Faculty of Natural, Mathematical and Engineering Sciences
Department of Engineering

Bush House, King's College London, Strand Campus, 30 Aldwych, London WC2B 4BG
Telephone 020 7848 2145
Fax 020 7848 2851

KING'S College LONDON

6CCE3EEP/7CCEMEEP

Individual Project Submission 2021/22

Name: Gustavo Masson Auriemo

Student Number: 19017464

Degree Programme: Electronic Engineering with Management

Project Title: Harmony Recognition through Nengo

Supervisor: Dr. Bipin Rajendran

Word count: 8170

[✓] I **agree** to the release of my project

☐ I **do not** agree to the release of my project

[✓] I **agree** to the release of my project video

☐ I **do not** agree to the release of my project video

Signature: [illegible]

Date:03/04/2022

I verify that I am the sole author of this report, except where explicitly stated to the contrary.

ABSTRACT

This report sets out a scalable method for piano harmony recognition utilizing the Neural Engineering Framework (NEF) developed by Eliasmith & Anderson and the Nengo Python package developed by Bekolay et al.

By highlighting the relevance and complexity of the problem of harmony recognition, and the applicability of the NEF for such a problem, the report aims in furthering the understanding of how biologically plausible models of cognition can perform when facing a task humans have difficulty in tackling. Moreover, it explores the way in which it learns to tackle the problem, and the subsequent conclusions that can be made if it truly models human cognition.

A proof-of-concept model is shown, along with results utilizing a dataset of 1645 piano harmony audio recordings with four different learning rates over 10 epochs. Three different approaches to consider the seriousness of error are taken and analysed in comparison to the pre-processed audio files.

In all scenarios, the model demonstrates the ability to improve on signal processing methods alone but does not demonstrate a competitive capacity to predict data it has not been exposed to. Consequently, it suggests that humans without the innate ability to recognize pitches and harmonies by ear benefit in having been exposed to sounds to be able to recognize them.

Table of Contents

INTRODUCTION

It doesn't take a specialist to be marvelled by the complexity and intricacy of some of the piano performances recorded by jazz pianist Bill Evans. Timing, voicings, and tone seem to have a life of their own, and in my own inability to emulate and transcribe what I heard, I felt powerless. That is a problem many musicians, beginners, and professionals struggle with, and something which I felt I had the tools and motivation to tackle.

Among the population of music students only about 4% have absolute pitch, which is to say they can name a note without any reference tone. This allows them, with enough training, to name the notes present in different chords (3 or more musical notes played simultaneously), and with certain ease transcribe music pieces. For the remaining 96%, it remains an arduous task which requires years of relative pitch training, technical expertise, and some trial and error. For example, in the 1970's a group of music students at the highly prestigious Berklee College of Music attempted to transcribe and compile several jazz standards into a series of books named "Real Book". To this day, errors persist, and differences arise regarding the best versions. This illustrates that even some of the best-trained musicians in the world have difficulty in carrying out this task.

In fact, it is arguably an example of the phenomenal capacity of human intelligence, leveraging perception of complex auditory scenes, cognitive ability, previous theoretical knowledge, and inference when trying new combinations (Benetos et al., 2018). The true issue is not about melody (i.e., one note played at a time), but rather about harmony (i.e., notes played simultaneously). The interaction between the different notes when played together create a complex set of harmonics (extra tones arising from the difference in frequency of the notes), which makes it extremely difficult for the ear to break it down into discrete parts. Consider that plus the vast number of combinations 88 keys (in a piano) can form, the poor quality of recordings, and fast changes, and you can begin to imagine the complexity of the problem of music transcription.

While undertaking a module on neuromorphic computing, we discussed how researchers are currently seeking ways to mimic the brain's functionality and efficiency, and how these are being implemented in software and hardware. Naturally, the idea of being able to model one's brain under different scenarios and study how accurate they were, is immensely appealing. The very "human" and complex nature of music appeared to be the perfect environment to explore these models, and further develop the knowledge within a field I am extremely passionate for.

Being presented to Nengo, a software tool that allows the implementation of biologically plausible models of cognition (Eliasmith & Anderson, 2003), proposed a new way to approach this problem and build on the work of other researchers to understand how such models would perform in face of a problem humans have difficulty in tackling.

By weaving together research in areas such as computational neuroscience, music theory, and signal processing, this report **aims in providing a proof-of-concept model for harmony transcription from piano**

audio recordings. Moreover, it aims to do so whilst **maintain a degree of fidelity to biological processes** in the brain and **attempting to explore the efficiency of such algorithms in relation to state-of-the-art techniques**.

BACKGROUND

The nature of this project is interdisciplinary, and as such, basic knowledge of each area will benefit the reader to better understand how each aspect connects to the other in a meaningful way. This section will be split into four: **Music Theory**, **Neural Networks**, **Neural Engineering Framework & Nengo**, and **Previous Work & General Remarks**.

Key words will be in bold and will be referenced throughout the report.

Music Theory

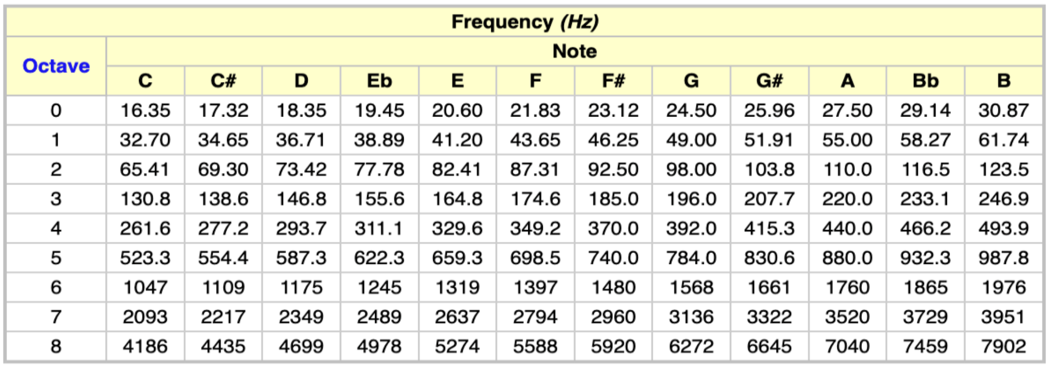
A musical **note (or tone)** is, in its most simple terms, a frequency. For example, when a string in a well-tuned guitar is plucked, it oscillates at this frequency and produces a sound. Now, if two strings are plucked at the same time, the interaction of such frequencies creates new patterns of oscillations, and subsequently new sounds. Such sounds are dependent on how far apart the frequencies between these two notes are.

It was defined that every note with a frequency ratio of 2 (or 0.5) to a subsequent note would be denoted as an **octave**, as due to their closely related harmonics they sound extremely similar (to the human ear). For example, given a note with a frequency of 440Hz, both 880Hz and 220Hz would have the relation of **octaves** to this note.

Every octave is denoted by the same letter, accompanied by a number to denote the frequency. Considering the example above, 440Hz is denoted "A4", whereas 880Hz and 220Hz are "A5" and "A3", respectively.

In the Western world, the most common way to divide the range of frequencies between two octaves is in 12 equally spaced frequencies on the logarithmic scale. This frequency ratio can be written as $2^{\frac{n}{12}}$, where n is the number of notes between the two frequencies, this is called an **interval** (n can also be called the number of **semitones** between the two notes).

Figure 1 below shows **octaves** for a range of frequencies. Note that the "accidentals" (# and b) accompanying some of the notes are a matter of how the labelling system was devised, but matters not for the purpose of this report, as frequencies and ratios will be mostly used.



Octave	Frequency *(Hz)*											
	Note											
	C	C#	D	Eb	E	F	F#	G	G#	A	Bb	B
0	16.35	17.32	18.35	19.45	20.60	21.83	23.12	24.50	25.96	27.50	29.14	30.87
1	32.70	34.65	36.71	38.89	41.20	43.65	46.25	49.00	51.91	55.00	58.27	61.74
2	65.41	69.30	73.42	77.78	82.41	87.31	92.50	98.00	103.8	110.0	116.5	123.5
3	130.8	138.6	146.8	155.6	164.8	174.6	185.0	196.0	207.7	220.0	233.1	246.9
4	261.6	277.2	293.7	311.1	329.6	349.2	370.0	392.0	415.3	440.0	466.2	493.9
5	523.3	554.4	587.3	622.3	659.3	698.5	740.0	784.0	830.6	880.0	932.3	987.8
6	1047	1109	1175	1245	1319	1397	1480	1568	1661	1760	1865	1976
7	2093	2217	2349	2489	2637	2794	2960	3136	3322	3520	3729	3951
8	4186	4435	4699	4978	5274	5588	5920	6272	6645	7040	7459	7902

Figure 1: Notes, their octaves, and their frequencies[1]

For example, between C1 and D1 we have two **semitones** ($n = 2$), which in terms of the frequency ratio is equivalent to the ratio between G1 and A1: in both cases their ratio is $2^{\frac{2}{12}} = 1.12$.

When two or more notes are played simultaneously, we call it **harmony**. Naturally, the more frequencies played at a time, the more complex their interactions become, and the harder is it to distinguish individual notes. **Harmony** and **chords** in this context will be used interchangeably as only one instrument is considered.

Convention has set names for certain combinations of notes: one may call the chord made up of the notes C4, E4, and G4, a C Major chord, for example. However, the same name would be attributed to a chord made up of C3, E4, and G5, with notes in different octaves. The latter is called a **voicing** of C Major and is extremely important in adding flair to music, something musicians often aim to do. The importance of voicings will become clearer once the literature is discussed.

Neural Networks

The field of Machine Learning at large is focused on developing efficient pattern recognition methods that can scale well with the size of the problem domain and of the data sets (Simeone, 2017). For this reason, many model classes, algorithms, and training methods have been devised, and are applied according to the purpose desired. One such model class is called Neural Network (Artificial Neural Network, or ANN for short). In simple terms, Neural Networks aim to extract the most useful features of the data and utilise such features to infer a result. This model is mostly used for situations in which it is difficult to determine good features of the data *a priori*. For example, if the input data is a text, the features may be determined *a priori* as a vector with the number of occurrences of given words, which can be a good way to determine the subject of the text. If the input was an image, however, the decision as to what good features are becomes less clear; and that is the main role of the Neural Network model.

Through a supervising signal, which determines whether the output of the model was correct or not, the parameters of the model (also referred to as weights) are updated to reflect such error, and hopefully yield

- [1] Engineering ToolBox, (2003). *Notes, Octaves and Frequencies*. [online] Available at: https://www.engineeringtoolbox.com/note-frequencies-d_520.html.

a better result. The method with which they are updated is usually by taking the gradient of a **loss function** over a small set of randomly selected data points (called mini batch) and changing the weights according to the negative direction of such gradient (i.e., minimizing the loss). This is called Stochastic Gradient Descent. The **loss function** is set *a priori* and depends on the nature of the problem. One such example is the detection-error loss, denoted $\ell(t, \hat{t}) = \mathbb{1}(t \neq \hat{t})$, which means that if prediction $\hat{t} = t$ (i.e., correct) the loss will zero, and otherwise it will be one.

Each computational unit in a neural network is called a neuron and, except for the input neurons (denoted x in Figure 2), receive a real number that is a result of the weighted sum of the outputs of the previous layer of neurons.

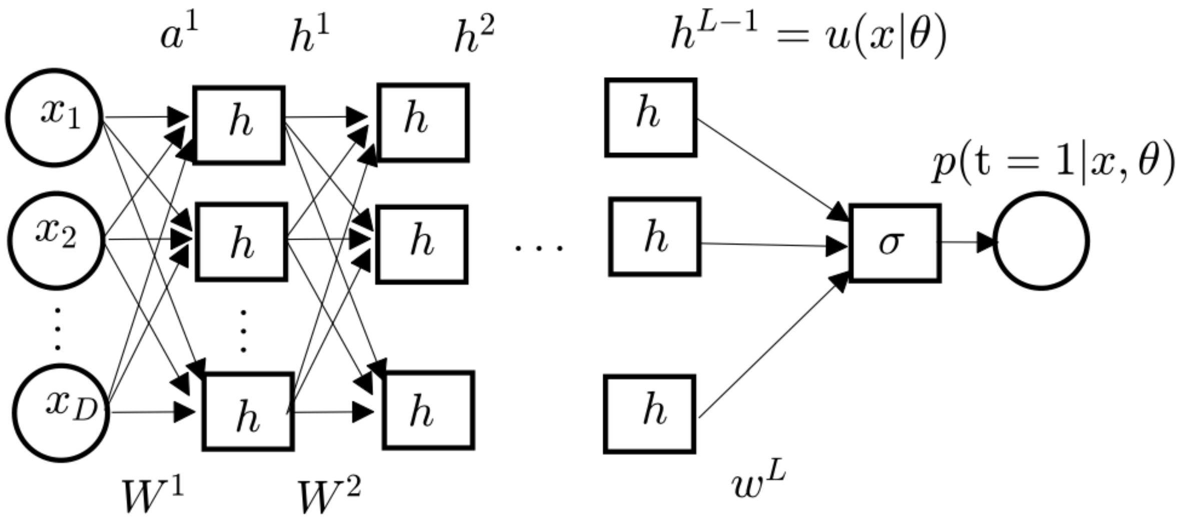


Figure 2 - Neural Network model[2]

The model in Figure 2 tackles the problem of binary classification as it can be seen by the output neuron, which denotes the probability of the target variable t being equal to 1.
Consider the first layer denoted h^1, each of the neurons h is receiving as input all the outputs of the previous layer (the input layer). The arrows for each input "represent" a **weight** (a real valued number), with which the output of the previous layer will be multiplied. The neuron then sums all these inputs and passes this real valued number (denoted a) through a non-linear (activation) function (denoted h). For a single neuron in layer h^1 this can be represented as:

$$\mathrm{h}\left(\sum_{i=1}^{D} x_i w_i\right) = \mathrm{h}(a)$$

Where w_i is the weight for each individual neuron-to-neuron connection (to a single neuron in layer h^1).

