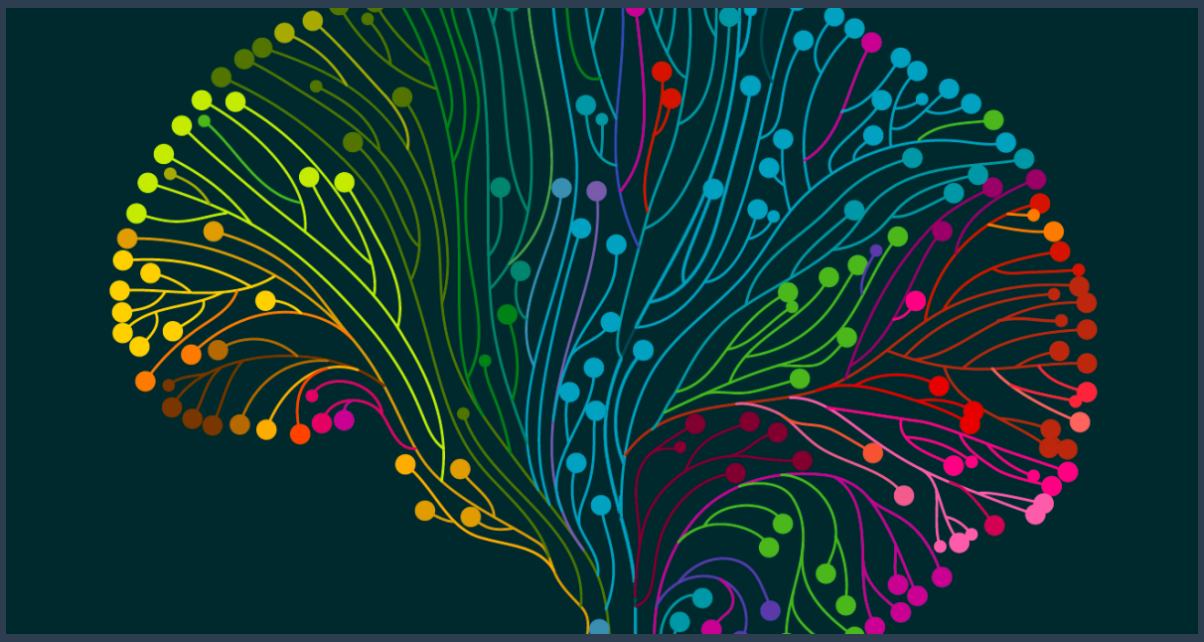
First Report

Sound Detection and Classification
using Spiking Neural Networks



COURREGE Téo
GANDEEL Lo'aï

Date: February 26, 2024

Contents

List of Figures

1 Introduction

Our goal is to perform a thorough investigation and subsequent implementation of spiking neural networks (SNNs). SNNs, which are inspired by the neural signaling patterns of the human brain, show promising potential in various applications, especially in real-time processing and pattern recognition.

A spiking neural network (or SNN) is a type of artificial neural network that mimics the functionality of biological neural networks. Unlike traditional artificial neural networks (ANNs), SNNs incorporate the concept of time into their operating model. The neurons in SNNs generate spikes of activity and communicate through these spikes (like brain neurons stimulating each other with electrical impulses), allowing them to process information in a more complex and potentially more efficient manner.

This type of neural network theoretically consumes less energy than some ANNs and is potentially more efficient in terms of processing power (in practice, it will be necessary to create an adapted hardware architecture to take advantage of this potential). Therefore, it seems that the Therefore, the use of this type of neural network could be a viable solution to a classification processing problem (especially in real time).

Our project will specifically address the challenges of applying SNNs to audio classification in the field of spiking neural network research. This report provides an overview of our initial progress in this area. Our project specifically addresses the problem of audio classification within the broader context of SNNs.

Before delving into the details, we outline key aspects including preprocessing, data manipulation/augmentation, initial model implementations, and a look at preliminary results.

