Recreation and Improvement of a Spiking Neural Network Composer Classification Temporal Coding

Amado Pena, Enzo Casamassima, Eri Montano
alp6659@rit.edu, ec1018@rit.edu, elm9478@rit.edu
Departent of Computer Engineering, Rochester Institute of Technology
Rochester, NY

***Abstract*— This paper recreates and improves the investigation done by [1]. Attempting to prove if computers are able to recognize and classify songs using the biological methods of how the human brain sound processing works. We discuss how the biological trait of pitch recognition is used in music theory and how the practice of Interval Ear Training can be mimicked and applied to develop a leaky-integrated-fire spiking neural network to implement music recognition and determine composers based on a dataset of songs. We found that [1] design is exclusive for classifying no more than two specific composers being a heavy limiting factor for further functionality. After recreating their design we redesigned and improved their model. We performed new tests unexplored by the original authors and achieved multiple composer classification (4 labels) with an accuracy of around 50%. Even though it's not fully accurate, it's a huge improvement from the original implementation.**

I. INTRODUCTION

The brain is heavily complex and impressive by how it processes multiple kinds of information at extreme levels of energy efficiency. Imitating brain performance in technology has attracted a lot of interest, thus having its own branch in the artificial intelligence industry known as Brain Inspired Computing. Some of the advanced functions based on biological traits had been explored and integrated such as; homeostasis, different types of plasticity, spike-time encoding and more. This paper explores one of the spike time encoding models, the leaky Integrate-and-Fire (LIF) model. This model is used to build a spiking neural network (SNN), which use spikes for encoding and processing information to frequency of spikes.

The information we process and analyze with a LIF SNN is music. Music recognition is a popular application in artificial intelligence. The question put to test is "Are Spiking Neural Networks able to recognize the composer of a specific song?". This can be accomplished through mimicking a common practice in music called Interval Ear Training. This training allows a person to recognize a relative pitch in a melody and determine the distance, duration and relation between notes. Ultimately, this allows the person to recognize a song and/or play it. Biologically this occurs due to how the human ear is able to separate the frequency components through spatial encoding in a sound signal. Our work is based of the investigation done in [1], where they present a learning mechanism that uses spiking neurons to classify composers in classical music. Their process consists of using a classification feature named the Most Common Melodic Interval Prevalence (MCMIP) [1]. With this, their system uses the MCMIP feature with an empirically identified threshold to classify two specific composers. We attempt to replicate their implementation and improve it by adding tests and extra functionality with varying levels of success.

II. BACKGROUND

A. Spiking Neural Networks (SNNs)

Neural networks (NNs) had been evolving over the years. The third generation of NNs are Spiking Neural Networks, born through the inspiration of the brain's communication scheme on how neurons interact with each other. This is done by transforming information through spikes (discrete action potentials) within a time frame instead on continuous values via adaptive synapses. SNNs hope to be the bridge that fills the missing link between machine learning and neuroscience.

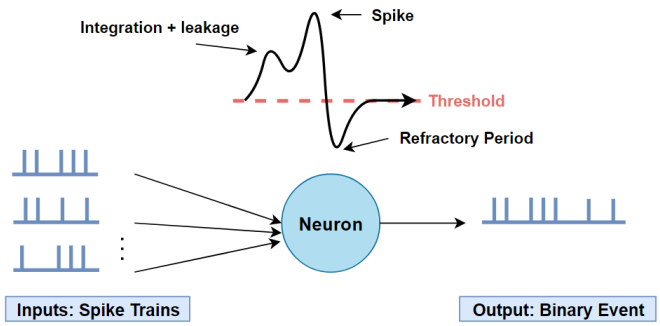


Fig. 1: SNN Graphical Representation

SNNs use biological realistic models of neurons to process information. Biologically, in our brain, a spike is generated after the sum of changes in the membrane potential of a neuron due to pre-synaptic stimulation cross a certain threshold [2]. Therefore, SNNs emulate this process in vastly simplified manner. Temporal encoding is applied to the analog input data to obtain spike trains that the network is able to read, the inputs are shown in Fig.1. When a neuron reaches specific membrane potential (voltage) counted as threshold it will spike. In this process a spike is determined by stochastic biological processes that are represented in

deferential equations. Fig.1 also shows the rest of the graphic demonstration of a single neuron of a SSN. Unlike other NNs, SNNs are more hardware friendly and energy efficient [2], they are also sparsely connected having an advantage in network topology. In addition, they are suitable for spatio-temporal event-based information and pattern recognition. That makes this kind of network perfect for this implementation.