[2] Simeone, (2022)

This output $h(a)$ will take the same role x took in our example (but for the second layer), and so forth for all the layers up to $L-1$. In layer L, also called the classification layer, there will be only one neuron, whose activation function is a sigmoid, and will output the respective probability $p(t=1|x,\theta)$. Note that θ is the vector of all trainable parameters and symbolises that the output probability is dependent on these parameters/**weights**.

Naturally, Neural Networks take some inspiration from the brain by having interconnected neurons transmitting information, however, the similarities stop there. Such models do not take into consideration how biological neurons communicate, and neither how they process information. In an effort to explore the processes behind the world's most power efficient computer (the brain), a new generation of Neural Networks was devised: Spiking Neural Networks (SNNs).

As the name suggests, SNNs transmit information through spikes rather than real numbers, as it was the case in ANNs. Consequently, a few differences arise:

- The learning rules are different, and in the case of SNNs, can allow the implementation of biologically plausible methods such as Spike-Time-Dependant-Plasticity (Caporale & Dan, 2008). This difference also allows SNNs to learn in real time (online learning rules), as it does not require error calculations to take place over the whole network.
- For each computation of the function $h(a)$ in an ANN, both multiplication and addition operations are utilised. Computationally, this is an expensive process. The binary nature of spikes means that for any given input for the neurons, the output neuron is only required to sum the weights of the corresponding neurons that spiked (Rajendran, 2021). This is represented in Figure 3, where it is estimated that additions diminish by a factor of 5 to 10, and multiplications go to zero.

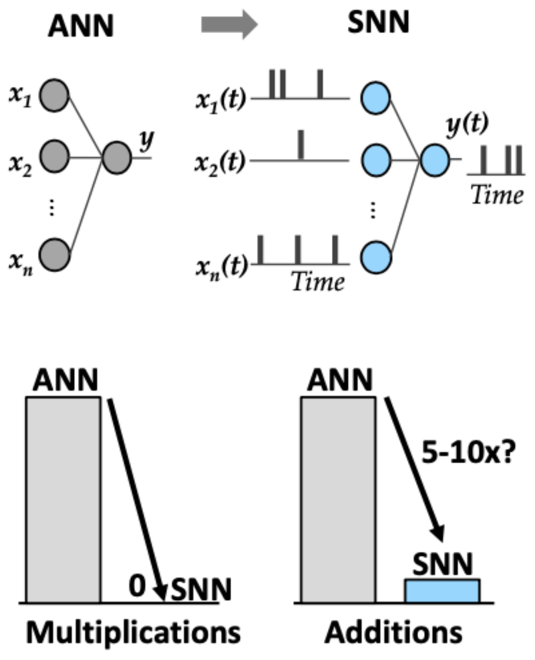


Figure 3 - ANN VS SNN (Rajendran, 2021)

The functionality of the SNN can be modelled as:

$$v(t) = f\left(\sum_{j=1}^{n} x_j w_j\right)$$

When $v(t) > V_T$, $y(t) \to 1, v(t) \to 0$, for a threshold voltage V_T.

Considering the above, SNNs tackles both an issue of computational feasibility and ability to mimic cognitive processes which are central to the successful completion of this project.

Neural Engineering Framework & Nengo

The Neural Engineering Framework (**NEF**) is a methodology that allows the construction of large-scale, biologically plausible neural models of cognition (Eliasmith & Anderson, 2003). Instead of setting connection weights between **neurons** manually or through some learning rule (such as Stochastic Gradient Descent), the NEF solves for these according to the function you desire to compute. For example, if one wants to compute $f(x) = x^2$, the framework will determine the connection **weights** between two ensembles of neurons that best approximates this function (the way in which it does will be discussed later). In the case where such function is not known, traditional methods can still be utilised.

Why model according to the NEF?

Considering that the **NEF** uses computational units based on biological neurons, any computation done by the model is forced to adhere to the basic operations that are available to neurons (Stewart, 2012). This allows insight into what sort of algorithms can and cannot be implemented in the human brain. Further, the representation of signals is done so as match the neuron ensemble behaviour seen in the brain, which is a direct effect of how neurons behave.

In 2014, Bekolay et al. developed an open-source Python package called **Nengo**[3], implementing the NEF for building and simulating such models in a computer environment. Given the goal of this project, all the models in this report were built on **Nengo**, following the **NEF**.

Previous Work & General Remarks

The current literature in the subject of harmony recognition is vast, but so is the complexity of the problem. Considering the several steps inherent to the problem; from the processing of the audio, to design choices of classification algorithms, to the scope of the solution, and the choice of databases, there were many unexplored aspects which were considered and utilised.

Most importantly, this model differs in its approach. That is, it is focused in utilising a biologically plausible approach for both the processing and the analysis the data, and for learning.

Fujishima (1999) presents **Pitch Class Profiles (PCP)**, a method that reduces the dimensionality of audio data in the frequency domain by restricting the representation of the signal to a 12-bin vector (i.e., one **octave**). The frequencies are effectively separated at their note-boundaries (i.e., $\pm 2^{\frac{0.5}{12}}$ actual frequency of the note), and the sum of their respective magnitudes is taken and assigned to a note-bin. Dixon (2003) utilises Short-time Fourier Transforms to analyse the data in the frequency domain and search for peaks in the magnitude spectrum, using these as a decision criterion and achieving 68.9% score for a 2-octave range. Barbancho et al. obtained a probability of each note of the piano being played by applying *parallel interference cancellation* (PIC) to frequency domain signals and achieved an error of 3% on predicting three-

[3] https://www.nengo.ai/

note chords within a set of 24 possibilities. Tolonand & Karjalainen (2000) proposed a model with some basis on the human auditory system for multipitch analysis by separating the signals into subband channels, although utilizing it for different purposes.
Inspired by the Cochlea in the human ear, this report chose to adhere to a similar format to that of Tolonand & Karjalainen, but adapting it to range of a piano, namely with 88 bins in **PCP**-like manner ranging from 27.5Hz to 4186Hz, one for each note of the piano. This allows for a representation of the position of the notes within the instrument and addresses the issue of representing **voicings**. Dixon's approach was also taken into consideration to extract features from the sound.

Only recently have machine learning methods (rather than relying on signal processing techniques and pattern matching), began to be used. One such research was done by Saputra et al. where the performance of different models such as Multi-layer Perceptron, Convolutional Neural Network, and Recurring Neural Networks were compared for the task of automatically transcribing 270 music pieces, reaching a score of 74% in the best model. Osmalskyj et al. (2012) managed to achieve 87% in classifying 10 different chords with a one-hidden-layer Neural Network and pre-processing utilizing the **PCP** of the audio files. Kong et al. (2022, ByteDance) achieved the state-of-the-art score of 96.72% utilizing a dataset with over 38 million notes by training their model, beating Google's previous score of 94.80%.

Methodology

Dataset and feature extraction

Datasets utilised in the current literature vary significantly, from full recorded pieces, to single notes, to MIDI versions of music pieces. Considering the goal of this project, namely harmony recognition, a few base characteristics were outlined for an ideal dataset:

1. Simultaneous playing of the notes
2. Full range of piano utilised
3. Labelling of the notes present
4. Recorded, as opposed to computer-generated
5. Different playing dynamics present

Since the purpose of the project was specifically for harmony recognition, databases containing music pieces were not suitable as they all incorporated melodies (to the extent that research revealed). The UMA-piano chord database developed by Barbancho et al. specifically focuses on harmony and contains over 275,000 chord recordings with different levels of polyphony and playing styles. Further, all the recordings are from a live piano in contrast to a MIDI, which allows for complex harmonic interactions to happen, and a range of dynamics – staccato, muted, sustained, etc., were recorded, which allows for a greater fidelity to the way musicians play.

Chords ranging from 3 to 10 notes are present, however, for the sake of providing a proof-of-concept, computationally feasible model (rather than a fully functioning model), only chords with 3 notes were considered. This yielded a dataset with 19503 different audio recordings.

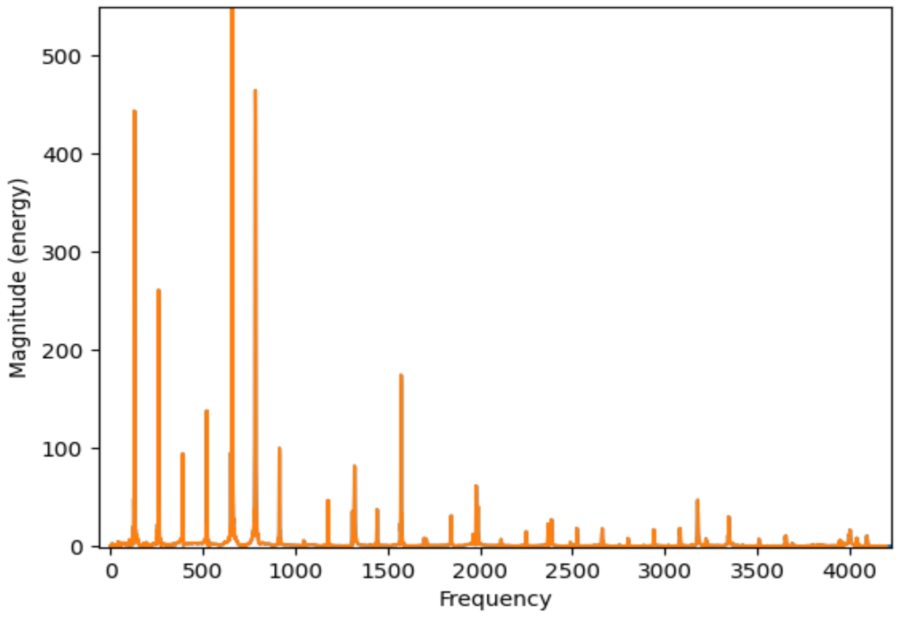


Figure 4 - Magnitude Spectrum of Audio File

Figure 4 shows an example of how an audio file was processed. Given all the audio files are in the *.wav* format, which have a sample rate 44.1kHz, the sampling frequency of the magnitude spectrum was also set to this value. By restricting the range of frequency responses to values equivalent to the range of a piano, namely from 27.5Hz to 4186Hz, a representation of the real-valued magnitudes associated with each frequency is obtained. Further, a dictionary with the precise frequencies of each piano note was created, and a boundary of $\pm 2^{\frac{0.5}{12}}$ (i.e., half a note) was set, creating a bin for each note. All the frequency values from the magnitude spectrum were assigned to a bin, and their respective magnitude values were summed, yielding a vector of 1×88 real values. This can be seen in Figure 5 below:

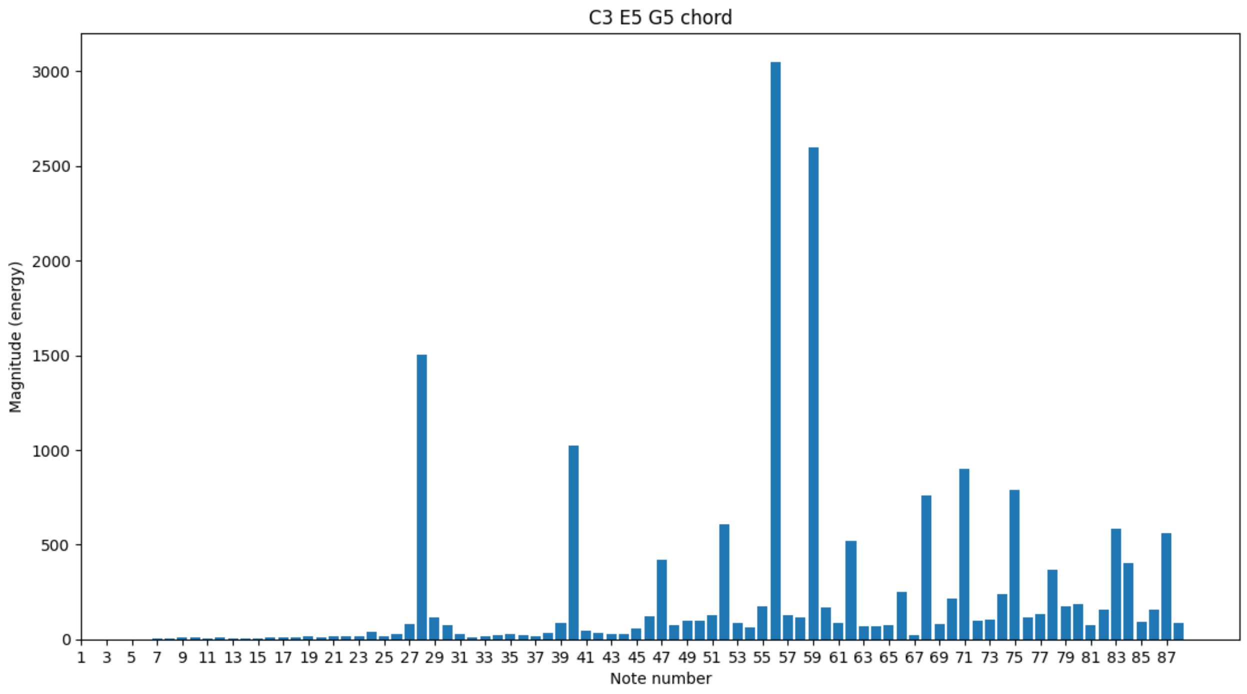


Figure 5 - Processed audio representation

The three highest peaks correctly identify the chord being played in this scenario, namely, note numbers 28, 56, and 59, or C3, E5, and G5. As a result, a matrix of 19503×88 was obtained and is available as "full_poly3-A-Eb-C.mat" on the Github repository in the Appendix.