Using audio data primarily from the Google AudioSet database, our work involves preprocessing, which includes converting signals to image representations, extracting features, and considering encoding schemes suitable for SNNs. Challenges related to data quality, context, and labeling complexity prompted the exploration of data augmentation strategies to improve model robustness.

In the following sections, we present the organizational structure of our project, the technical environment used, a detailed description of the work performed, the challenges faced, and the solutions implemented, concluding with a reflection on our achievements and future perspectives.

2 The project

2.1 Description of the project - Spiking Neural Networks

Inspired by the neural signaling patterns of the human brain, SNNs introduce a temporal element into artificial neural networks. This temporal characteristic positions SNNs as promising candidates for real-time processing and pattern recognition tasks.

A spiking neural network is a variant of artificial neural networks designed to more closely mimic biological neural networks. Unlike traditional artificial neural networks (ANNs), which work with continuously changing time values, SNNs work with discrete events that occur at defined times. They take a set of spike values as input and produce a set of spike values as output.

The spiking behavior of a neuron in an SNN is modeled by a membrane potential equation. For example, in a leaky integrate-and-fire (LIF) neuron model, the membrane potential equation is defined by a set of parameters including a time constant (τ), a resting potential (u_{r1}), a reset potential (u_{r2}), synaptic weights (w_j), and a firing threshold (u_th). The output spike (s) is determined based on the membrane potential (u) and various conditions. This discrete, event-based approach distinguishes SNNs from other types of neural networks.[1]

$$\begin{cases} \tau \frac{d\,u(t)}{dt} = -[u(t) - u_{r_1}] + \sum_j w_j \sum_{t_j^k \in S_i^{Tw}} K(t - t_j^k) \\ \begin{cases} s(t) = 1 \quad u(t) = u_{r_2} \text{ if } u(t) \geq u_{th} \\ s(t) = 0 \quad \text{otherwise} \end{cases} \end{cases} \tag{1}$$

This equation was first written as :

$$\tau \frac{d\,u(t)}{dt} = -[u(t) - u_r] + RI \tag{2}$$

From a mathematical point of view, equation (2) is a linear differential equation. Alternatively, an electrical engineer might recognize it as the equation of a leaky integrator or a RC circuit with a parallel resistor (R) and capacitor (C). In neuroscience, this equation is called the passive membrane equation. [2]

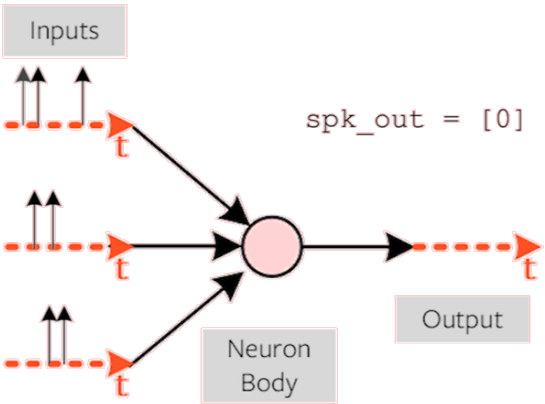


Figure 1: SNN input

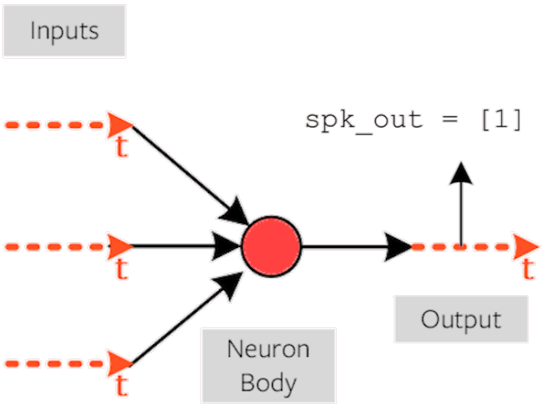


Figure 2: SNN output

2.1.1 Audio Classification Task

Audio classification involves assigning a label to an audio clip based on its content. The audio classification task can be further divided into two subtasks: sound event detection (SED) and sound event classification (SEC). SED involves detecting the onset and offset times of sound events in an audio clip, while SEC involves assigning a label to each detected sound event.