B. Leaky Integrate-and-Fire (LIF) Model

The most popular SNN model is the Leaky integrate-and-fire (LIF) model. This model was created to improve the integrate-and-fire (IF) model which had no time-dependent memory. Where, if the model received a below-threshold signal at time *t*, the voltage boost would continuously remain until the neuron fired again [3]. The LIF model solved this problem by adding a 'leak' to the membrane potential. This phenomena comes from the diffusion of ions that happen through the membrane if a specific equilibrium of the cell was not reached. When the LIF model receives an input and it exceeds the threshold then the cell fires otherwise it will leak out any change in potential [3]. The mathematical expression of the LIF model is given by the simple formula that represents a simple resistor-capacitor (RC) circuit for a single neuron:

$$\tau_m \frac{ds_i}{dt} = -s_i(t) + RI(t) \tag{1}$$

Where τ_m is the membrane time constant, $s_i(t)$ (also seen as $v(t)$) is the membrane potential at time t, R represent the membrane resistance and $I(t)$ is the input and the leakage occurs a cause of the resistor and the integration of the input where the resistor is found parallel to the capacitor. The extended detailed version of the same equation is shown below.

$$\tau_m \frac{ds_i}{dt} = s_{rest} - s_i(t) + R \sum_j w_{ij} \sum_k \alpha(t - t_j) + RI^{ext} I(t) \tag{2}$$

This equation takes a more realistic approach where the neuron is stimulated pre-synaptic spikes that arrive at the synapses [4]. This pre-synaptic stimulation is represented by the function $\alpha(t)$, calculating the total post-synaptic total current. In addition, w_{ij} is the weight, also known as the "strength" of the synaptic efficacy that makes the connection from one neuron to another. The visual representation of the LIF model is shown in Fig.2.

C. Music Theory: Semitones & Melodic Intervals

To understand the MCMIP feature, is important to understand some basic concepts of music theory. A semitone is the smallest musical interval also known as a half step. It's the interval adjacent from one note to another; think of piano keys, counting both black and white keys.

A melodic interval is the distance in scale steps between two notes played separately one after another [1]. They are counted based on the note position on the musical staff

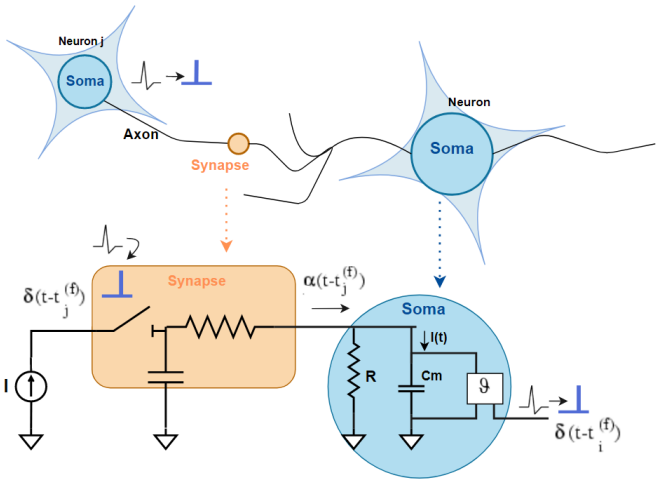


Fig. 2: LIF Model Graphical Representation

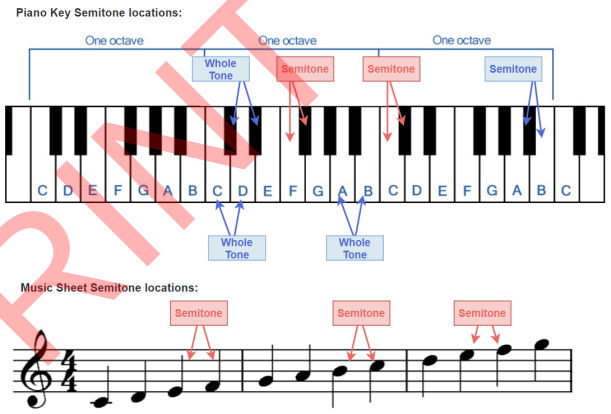


Fig. 3: Semitone locations in a piano keyboard (top) and music sheet/staff (bottom)

(which could be on the line or in between lines) making either a unison, second, third, fourth, fifth, sixth, seventh or an octave.