Finally, it became clear that utilising the full range of the piano (88 notes) was not feasible in the hardware available, such that only the first 30 notes were selected as input. Consequently, the dataset had to reflect chords whose notes were all present in this range, this led to a smaller dataset of 1645 audio recordings instead.

The training and testing sets were planned to have a 70/30 split, but due to a miscalculation which was noted after the training had been done, they were divided as 77/23 (1266/379). Nonetheless, both sets had a ratio of samples per note per possible combinations in line with the research (Chen & Su (2018), Osmalskyj et al. (2012)).

Framework

Leaky-Integrate-and-Fire Neuron Model in the NEF

There are several different neuron models, with different characteristics and idiosyncrasies. For this project, the standard Leaky-Integrate-and-Fire (LIF) was chosen as it demonstrates characteristics of biological neurons such as the diffusion of ions through the membrane (leaky), a refractory period, and the action potential, whilst being simple enough to model without computational constraints. It is modelled by the equation:

$$C_M \frac{dV}{dt} = -g_l(V - V_l) + I_{app}(t)$$

Where V_l is the resting potential, g_l is the leak conductance, $I_{app}(t)$ is the external input current, C_M is the membrane capacitance, and V is the membrane potential.

The spikes in the model are triggered by the membrane potential reaching a threshold called the action potential. Given the external input current, that extends to a notion of *current* threshold, where if $I_{app}(t) > I_{th}$, it will cause the neuron to start firing (as the increase in voltage becomes greater than the decay). Subsequently, for higher values of $I_{app}(t)$ we see a higher frequency in the spikes — as modelled by the response curve in Figure 6.

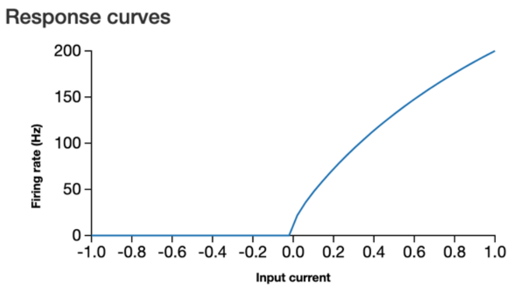


Figure 6 - Response Curve for one neuron

The NEF makes a strong claim that this input current is a linear function of the value x being represented, given by the equation:

$$I(x) = \alpha x + I_{bias}$$

Where α is some gain value and I_{bias} is a constant bias current, both of which are randomized to generate heterogeneous response curves.[4]

In Nengo, the value for which the neuron will start firing is given by the value of x for which $I(x) > I_{th}$, that is:

$$x > \frac{I_{th} - I_{bias}}{\alpha}$$

[4] https://forum.nengo.ai/t/deriving-spiking-voltage-and-spiking-frequency-of-a-neuron-from-a-train-nengo-dl-model/1898/4

As a default, $I_{th} = 1$, which yields:

$$x > \frac{1 - I_{bias}}{\alpha}$$

For example, by setting $I_{bias} = 0.6$, and $\alpha = 1$ for a single neuron, we obtain:

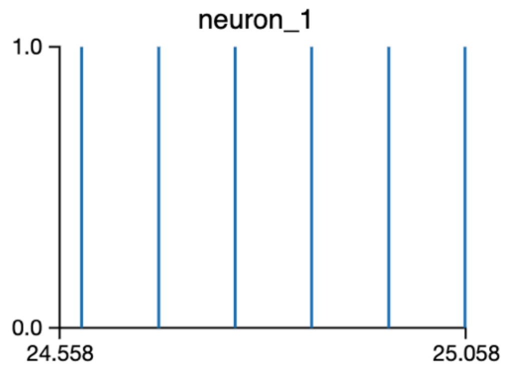


Figure 7 - Spike response for x=0.41

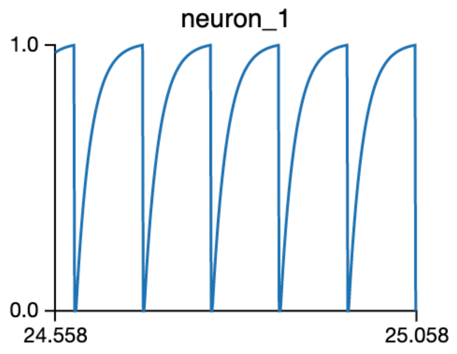


Figure 8 - Voltage for x=0.41

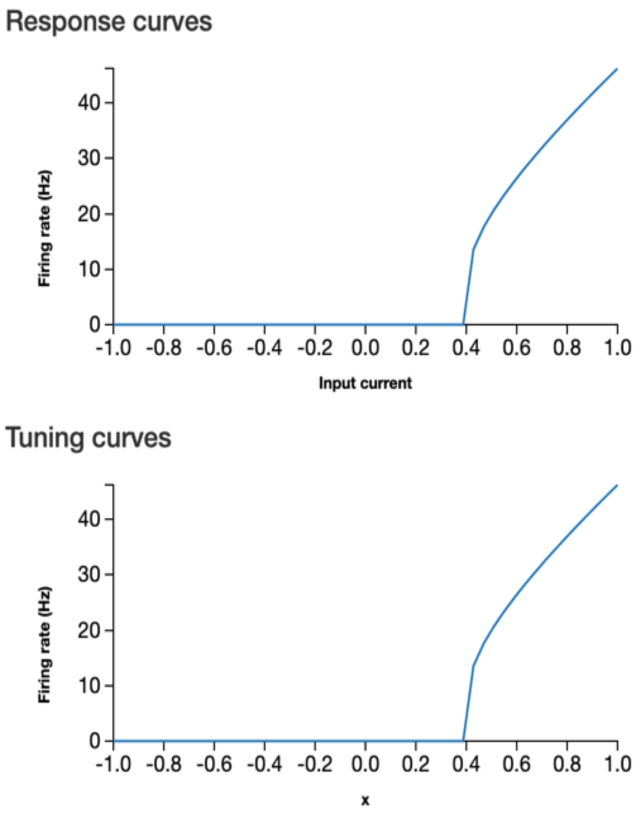


Figure 9 - Response and Tuning Curves

As expected, Figures 7, 8, and 9 show how the neuron starts spiking for values of x greater than 0.4 (but not equal to) and the rate remains constant for a non-changing x, which is in line with the discussion above.

Let us now consider more complex representations, where the input has more than one dimension, and the ensemble of neurons has more than one neuron. For a signal vector $\mathbf{x}$, the NEF uses an encoding vector $\mathbf{e}_i$ to represent neuron activity a_i of a given neuron i. That is, it randomly specifies a "preferred direction vector" for that neuron, which will affect the way it responds to signals (Stewart, 2012). Effectively, the spiking activity by a neuron will be given as:

$$a_i = G(\alpha_i \, \mathbf{e}_i \cdot \mathbf{x} + I_{bias})$$

Where the function G is the spiking LIF neuron model, which maps input current to spiking activity.

Figure 10 below shows how an actual ensemble with 20 neurons behaves in Nengo. The parameters for each neuron are set at random to provide a realistic range of responses. In turn, by randomizing them, it allows the ensemble to decode/represent a larger range of functions (i.e., to respond in different ways to values).

ω_i

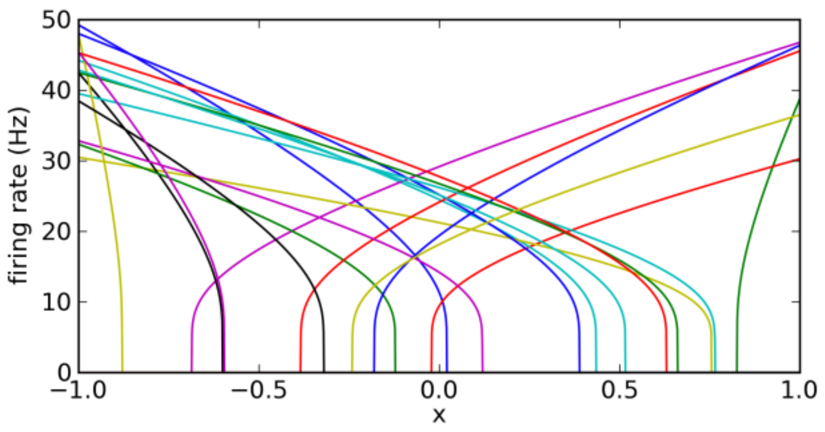


Figure 10 - Tuning Curves for an Ensemble[5]

If all neurons had an intercept at 0 for example, the gradient of the tuning curves for values surrounding 0 would be similar, and less clarity in reflecting changes in the functions around this range would ensue. Neurons whose firing rate increase with x have $\mathbf{e}_i = 1$ while the others have $\mathbf{e}_i = -1$ (Stewart, 2012).

Decoders

In the same way that we seek to map input $\mathbf{x}$ into spiking activity, we seek to map spiking activity back into some value (which could be simply $\mathbf{x}$ again or some function of thereof). To do so, a set of decoding weights $\mathbf{d}_i$ is determined such that:

$$f(\mathbf{x}) = \sum_i a_i \boldsymbol{d}_i$$

These decoders are found through a least-squares minimization of the difference between the decoded estimate and the actual values of $f(\mathbf{x})$:

$$\mathbf{d} = \Upsilon^{-1}\Gamma \qquad \Gamma_{ij} = \int a_i\, a_j dx \qquad \Upsilon_j = \int a_j f(\mathbf{x}) dx$$

Where a random sample of possible $\mathbf{x}$ values are considered.
(Bekolay et al.,2013).

Applying this concept to connections between **ensembles of neurons**, it is clear to see that by considering the decoding weights, which allow us to retrieve representations from neural activity, and encoding weights, which allow us to obtain a neural activity representation from input values (with a gain factor), it is possible to derive the following connection weights between two ensembles:

$$\omega_{ij} = \alpha_j \mathbf{e}_j \mathbf{d}_i$$

[5] Stewart, 2012

Where indexes j correspond to the neurons in postsynaptic ensemble, and indexes i correspond to neurons in the presynaptic ensemble; α_j is the gain factor corresponding to the neuron j.

Learning

Given the framework discussed, one should wonder how to compute a set of decoding weights in a situation where the function is unknown. In other words, how to minimize the error in approximating a function $f(\mathbf{x})$ where only the output is known.

By denoting a vector $\mathbf{E}$ as the error we want to minimize, MacNeil and Eliasmith (2011) proposed the following learning rule:

$$\Delta \mathbf{d}_i = \kappa \mathbf{E} a_i$$

$$\Delta \omega_{ij} = \kappa \alpha_j \mathbf{e}_j \cdot \mathbf{E} a_i$$

Where κ is a scalar learning rate, and all other terms are as before.

By depending only in information local to the neuron, as opposed to global information of the neuron population, this rule allows for biologically plausible, real-time learning of the set of decoders weights that best approximates the function we want to model. In the NEF it is called the Prescribed Error Sensitivity (PES) rule and was the chosen method for the model in this project due to the characteristics mentioned previously.

