleep(4)

   for i in string:

       if i == ' ':

           event('spacebar')

       elif i == '!':

           event('left\_shift', '1')

       elif i == '@':

           event('left\_shift', '2')

       elif i == '{':

           event('left\_shift', '[')

       elif i == '?':

           event('left\_shift', '/')

       elif i == ':':

           event('left\_shift', ';')

       elif i == '"':

           event('left\_shift', '\\')

       elif i == '}':

           event('left\_shift', ']')

       elif i == '#':

           event('left\_shift', '3')

       elif i == '$':

           event('left\_shift', '4')

       elif i == '%':

           event('left\_shift', '5')

       elif i == '^':

           event('left\_shift', '6')

       elif i == '&':

           event('left\_shift', '7')

       elif i == '\*':

           event('left\_shift', '8')

       elif i == '(':

           event('left\_shift', '9')

       elif i == ')':

           event('left\_shift', '0')

       elif i == '\_':

           event('left\_shift', '-')

       elif i == '=':

           event('left\_shift', '+')

       elif i == '~':

           event('left\_shift', '`')

       elif i == '<':

           event('left\_shift', ')')

       elif i == '>':

           event('left\_shift', '.')

       elif i == 'A':

           event('left\_shift', 'a')

       elif i == 'B':

           event('left\_shift', 'b')

       elif i == 'C':

           event('left\_shift', 'c')

       elif i == 'D':

           event('left\_shift', 'd')

       elif i == 'E':

           event('left\_shift', 'e')

       elif i == 'F':

           event('left\_shift', 'f')

       elif i == 'G':

           event('left\_shift', 'g')

       elif i == 'H':

           event('left\_shift', 'h')

       elif i == 'I':

           event('left\_shift', 'i')

       elif i == 'J':

           event('left\_shift', 'j')

       elif i == 'K':

           event('left\_shift', 'k')

       elif i == 'L':

           event('left\_shift', 'l')

       elif i == 'M':

           event('left\_shift', 'm')

       elif i == 'N':

           event('left\_shift', 'n')

       elif i == 'O':

           event('left\_shift', 'o')

       elif i == 'P':

           event('left\_shift', 'p')

       elif i == 'Q':

           event('left\_shift', 'q')

       elif i == 'R':

           event('left\_shift', 'r')

       elif i == 'S':

           event('left\_shift', 's')

       elif i == 'T':

           event('left\_shift', 't')

       elif i == 'U':

           event('left\_shift', 'u')

       elif i == 'V':

           event('left\_shift', 'v')

       elif i == 'W':

           event('left\_shift', 'w')

       elif i == 'X':

           event('left\_shift', 'x')

       elif i == 'Y':

           event('left\_shift', 'y')

       elif i == 'Z':

           event('left\_shift', 'z')

       else:

           event(i)