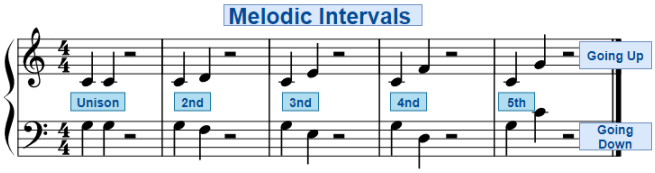


Fig. 4: Melodic intervals in a music sheet

Intervals are important as they are the building blocks of a relative pitch [5], how high or low a note is compared one to another. In music, interval ear training is crucial to identify interval in music to pick out a melody based on a familiar song, thus identifying these notes allows you to recognize the song and play it by ear. We speculate this to be the motive and inspiration of the work done by the sources cited in [1], since they were not clear why they picked melodic intervals to be their choice of music recognition.

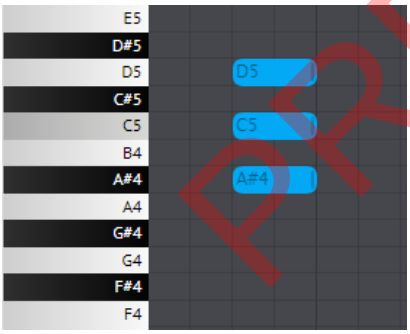
III. METHODOLOGY

A. Simulation Environment

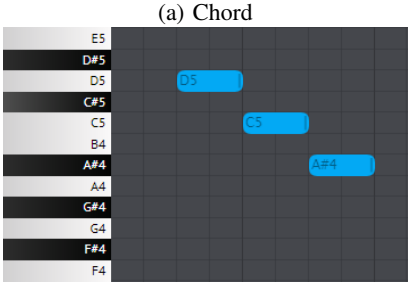
For our simulation environment we use BindsNET [6], which is a Python library enabling SNN functionality on top of PyTorch.

B. Construction of the Dataset

The dataset used in [1] was not made available, therefore we had to look at many other sources for data to use that could be closest to the original. An archive of multiple MIDI files of multiple different classical composers was obtained from [7]. We selected a total of 23 pure piano classical songs for each composer to reduce complexity and only work with a singular instrument. However, these files were not modified in any sort of way, the MIDI files were composed of multiple channels. For example, one channel could be handling the right hand and another channel handles the left hand or both channels handle both hands playing or certain parts of the song are put into different channels. In addition, if a chord were to play, this has to be broken up to be played one note at a time similar to an arpeggio. An example is shown in Fig.5a how a chord is shown in a MIDI file and the format desired shown in Fig.5b. The goal is to obtain a singular channel with consecutive tones and eliminate any unnecessary tones such as the repetition of tones while it simultaneously plays in different channels. Once broken up each note will be put into a new array list where all the notes are played one at a time to be easily distinguishable. All the data importing and MIDI handling is done by using the Python package `pretty_midi` [8]



(a) Chord



(b) Broken-up Chord to an Arpeggio

Fig. 5: Piano roll view for example chord

Furthermore, another modification is to even out the song's start time. All the songs start at different times, the ideal case is to have all of them start at zero. If a song starts playing but nothing can be heard until, for example 5 seconds in, then 5 seconds must be subtracted, essentially the song shifts backwards to start at zero. In the case of a song that starts precisely at zero, nothing happens since there is nothing to subtract, thus remaining the same. Our finalized data set has a collection of clean arrays, one array per song, each initializing at zero, and each composed of a consecutive sequence of notes. Having the data set in this particular way makes using this input with the model implementation easier in the simulation environment.