Design

The design of the model has two time-varying input signals, the hearing stimulus, which receives a scaled 30-dimensional vector representing a chord every 150ms, and a "correct" stimulus, which receives a 30-dimensional vector with ones in the correct position of the notes every 150ms (discussed later). For the training model, these are connected to an error ensemble that allows the learning process, whereas in the inference model only the hearing input and a decision ensemble are present. This allows for real-time detection of the notes once the decoder values are trained.

Due to the nature of the tuning curves of the neuron models in the NEF, which by default respond to input values between -1 and 1 (Figure 11), scaling them to respond to larger values reduces the quality of the representation. This is because the range of frequencies for the neurons are limited, and the further it is spread, the lower the resolution of the output.

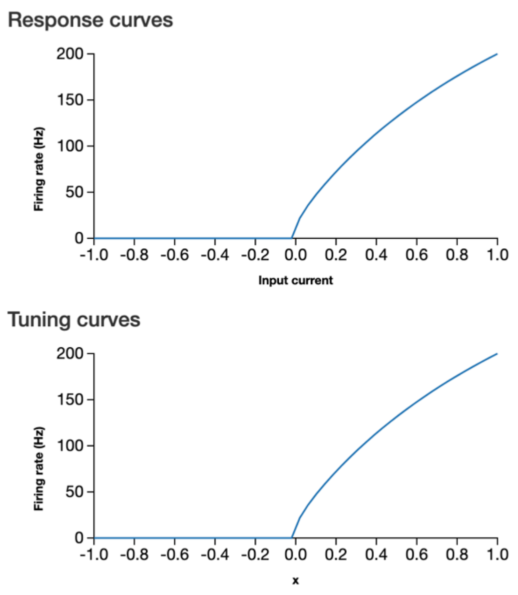


Figure 11 - Neuron with Radius parameter = 1.

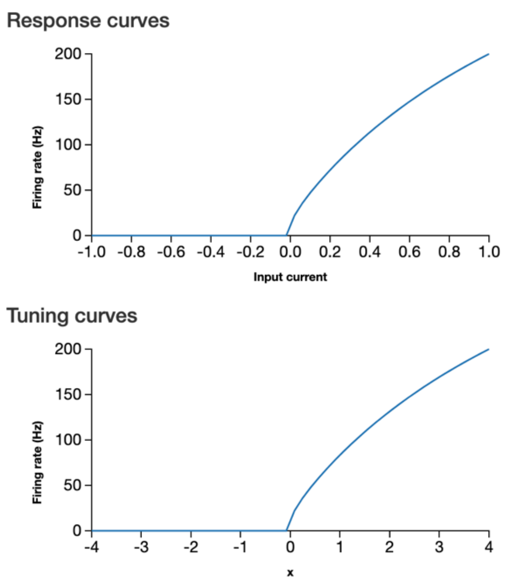


Figure 12 - Neuron with Radius parameter = 4.

Refer to Figures 11 and 12 above, both neurons were set to have the same parameters for maximum fire rate, axis intercept, and encoders. Notice how by increasing the range of input response to 4 (Figure 12), it spreads the firing rate over a large output domain, losing resolution. The only way to fix this issue for ensembles is by increasing the number of neurons, and subsequently the computational load.

The linear relation between input current and the value being represented can be seen by how the curves maintain their shape for both Response curves and Tuning curves regardless of what maximum values they were set to represent.

Considering the discussion above, the hearing input stimulus given by the 30-bin magnitude values of the Fourier transform was scaled with a log_{10} function to restrict the values to a range between 0 and 4 (decimal values were set to 0 to prevent negative numbers). This allowed for a smaller number of neurons to be used for representing the signal. Consequently, the label vector, which was initially made up of ones was scaled by 3 to match the highest magnitudes of the hearing stimulus.

As mentioned, due to computational constraints, the range of the model was reduced to 30 notes instead of 88, however, the methodology would remain the same for the latter. This is crucial, as it allows the model to scale and tackle this problem with a larger number of notes.

Figure 13 below shows a visual representation of the learning model while running:

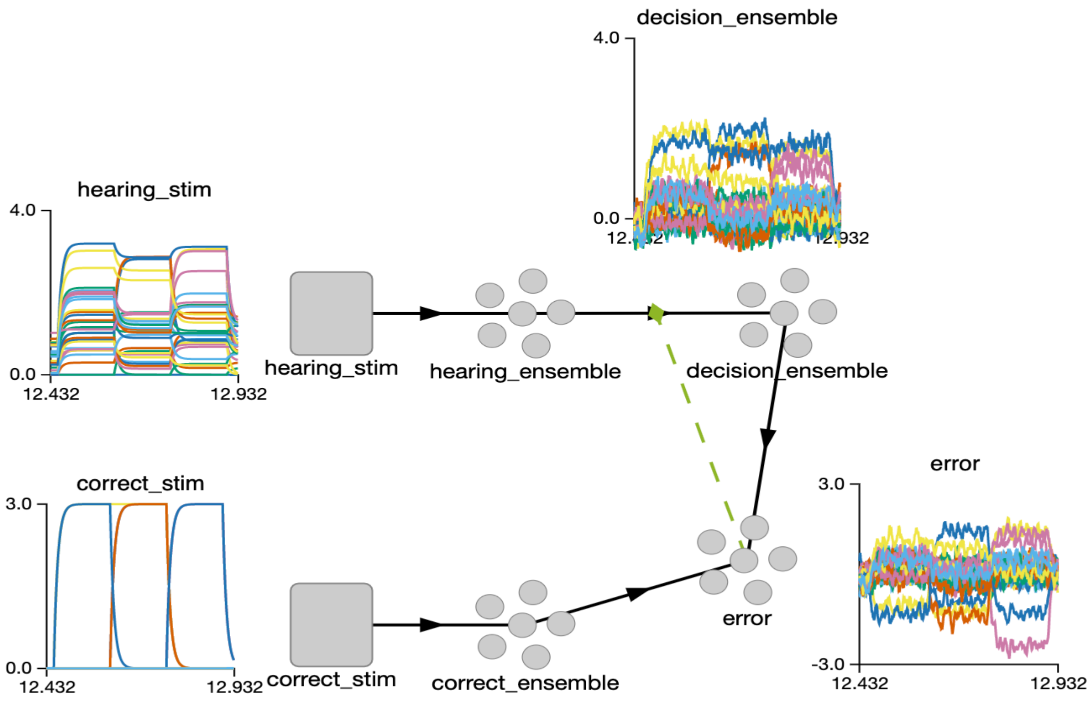


Figure 13 - Learning Model

Blocks made up of the six circles are neuron ensembles, and the rectangles are signal nodes.

Beside each block is a visual representation of the values it is representing. Each colour represents one dimension of the 30 inputs (i.e., one value of the 30-dimensional vector).

As it can be seen, both the "hearing" signal and the label signal are changing every 150ms, shown by the increase and decrease of the coloured lines. As expected, the range of values in the hearing signal are less clear, and the maximum values vary depending on the row of the data matrix that is being represented.

The application of the learning rule can be seen by the green arrow from "error" to the connection between "hearing" and "decision". The error vector defined previously in the PES rule is given by a sum of the label vector (correct_ensemble) multiplied by -1 and the "decision_ensemble" vector; this can be seen in the graph besides the error ensemble. By utilising this error signal with the PES rule, the decoders between the hearing ensemble and the decision ensemble learned how to improve the clarity of the peaks of three notes present in the chord — suppressing the ones that shouldn't be present and highlighting the ones that should.

For training, four different learning rates (LRs) were selected based on highest possible learning rate discussed by Stewart on the "Simulating Neurobiological Systems" lecture series[6], decreasing by a factor of 25 from the highest to the lowest: 5e-4, 2e-4, 8e-5, and 2e-5.

Overall, each simulation ran for 10 epochs of 190 seconds (simulation time). That is, each set of decoders was exposed to 12,660 chord representation values by the end of training (1266 chords per epoch).

The LRs decreased by 9% per epoch to allow larger weight changes to happen in the beginning, and fine-tuning to happen towards the end. The reasoning behind this rate of change was to limit the lowest value of the learning rate to be 10% of the highest value, effectively modelling how significant a 10-fold change in this parameter would be for the decoders.

Ideally, a deeper analysis into suitable ranges for the learning rates would be conducted, however IT issues arose which limited the capacity to carry out further research into the matter. Nonetheless, methods to analyse the results with the given LRs were devised to gain insight into how the learning process affected the prediction ability of the model.

Once the decoders were trained, they were saved into a matrix and loaded into the model in Figure 14 for the connection between hearing_ensemble and decision_ensemble.

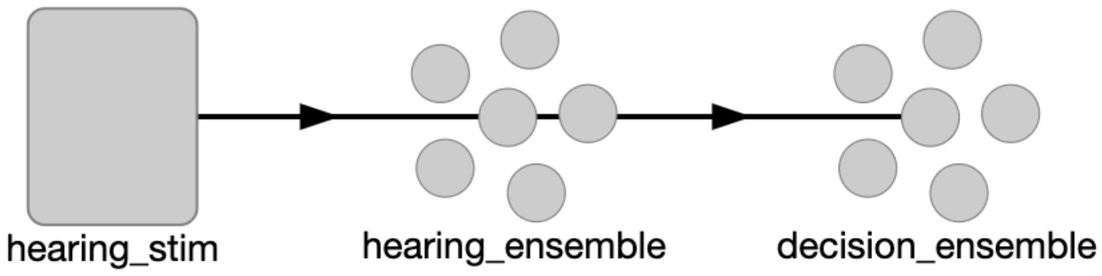


Figure 14 - Prediction model

The prediction of the model was given by index of the three values with the highest magnitude (i.e., the notes with the highest magnitudes). Both the training dataset and the testing data set were processed by the model under the different set of decoders trained previously, and the results will be discussed briefly.

[6] https://www.youtube.com/watch?v=rdJxFaJk9u4&t=2830s&ab_channel=terrencecstewart

Seriousness of error

As it is the case with the human process of music transcription, there are many different mistakes one can make, some of which are more serious than others. To reflect this **seriousness of error**, three different loss functions were devised to judge the predictions of the model:

1. **Detection-error loss (DTE)** → for each correct note prediction of a chord the loss was 0, and 1 otherwise (for a maximum error value of 3 per chord). That is, it is measuring the percentage of True Positives.
2. **Distance loss (DL)** → by comparing the notes of the predictions and the labels, the error was calculated based on the distance in terms of number of semitones (modulus 12) to the closest match.
3. **Consonance loss (CL)** → based on the research of Plomp & Levelt (1965) on tonal consonance (how similar intervals sound to the human ear), an error scale was devised from most consonant to least consonant intervals. Like the DL method, the notes in the prediction were compared to the labels, and the values that yielded the highest consonance values (lowest loss modulus 12) were considered. The equivalence to DL is mentioned in the Appendix.

Whereas DTE can be simply averaged, the values for both DL and CL range from 0 to 12, and should not necessarily carry the same "seriousness of error". As such, two different ways to turn these losses into error values between 0 and 1 were considered:

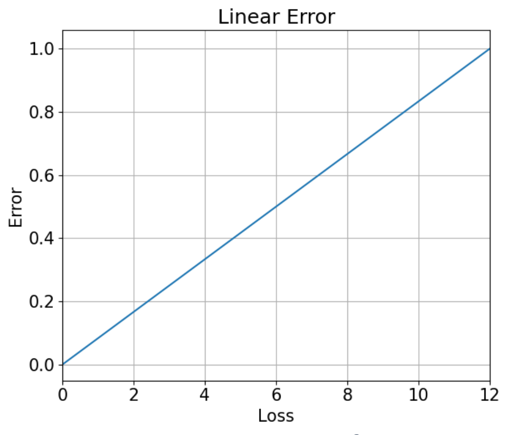


Figure 15 - Linear error function

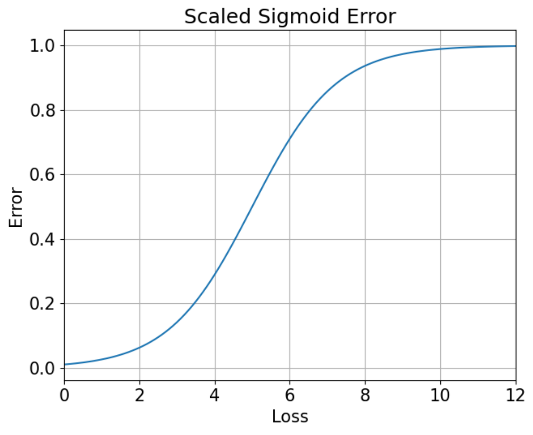


Figure 16 – Scaled sigmoid error function

Where the function in Figure 15 is given by the equation $f(x) = \frac{x}{12}$, and the function in Figure 16 given by the equation $f(x) = \frac{1}{1+e^{4.5-(0.9x)}}$.

This allows for distinct approaches in understanding how the model is working. For example, for the Scaled sigmoid function, values close to a loss of zero have a small error, whereas values greater than 10 already have an error very close to 1. This gives a clear weighting to the seriousness of loss, attaching less

importance to predictions that were wrong but close to the correct value, and more importance to significant mistakes.
Consider a scenario where the Scaled Sigmoid Error version of the Distance Loss went up, whereas the Linear Error version didn’t go up by as much. One can infer from these results that the loss was with a value greater than 4. When considering this in evaluating the performance of the model over different epochs, it allows the visualization of how the model learns over time.

Findings

The project set out to achieve the following three things:

1. Develop a proof-of-concept model for harmony transcription from piano audio recordings.
2. Maintain a degree of fidelity to biological processes in the brain.
3. Explore the efficiency of such algorithms in relation to state-of-the-art techniques.

Point number 2 was explored in detail in the Methodology and Design sections. Concurrently, the results of simulations revealed that in addition to the characteristics of the framework, the neurons in the model demonstrated a spiking behaviour in line with biological neurons, with each one spiking at an average rate of 92.60 times per second with the training dataset, and 91.36 times with the testing dataset.
Regarding points 1 and 3, an analysis of the results obtained will be done, and then a comparison and evaluation against literature results mentioned in the Background section will be carried out.

The base comparison for all predictions of the model is the processed audio signal (30-dimension vector of magnitudes) which was fed into the hearing ensemble, without any trained decoders. When considered against the labels, it achieved a score of 33.9% with the detection-error loss function over the training set, and 9.3% over the test set. This already poses an interesting question: why is it that simply changing the sets affects the prediction score so heavily? The first instinct may be to assume that the size of each set is to blame. However, this was easily proven wrong by simulating the model over a subset of the training set of equal size to that of the test set, where it achieved a score of 32.9%. Since the only factor affecting the decision of the model in this scenario is the magnitude of the notes given by the Magnitude Spectrum, these results imply that different parts of the data have different degrees of clearness (of the peaks). Logically, this is due to the nature of the dynamics of the recordings, which are the only factor affecting the sounds. That is, a poor distribution of these sounds was done. Overall, however, it should not affect the conclusions that can be taken from the data, as the reference point will always be the results of the model without decoders, and the improvement upon thereof.

Five different simulations for each learning rate were done for both the test set and the training set:

- Detection-error loss (DTE)
- Distance-loss under the sigmoid function (DL)
- Distance-loss under the linear function (LDL)
- Consonance-loss under the sigmoid function (CL)
- Consonance-loss under the linear function (LCL)