C. Pre-Processing of Music Files

After the data set is complete, the semitones are able to be identified including their duration for then finding the melodic intervals. Ultimately, both Batch and Beethoven have the same quantity of songs and the same duration per song. The difference will lay at the number of semitones played and how their melodic interval forms. After finding the melodic intervals, the input data looks like a spike train that is readable for the LIF SNN built for the composer recognition.

D. Music classification

The way music classification occurs in [1] is by Most Common Melodic Interval Prevalence (MCMIP) where they normalized the histogram across 12 melodic intervals for the classification of the specific composes Bach and Beethoven. The MCMIP fraction calculated for Bach resulted to be 0.41 and Beethoven's was 0.18. From these numbers they calculated a threshold of 0.324 where if the fraction calculated was less than the threshold then the classification would determine Beethoven otherwise it would determine Bach. The calculated threshold for our design was 0.25. The difference occurs due to the difference in songs, the selection of songs they used was not specified therefore we randomly selected 23 songs per composer. The algorithm diagram is shown in Fig.6

E. Composer Classification via SNN

[1] developed a SNN for their composer classification mechanism, shown in Fig.7. Their network consists of 4 layers to determine the MCMIP of each song. We determine the number of notes based on the number of keys of a full piano keyboard since piano songs are our target. Therefore, the first layer has a total of 88 neurons. Differently from the original implementation that has 81 starting neurons, the reason behind that number was not disclosed by the authors. Furthermore, in the same figure, each neuron has a self-excitatory synapse (blue arrow) and each one of them is connected to every other neuron by an inhibitory synapse (green arrows) [1]. When a neuron obtains a stimulus this neuron spikes and prevents all the other neurons from spiking. In addition, self excitation would cause the neuron continue spiking even if the stimulus is over. The weights for the self-excitatory synapse and the inhibitory synapses are determined when a neuron receives the input stimulus.

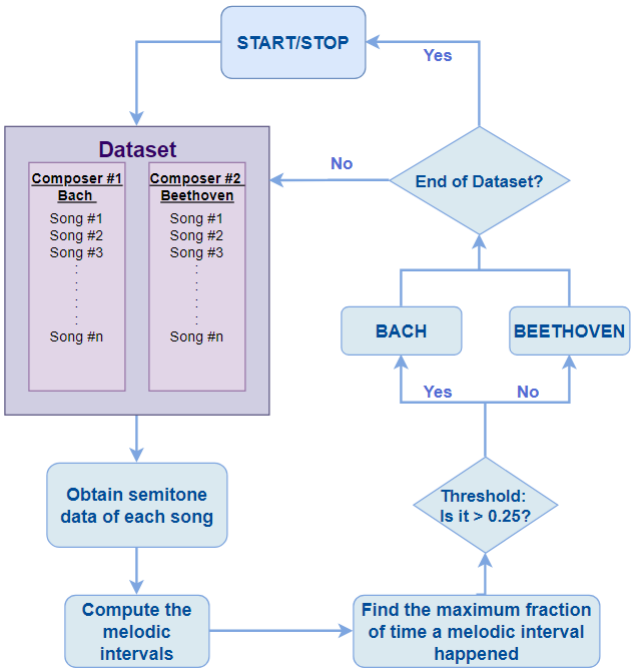


Fig. 6: Algorithm base diagram

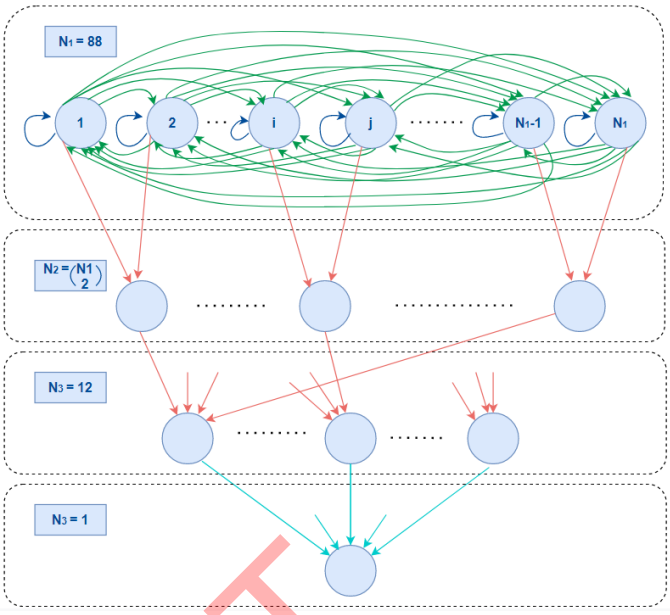


Fig. 7: SNN Model Structure

A transition period happens in between when the neuron currently being stimulated and the neuron stimulated before are both spiking simultaneously [1]. Moreover, layer 2 has a total of $M_2 = \binom{N_i}{2}$ neurons, each one of them is connected to a pair of neurons from the first layer. The synaptic weight from layer one to layer 2 is determined when all the neurons from the first layer spike simultaneously would trigger the second later to spike. The second layer's spikes partially obtain the present melodic interval's information. Then, in layer 3 we have a neuron per melodic interval and there are the first 12 melodic intervals that occur in a song [1]. All the neurons in the second layer are connected to the neurons in the third layer based on the first layer's neurons that are separated by k semitones connect to the k^th neuron in the third layer [1]. Finally all neurons in layer 3 are connected to the singular output neuron. This process remains same as the one in [1] shown in Fig.7.

1) SNN STDP Training: The synapses are the connections between the neurons, as previously mentioned they have a weight value also know as the "strength" of the synapse. While [1] isn't specific about their training model, they use an equation exactly like the Spike-Timing-Dependent Plasticity(STDP) learning rule. We apply this in both the recreated implementation and our improved implementation. The process of training the weights with STDP is abstracted and done by BindsNET.

IV. Improved Design of Composer Classification

1) Improved SNN Training & Learning: To eliminate the threshold that the authors from [1] used to classify each song by composer, we decided to use Kullback-Leibler (KL) divergence. The KL divergence is an statistical approach used to measure how different two probability distributions are [9]. It is defined in equation 3

$$D_{KL}(p||q) = \sum_i p_i \log p_i/q_i \tag{3}$$

To train our network weights, we kept the weight values equal to the SNN network model used in [1], and used STDP as the learning method to train the weights values between layers 3 and 4. For the training process, we iterated through each of the shuffled songs in the dataset, and stored the melodic interval changes for each composer to obtain the representative distributions. Then, with the trained model, and using the obtained melodic interval distribution for each composer as labels, we iterated trough all the songs again to extract the individual distributions, and used KL-divergence to compare each one against the composer's distributions. The lowest result obtained from the KL-divergence determines the model prediction label for each song. Figure 8 is a flow chart of our approach, there you can see that we did not change the SNN model structure, but only the way it predicting the composer for each song.

With this approach we increased the number of labels to 4. However, there is no limit for the amount of labels that we could use with it. We did not add more labels because we do not have to much training data and the accuracy is affected if if the labels are increased.

V. Results & Analysis

In Fig. 9 we can see the results of adding noise on the input song to the SNN. The plot contains each of the layers in order (top to bottom). We can observe that even a small amount of noise in the input layer, has cascading results in layers 2 and 3, manifesting themselves as 'columns', which indicate all the neurons are firing at that point in time and consequently making the identification of intervals impossible.

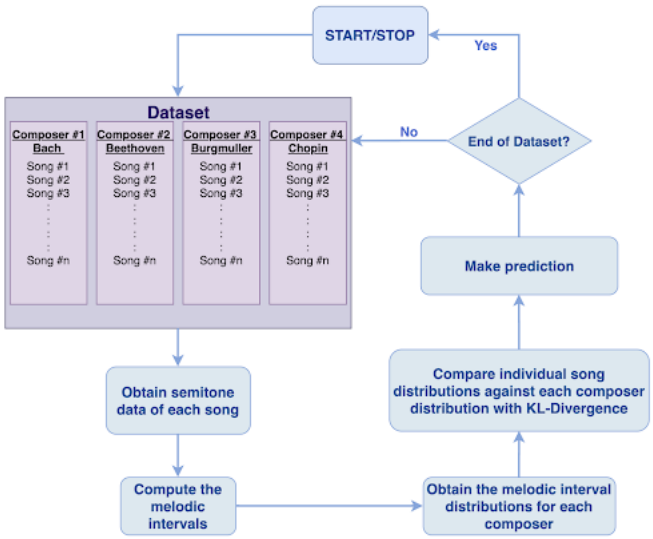


Fig. 8: Algorithm base diagram of our improved implementation.