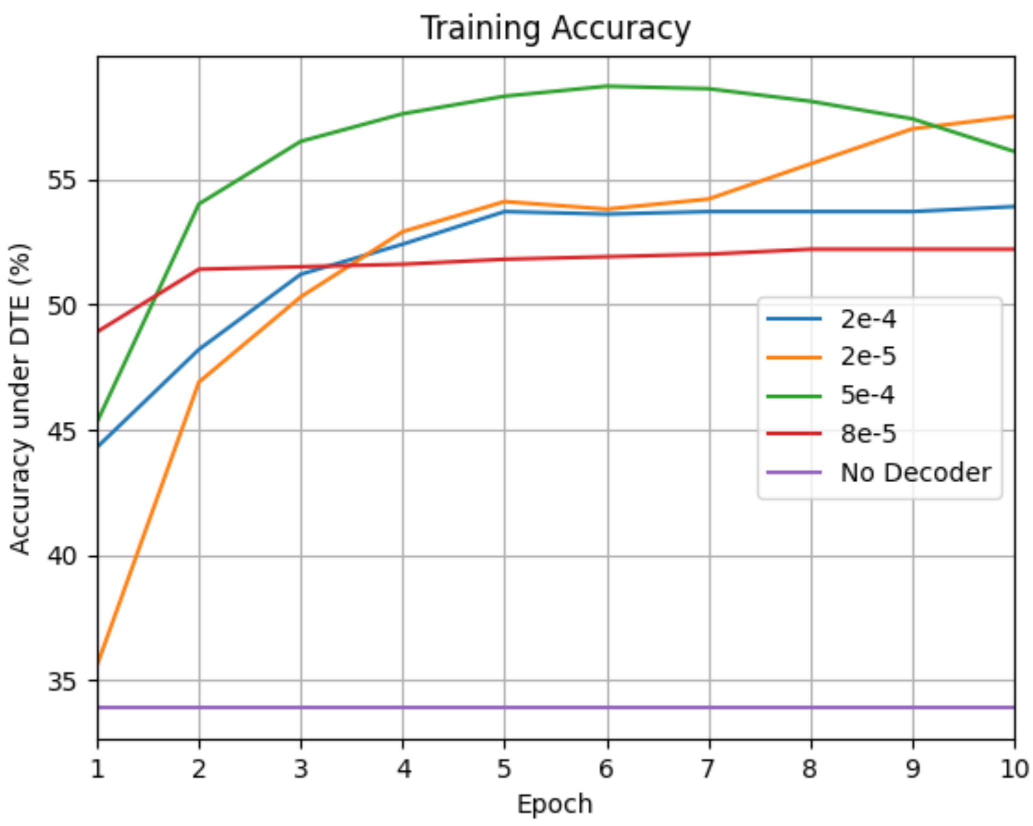


Figure 17 - Training data results under DTE

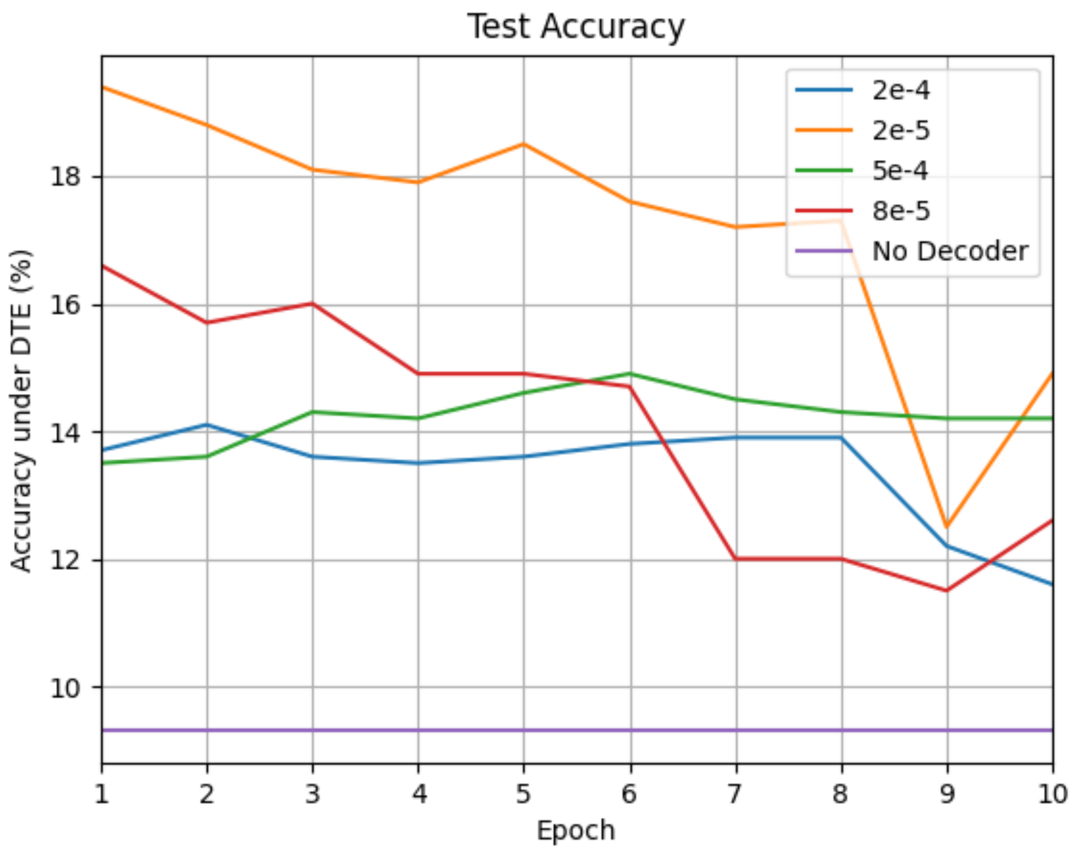


Figure 18 – Test data results under DTE

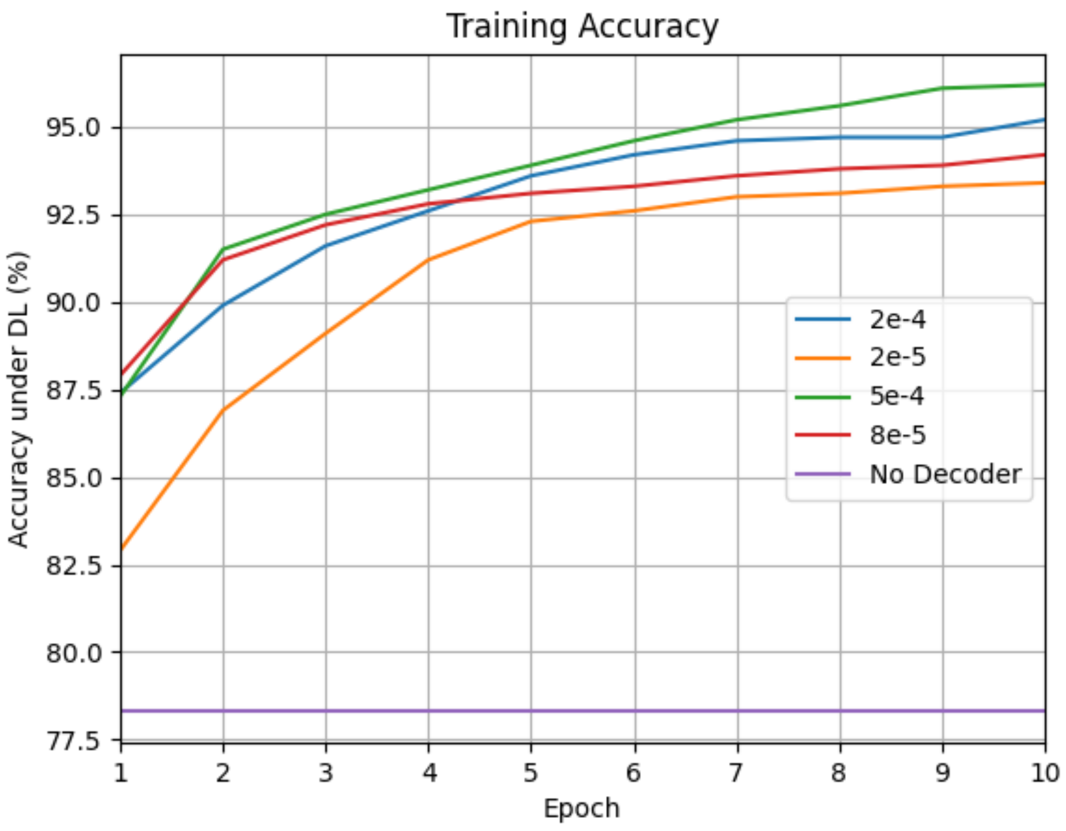


Figure 19 – Training data results under DL

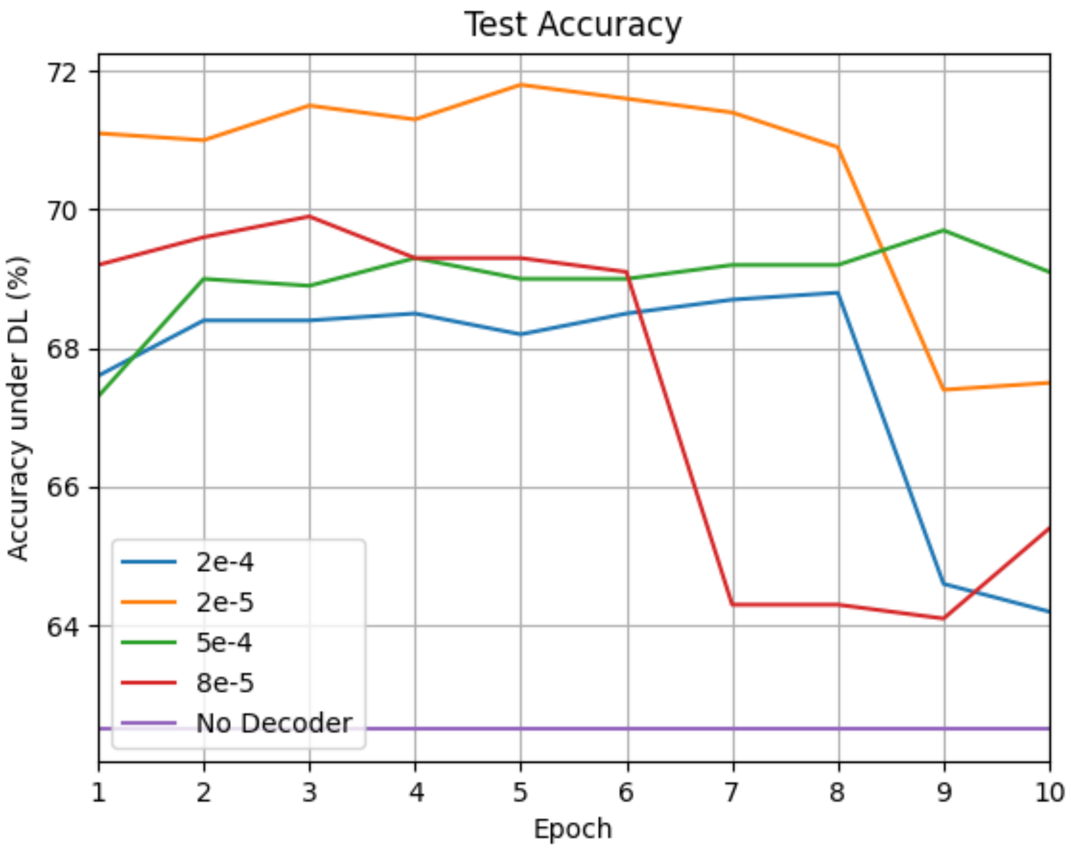


Figure 20 - Test data results under DL

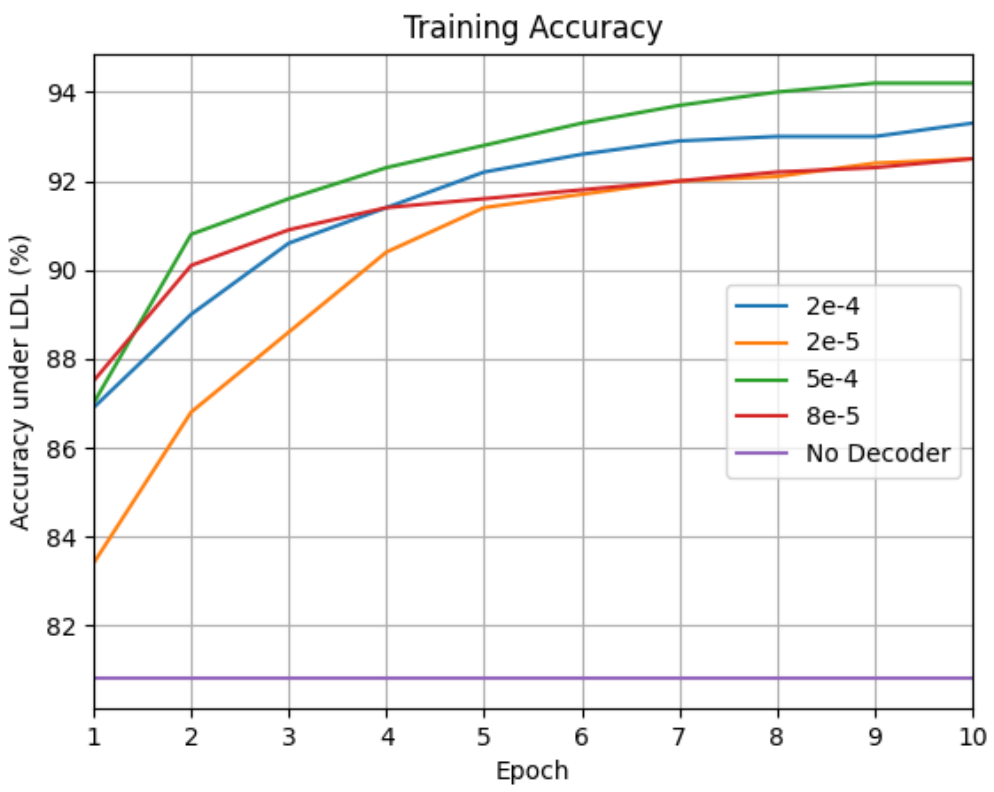


Figure 21 – Training data results under LDL

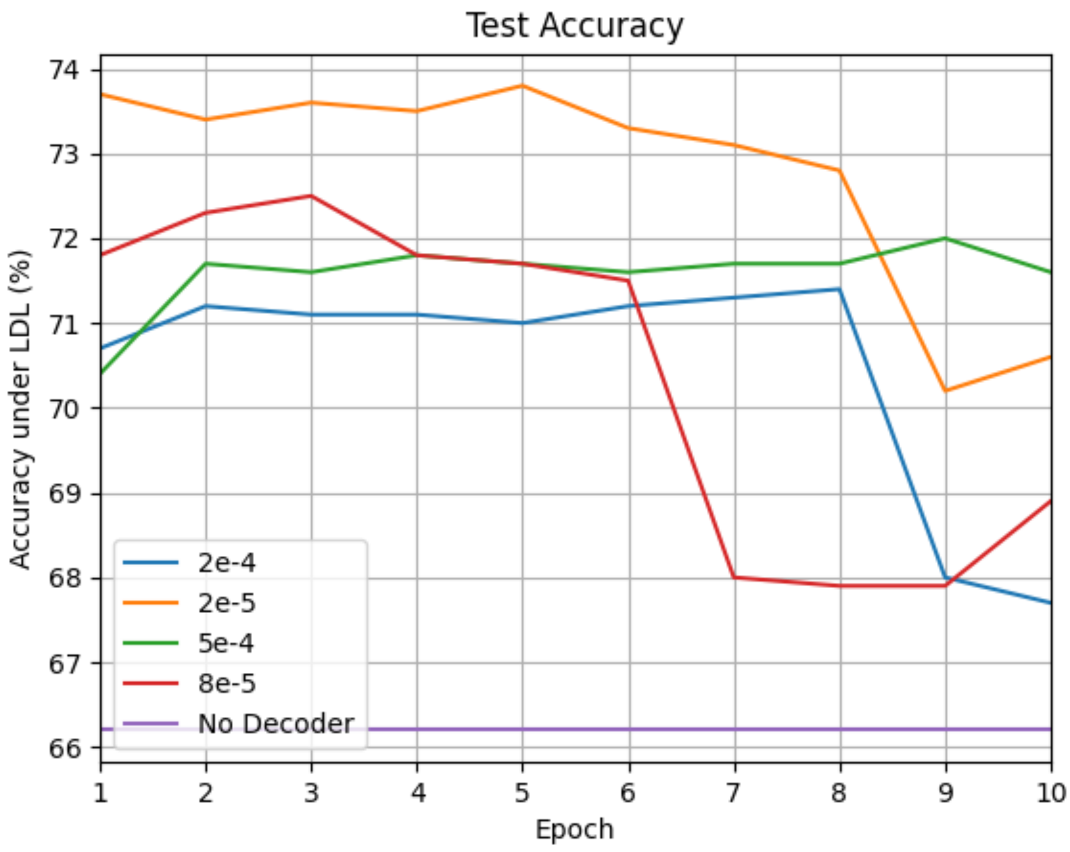


Figure 22 - Test data results under LDL

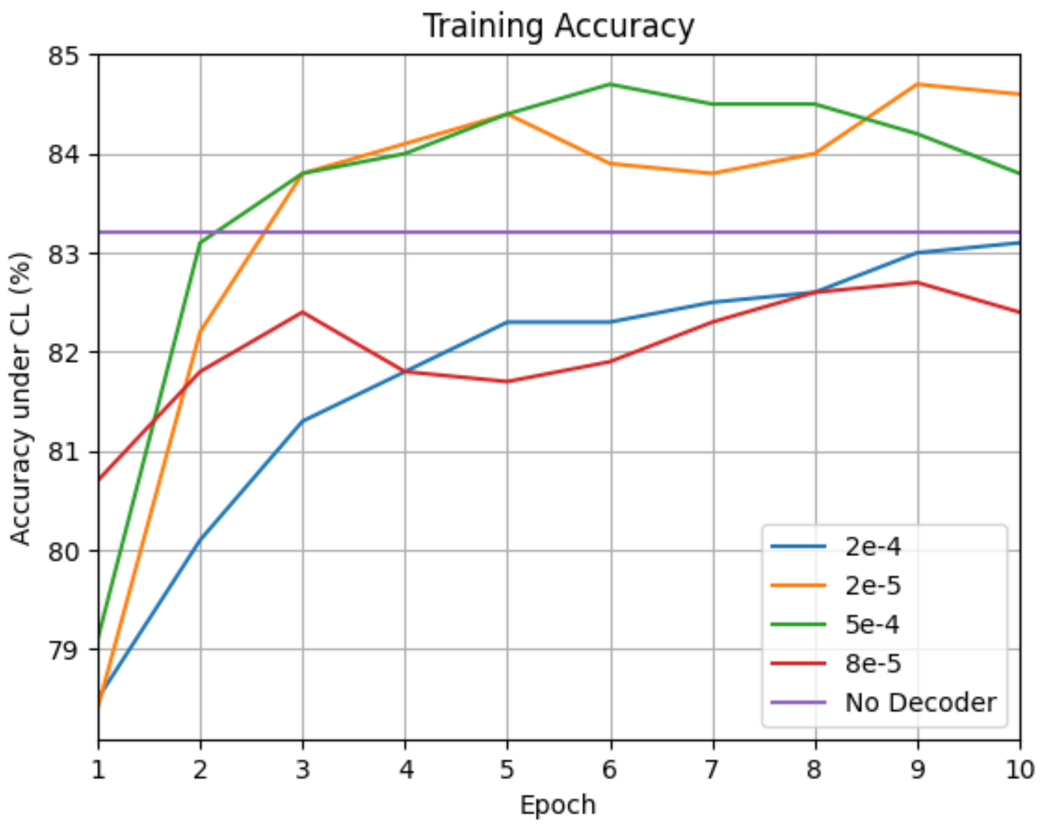


Figure 23 – Training data results under CL

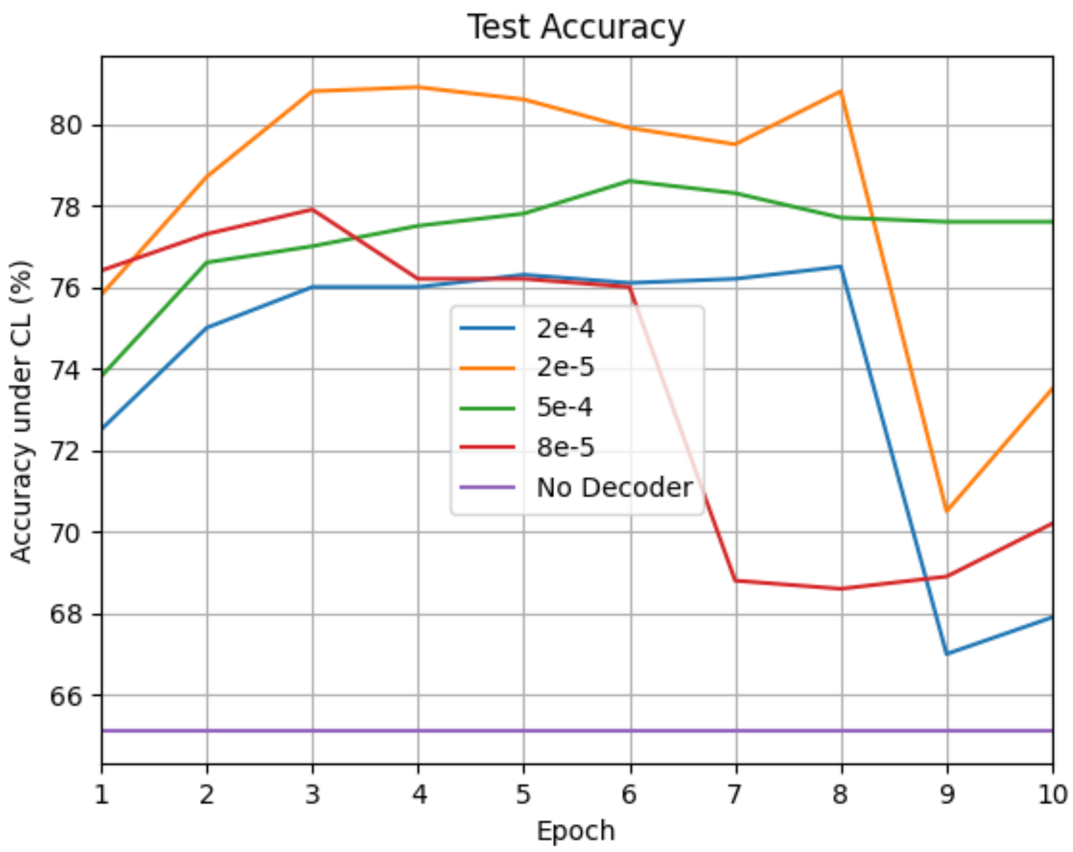


Figure 24 - Test data results under CL

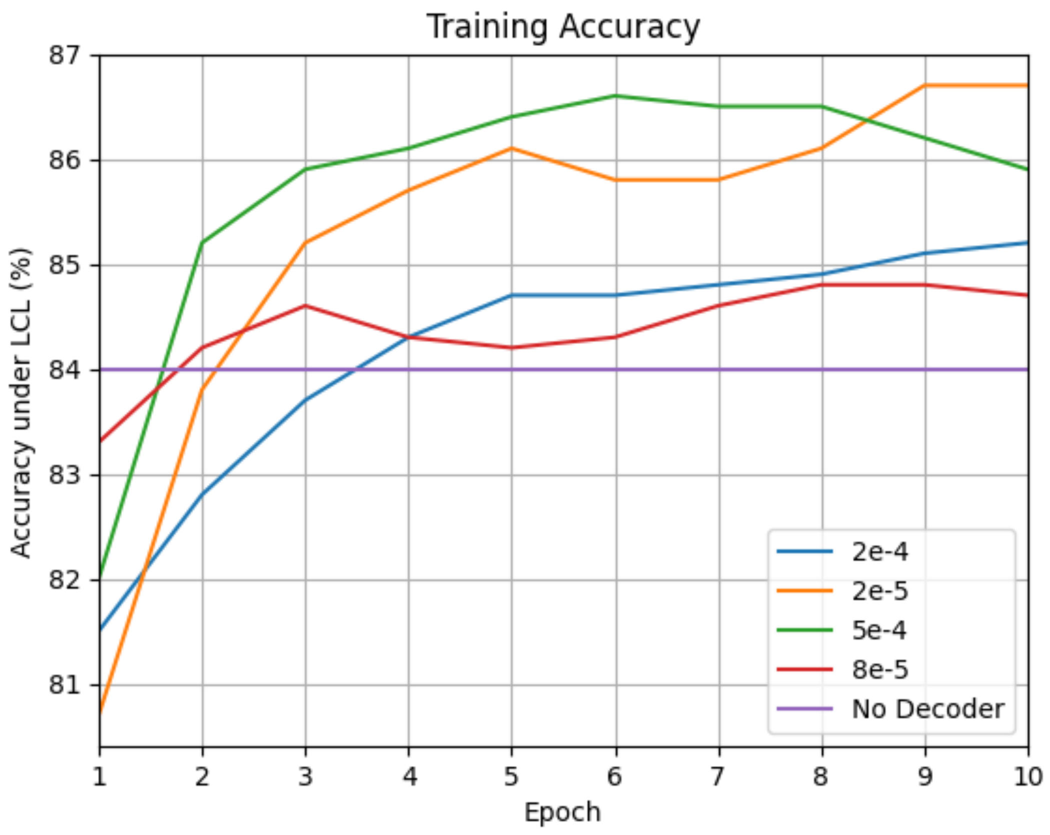


Figure 25 – Training data results under LCL

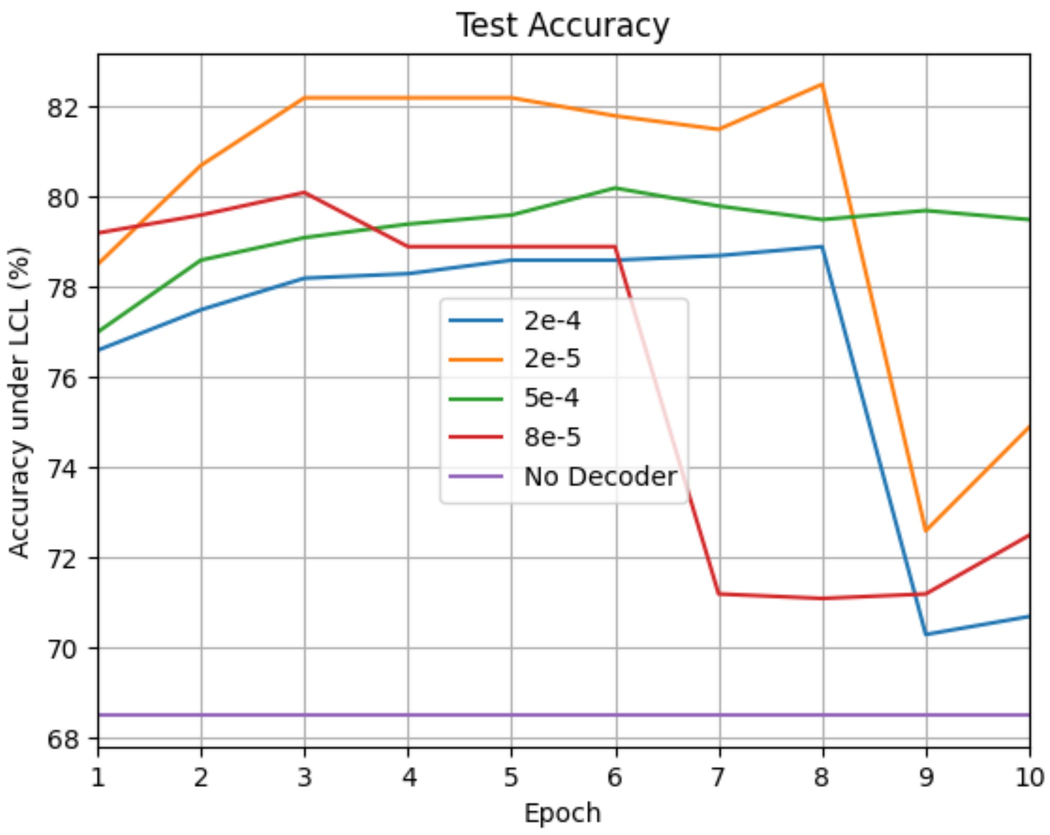
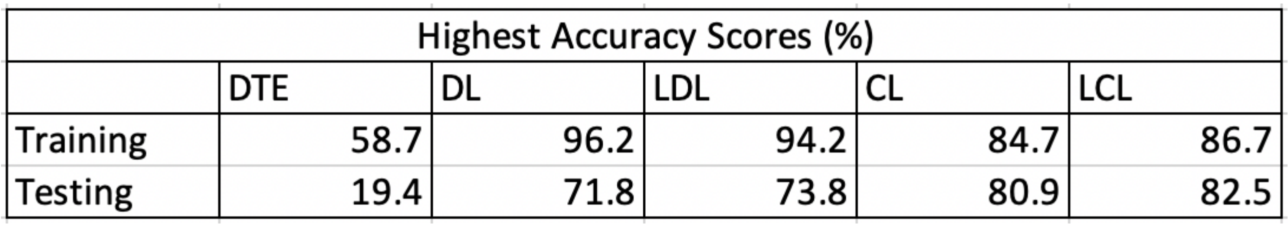


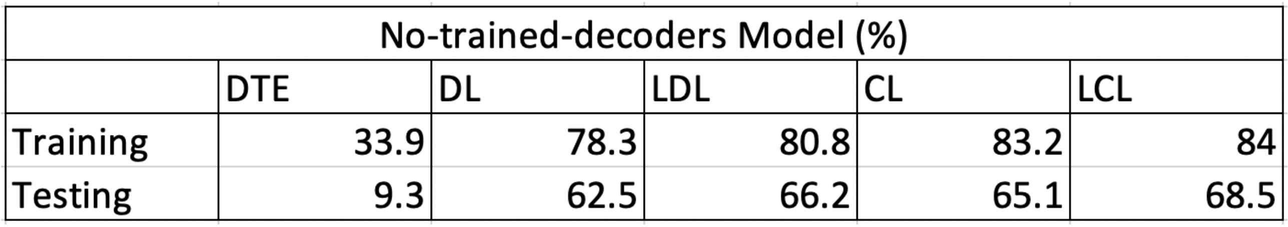
Figure 26 - Test data results under LCL

Figures 27 & 28 below summarise the highest scores for each of the loss functions for the trained models and the scores obtained by the non-trained model:



Highest Accuracy Scores (%)					
	DTE	DL	LDL	CL	LCL
Training	58.7	96.2	94.2	84.7	86.7
Testing	19.4	71.8	73.8	80.9	82.5

Figure 27 - Highest Accuracy Scores per loss function



No-trained-decoders Model (%)					
	DTE	DL	LDL	CL	LCL
Training	33.9	78.3	80.8	83.2	84
Testing	9.3	62.5	66.2	65.1	68.5

Figure 28 - Scores for non-trained model

On a high level a couple of things can be inferred:

- Training losses in all scenarios improved in relation to the first epoch, effectively demonstrating that learning is happening.
- This learning did not carry over to the ability to generalize to unseen examples as it can be seen by falling accuracy over the epochs in the test set.
- Even if so, results over the test set show that all trained models beat the no-trained-decoder model.

Observations and justifications

→ The model with the lowest learning rate initially performed the best in the test set for all loss functions, which implies that large initial changes in decoder values causes the model to have a lower ability to generalize. For further epochs, however, the training set was modelled too specifically, and the test set accuracy fell.

→ The largest learning rate produced the most stable accuracy for the test set, this is most likely because large learning rates do not allow the model to overfit the training data as much.

→ Training accuracy under the CL and the LCL were particularly low in comparison to the no-trained-decoder model. Most likely, this is due to the high magnitude values associated to octaves of the correct notes. That is, octaves have a similar set of harmonics, and often, the same note but in a different octave was predicted. Given that the octave is the lowest possible loss in the consonance loss (besides the note itself), the no-trained-decoders model performed above two of the trained models.

→ The clearest pattern in the results is the improvement in the accuracy for Distance-loss over the training set. This implies that the model was correctly suppressing values that were distant from the notes and highlighting those that were close. Considering the way in which the audio was processed, it is possible that the neighbouring frequencies (of the correct frequency) also had large magnitudes, which in turn increase the probability of the model predicting them. This is made clear by the fact that the DL model under the sigmoid function performed better than the one under the linear function. That is, the sigmoid function associates a lower loss for values smaller than 4.

→ The prediction of True Positives (DTE) improved by nearly 25%, which is equivalent to an increase of 73% over the value of the model with no trained decoders (i.e., 25% increase over a score of 33.9%). However, it still does not yield a competitive result.