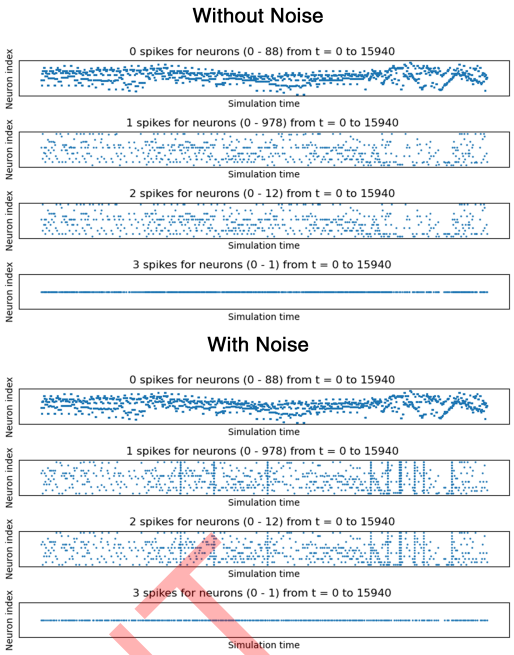


Fig. 9: Effects of noise on input data and SNN

Fig. 10 plots the results of our different experiments. By comparing our approach and the replication of [1], we can see that our method achieves significantly lower accuracy but it is more resistant to increasing amounts of noise, surpassing the original when using white noise with a standard deviation of 0.2 or more. The reason for this is that while the original implementation uses a threshold, our implementation looks at the distribution of melodic intervals and uses the KL divergence to predict the composer. In addition, our method also supports the addition of more composers, however the results show that adding more (4 composers) decreases the accuracy even further. Again, while using the KL divergence as a decision measure makes the model more robust to noise, if the distributions are too similar it can end up misclassifying a song as pertaining to another composer as seen in Fig. 11.

VI. CONCLUSION

Music recognition and other compelling human brain abilities will continue to be explored and achieved in technology. Brain inspired computing keeps making revolutionary advances in the technology industry and being able to explore this topic was rewarding. This investigation proved that it's possible for computers to recognize music and classify their composers based on pattern recognition of melodic intervals a similar process how the human brain processes music and how a human would practice interval ear training to have an enhanced pitch recognition. The approach taken by [1] was exclusive to classify only two composers of their choosing. Unfortunately this had shown lack of functional advancements. We were inspired to develop a design that had the flexibility of choosing multiple composers and successfully classify them. Our average accuracy is lower than [1] approach but that's to be expected when working with a limited dataset and multiple composers.

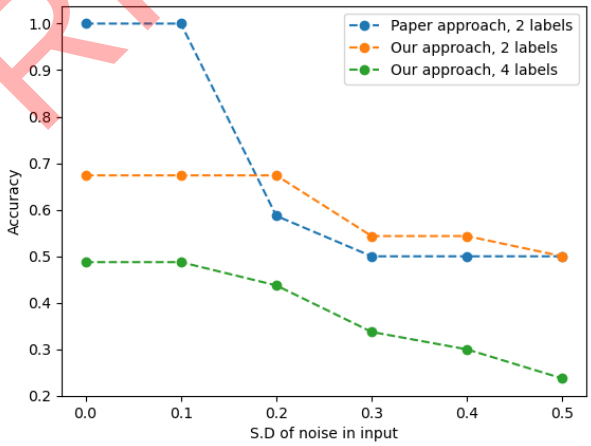


Fig. 10: Accuracy and noise resistance between implementations

VII. LIMITATIONS & FUTURE WORK

We showed that our improved model achieved to classify multiple composers. The accuracy is proof of the limitations we encountered, one of them being the size of the dataset. They trained their network using 120 songs per composer while we only used 23 songs per composer since that was what we had available. This along with the randomness of what songs we got for each composer to analyze resulted into obtaining a different threshold than theirs. We would like to work with a larger dataset that would help our accuracy value. Having a small dataset most definitely put a strain in our accuracy since there was only so much our network could work with. This also means that we'd need more time