→ It should be noted that the most important result is effectively given by the DTE. Nonetheless, the other approaches serve to provide an insightful view of what the learning is doing, and what mistakes are being made.

→ The model demonstrates a superior ability to classify chords it had heard previously as opposed to new sounds, which is to say, if this really mimics cognition, humans without the innate ability to recognize pitches benefit from having heard the sound previously to classify it.

When contextualizing these findings, this model clearly demonstrates the ability to learn and improve upon signal processing techniques alone with very little training. That is the case for both sounds it had been exposed to previously and sounds it hadn't.

Evaluation

Considering the signal processing techniques in the literature (and their respective results), it seems clear that the biggest flaw of the model is not on the learning part, but rather on the feature extraction of the audio recordings, which by themselves performed far lower than the results discussed in the Background section. One possible way to approach this issue is suggested by the results in the LCL and the CL: since octaves are clearly affecting the predictions in a negative way, if the dimensionality of the data representation had been reduced to consolidate the 88 notes into a 12-note vector, the total amplitude of each note's frequency bin would reflect any large magnitude values in the octaves. This has a direct trade-off with the ability to represent voicings and is something the context of use will determine (a possible hyperparameter that can be set through testing and validation). On the other hand, the irregularity of the dataset distribution (discussed in the beginning of this section), also provides a possible cause for these results.

This project aimed to tackle all possible combinations of chords for a given number of notes. However, it seems clear that there are chords played more often than others, especially in pop music. That being the case, filtering the database in such a way as to only consider chords with a probability above a certain threshold has the potential to produce an accurate model for a smaller but more likely set of outcomes, requiring significantly less computational power. Further to this, knowing the possible output labels allows the chord to match the patterns as it was done by Osmalskyj et al. (2012). This is beyond the scope of the project but is nonetheless a direct consequence of the model and conclusions obtained.

The original objectives set a new way to explore a complex problem. They did not, however, outline this project as an attempt to beat the state-of-the-art benchmarks set by ByteDance and Google. What they did do, is allow an interdisciplinary approach for applying engineering competencies and to gain new skills in the process. In effect, the project allowed the exploration of the learning capabilities of biologically plausible model of cognition more than it did solve the problem of harmony recognition. Indeed, the model retains the status of proof-of-concept for harmony recognition, as it is scalable and has a proven capacity to learn. Above all, it approaches the problem in a new way, and perhaps provide some insight into the biological processes in the brain when facing this task.

CONCLUSION

The project presented a novel way of approaching the problem of harmony recognition by endeavoring to mimic the brain's computational processes whilst adapting techniques from signal processing and machine learning. By introducing the NEF and the benefits of SNNs, the report attempted to convince the reader of the importance in modelling according to the brain, and the possible ways of doing so effectively. Moreover, it justified its approach by carrying out an analysis over different possible parameters (learning rates), and by devising new ways to reflect the results and the effects of changing such parameters (loss functions).

The results obtained demonstrate that the utilization of an online learning rule is an effective way of training a spiking model and suggests that it is currently feasible to utilise such Neural Engineering Framework to devise algorithms that might be reflective of the brain's computational processes. Beyond this, the model can provide predictions in real-time, much like it is expected from a human.

By providing a method for scalability and being faithful to the concepts discussed in the Background section, the project has achieved two of the objectives that were initially set out. Arguably, the lack of ability to generalise reduces the model's usefulness and provides a few possibilities to tweak and improve on the performance (some of which have been discussed previously).

For further development, a framework called Semantic Point Architecture (SPA) in Nengo is being considered. It allows representations that carry semantic content and can capture the relations between different concepts in a vector space. This is a possible way to expand on the work done so far, and explore on the relations between musical chords, namely, functional harmony. Beyond this, as a way to facilitate the use of such model by a non-technologically-versed public, the development of a Graphical User Interface is being considered.

PROFESSIONALISM AND RESPONSIBILITY

Artificial Intelligence and mimicking the Human brain

A core issue that is extremely relevant nowadays is determining whether Artificial Intelligence is being used as a substitute for human intelligence or as a way to enhance it. Efforts have been made in this project to create a model that could *aid* anyone who intends to tackle harmony transcription and recognition. It is critically important to be aware and clear about the purposes behind any project regarding AI, as it involves leveraging techniques can impact other's lives. In a wider sense, we should reflect on how responsibility is allocated for the results of today's artificial intelligence. That is, we can *measure* results and *see* their efficacy in carrying out a diverse range of tasks, but the extent to which we fully understand how each weight parameter affects the end result is not clear (especially in the large, deep learning models). Ethical, social, and environmental concerns are, therefore, part of any technological development in this field, and should permeate any research to support sustainable development.
That being the case, the scope of this project was limited to a single complex task which even professionals have difficulty in tackling. The framework utilized is biologically plausible and leverages property of neurons, but to the extent that is known today, does not necessarily act in the same manner as the human brain does. Overall, the goal of this project is providing a tool that can be adapted for the needs of the user (i.e., higher range of notes, different datasets) whilst not providing a definitive solution for the problem. In line with the Open Source Initiative, the code for both the model developed and the framework are freely available online for public knowledge (link in the appendix).

Social, environmental, and ethical impact

On a wider scope, this is a global responsibility, where much like in the 19th century when a few countries managed to lead the industrialization and subsequently develop faster than others, the risk now exists in a less tangible, cognitive atmosphere.

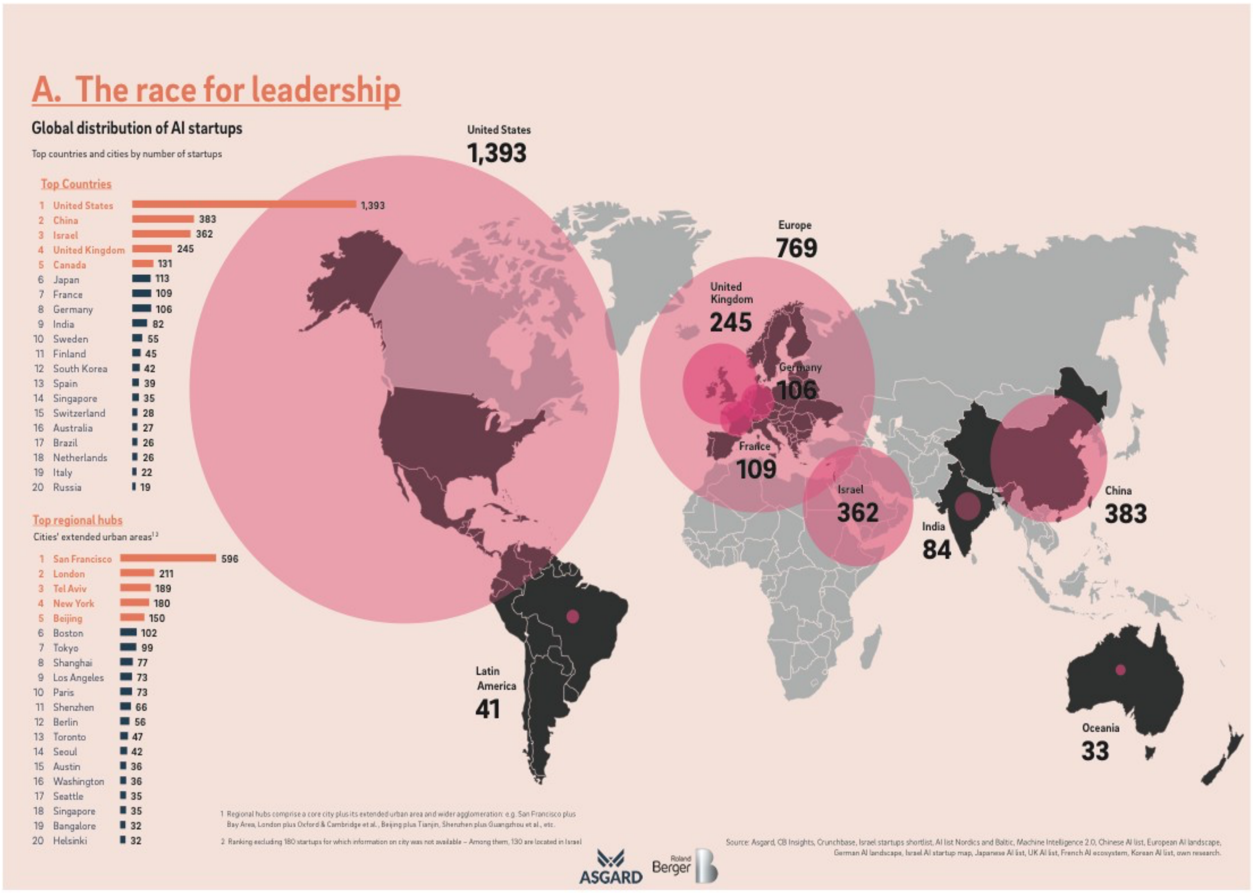

A. The race for leadership
Global distribution of AI startups
Top countries and cities by number of startups
Top Countries
1 United States 1,393
2 China 383
3 Israel 362
4 United Kingdom 245
5 Canada 131
6 Japan 113
7 France 109
8 Germany 106
9 India 82
10 Sweden 55
11 Finland 45
12 South Korea 42
13 Spain 39
14 Singapore 35
15 Switzerland 28
16 Australia 27
17 Brazil 26
18 Netherlands 26
19 Italy 22
20 Russia 19
Top regional hubs
Cities' extended urban areas1 2
1 San Francisco 596
2 London 211
3 Tel Aviv 189
4 New York 180
5 Beijing 150
6 Boston 102
7 Tokyo 99
8 Shanghai 77
9 Los Angeles 73
10 Paris 73
11 Shenzhen 66
12 Berlin 56
13 Toronto 47
14 Seoul 42
15 Austin 36
16 Washington 36
17 Seattle 35
18 Singapore 35
19 Bangalore 32
20 Helsinki 32
United States 1,393
Europe 769
United Kingdom 245
Germany 106
France 109
Israel 362
India 84
China 383
Latin America 41
Oceania 33
1 Regional hubs comprise a core city plus its extended urban area and wider agglomeration: e.g. San Francisco plus Bay Area, London plus Oxford & Cambridge et al., Beijing plus Tianjin, Shenzhen plus Guangzhou et al., etc.
2 Ranking excluding 180 startups for which information on city was not available – Among them, 130 are located in Israel
Source: Asgard, CB Insights, Crunchbase, Israel startups shortlist, AI list Nordics and Baltic, Machine Intelligence 2.0, Chinese AI list, European AI landscape, German AI landscape, Israel AI startup map, Japanese AI list, UK AI list, French AI ecosystem, Korean AI list, own research.
ASGARD
Roland Berger

Figure 29 - The race for leadership[7]

Figure 29 shows the distribution of AI startups as of 2018. The disparity shown emphasizes the importance of the point made, as the concentration of technological power in a small number of countries begs a discussion for the ways in which ethical work can be carried out and how regulations can be implemented to aid sustainable progress.

In the UK, an AI Council was devised to propose a roadmap for the future, with suggestions for possible frameworks which allow for scrutiny of the processes and data used, whilst supporting the growth of AI initiatives. Among the ideas proposed, there is a focus in improving AI literacy, governance, and support for R&D, which if implemented together can result in an increased incentive for attracting talent, shaping a global strategy, and facilitating implementation.

On average, the computational resources need to keep up with the best AI models doubles every 3.4 months[8]. The state-of-art models improve on an astonishing rate, and as datasets continue to grow, the need to re-train and develop is constant. Along with this comes the enormous amount of energy utilized and subsequent carbon footprint. This issue is very much in line with the aim of this report: mimicking the world's most efficient computer. Efforts focused in improving computations (intelligence) per Kilowatt Hour per

[7] https://asgard.vc/global-ai/
[8] https://www.forbes.com/sites/robtoews/2020/06/17/deep-learnings-climate-change-problem/?sh=4b86a0356b43

square millimeter are essential, both on the forefront of sustainability, and for accessibility (E.g., where vast amounts of energy are not available).

Considering the matter of Intellectual Property, the database and the software were used for educational purposes only, which limits any liability, but nonetheless have been properly referenced as being owned by a 3rd party. All images and videos utilized in the project report and demonstration have either been sourced from a copyright-free source or referenced in a clear manner.

BIBLIOGRAPHY

Barbancho et al. (2010). PIC Detector for Piano Chords. EURASIP Journal on Advances in Signal Processing. 10.1155/2010/179367.

Benetos, E., Dixon, S., Duan, Z., & Ewert, S. (2018). Automatic music transcription: An overview. IEEE Signal Processing Magazine, 36(1), 20-30. https://doi.org/10.1109/MSP.2018.2869928

Chen, Tsung-Ping and Li Su. (2018) "Functional Harmony Recognition of Symbolic Music Data with Multi-task Recurrent Neural Networks."

Eliasmith, C., & Anderson, C. H. (2003). Neural engineering: Computation, representation and dynamics in neurobiological systems. Cambridge, MA: MIT Press.

Engineering ToolBox, (2003). Notes, Octaves and Frequencies. [online] Available at: https://www.engineeringtoolbox.com/note-frequencies-d_520.html

Hatfield, Michael W., (2019). Professionally Responsible Artificial Intelligence. Arizona State Law Journal, Vol. 51, No. 3, Pp. 1057-1122, 2019, Available at SSRN: https://ssrn.com/abstract=3516287

Heng-Tze Cheng, Yi-Hsuan Yang, Yu-Ching Lin, I-Bin Liao and H. H. Chen, (2008). "Automatic chord recognition for music classification and retrieval," 2008 IEEE International Conference on Multimedia and Expo, pp. 1505-1508, doi: 10.1109/ICME.2008.4607732.

Kong et al. (2022). GiantMIDI-Piano: A large-scale MIDI Dataset for Classical Piano Music, Transactions of the International Society for Music Information Retrieval, https://doi.org/10.48550/arXiv.2010.07061.

Natalia Caporale & Yang Dan, (2008). Spike Timing–Dependent Plasticity: A Hebbian Learning Rule. Annual Review of Neuroscience 2008 31:1, 25-46

Osvaldo Simeone (2017). A Brief Introduction to Machine Learning for Engineers. Available at: https://nms.kcl.ac.uk/osvaldo.simeone/notesMLSimeone.pdf

R. Plomp & W. J. M. Levelt, (1965). Tonal Consonance and Critical Bandwidth. The Journal of the Acoustical Society of America 38, 548; https://doi.org/10.1121/1.1909741

Rajendran, B. (2021). Brain-Inspired Computing & Hardware Design.

Saputra, F., Namyu, U. G., Vincent, ., Suhartono, D. & Gema, A. P. (2021). Automatic Piano Sheet Music. Transcription with Machine Learning. Journal of Computer Science, 17(3), 178-187. https://doi.org/10.3844/jcssp.2021.178.187

Simon Dixon, (2003) "Extraction of Musical Performance Parameters from Audio Data"

Stewart, T. C. (2012). (tech.). A Technical Overview of the Neural Engineering Framework.

T. Tolonen and M. Karjalainen, (2000). "A computationally efficient multipitch analysis model," IEEE Transactions on Speech and Audio Processing, vol. 8, no. 6, pp. 708–716

Takuya Fujishima (1999). Realtime Chord Recognition of Musical Sound: A System Using Common Lisp Music, ICMC Proceedings.

APPENDIX

Resources utilised in the model, including the results: https://github.com/gusauriemo/Final-Year-Project

Nengo's website: https://www.nengo.ai/

Error equivalence between Distance-loss and Consonance-loss:

Number of semitones for equivalent error	
DL	CL
0	0
1	12
2	10
3	7
4	5
5	4
6	8
7	2
8	6
9	3
10	11
11	9
12	1