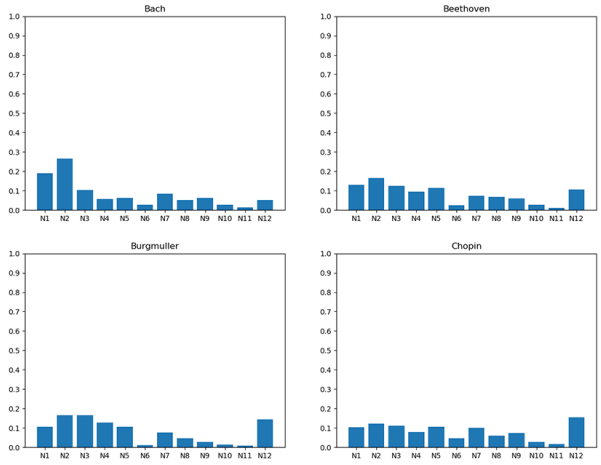


Fig. 11: Distribution of melodic intervals by composer

to train our network with lager data, one of our limitations was the time frame we had to work on this project.

ACKNOWLEDGMENTS

We thank Dr. Cory Merkel for providing informative and helpful learning material to prepare us for this project. In addition we thank the authors that ran the investigation in [1] for their clever SNN structure and coming up with a method to classify composers.

REFERENCES

[1] C. P. N, K. Saboo, and B. Rajendran, "Composer classification based on temporal coding in adaptive spiking neural networks," 2015 International Joint Conference on Neural Networks (IJCNN), 2015.

[2] A. Tavanaei, M. Ghodrati, S. Kheradpisheh, T. Masquelier, and A. Maida, "Deep Learning in Spiking Neural Networks," Jan. 2019.

[3] Zubin Bhuyan, "Neuroengineering Tutorial: Integrate and Fire neuron modeling," SlideShare, 19-Apr-2013. [Online]. Available: https://www.slideshare.net/ZubinBhuyan/x-neuro-integrate-and-fire-neuron-modeling. [Accessed: 07-Dec-2020].

[4] E. Orhan, "The Leaky Integrate-and-Fire Neuron Model." 20-Nov-2012.

[5] C. Sutton, "What's the point of Interval Ear Training?," Musical U, 03-Mar-2017. [Online]. Available: https://www.musical-u.com/learn/interval-ear-training-whats-the-point/. [Accessed: 07-Dec-2020].

[6] H. Hazan et al., "BindsNET: A Machine Learning-Oriented Spiking Neural Networks Library in Python," Frontiers in Neuroinformatics, vol. 12, p. 89, 2018, doi: 10.3389/fninf.2018.00089.

[7] Classical Archives, LLC, Classical Archives The Greats (MIDI Library). 2017.

[8] C. Raffel and D. P. W. Ellis, "INTUITIVE ANALYSIS, CREATION AND MANIPULATION OF MIDI DATA WITH pretty_midi," 2014.

[9] J. Shlens, "Notes on Kullback-Leibler Divergence and Likelihood," arXiv:1404.2000 [cs, math], Apr. 2014, Accessed: Dec. 08, 2020. [Online]. Available: http://arxiv.org/abs/1404.2000.

Amado Pena Mallen is a Computer Engineering MS student. In 2019 he moved from San Cristobal, Dominican Republic to Rochester, NY to obtain his Master's Degree. His research is about using Generative Adversarial Networks (GANs) for sequential information to learn and generate malicious netflows.

Eri Montano is a Computer Engineering MS student. Currently researching qubit mapping optimizations for quantum computers using graph neural networks. She moved to Massachusetts in 2010 from Cbba, Bolivia. She has a passion for electronics, hardware design and quantum computing. Outside of academic life she's a freelance artist.

Enzo Casamassima is a Computer Engineering MS student. His current research focuses on using adversarial machine learning and insight from human vision to better understand why A.I. models fail, and how to make them more robust.