In the context of audio classification, training an SNN involves working with image representations of audio data and encoding schemes suitable for SNNs.

2.1.2 Project Objectives

The primary goal of our project is to exploit the temporal processing capabilities of SNNs for audio classification tasks. Specifically, we want to develop models capable of classifying (and possibly detecting) sound events from audio data.

Furthermore, knowing that SNNs consume less power than traditional Artificial Neural Networks (ANNs), but have lower overall accuracy, we want to perform a performance comparison of SNNs with ANNs.

Achieving these goals would allow us to determine the potential of SNNs for audio classification tasks and identify the advantages and disadvantages of SNNs compared to other neural networks. It is also a great way for us to learn more about SNNs and audio classification.

3 Organizing the project

3.1 Main tasks

During the last full time period, we worked on:

- **A preprocessing pipeline** that allows us to download, format, and segment the audio portion of the YouTube videos that make up the Google Audioset audio files into images. To be efficient, the pipeline needed to be parallelized.
- **Finding some correct data augmentation techniques** that can be used to improve the performance of the SNNs.
- **Finding a way to encode the audio data into spikes** that can be used as input for the SNNs.
- **Implementing the SNNs** that will be used for the audio classification task.

  Training the SNNs on the audio data.
- **Implementing the ANNs**, which we would compare with the SNNs.
- **Comparing the performance of the SNNs and the ANNs** on the audio classification task.

3.2 Planning and team organization

3.2.1 Previous work - full time period

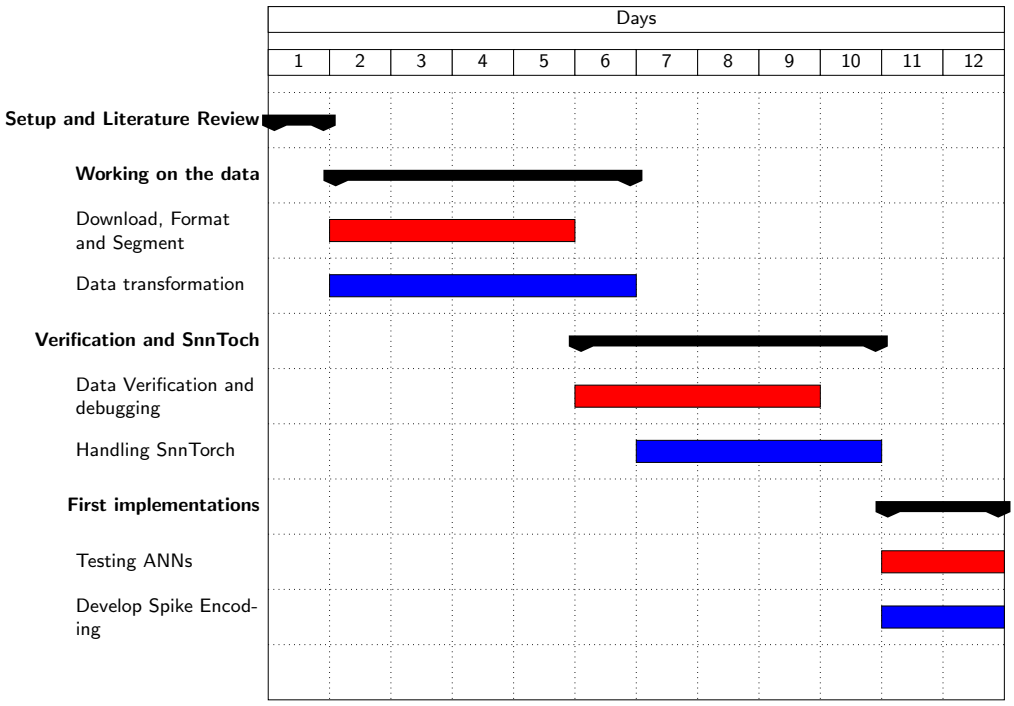


Figure 3: Planning of the full time period

- Téo : blue
- Loaï : red

So far, we have been able to encode and decode the audio data into spikes.

Since the first tasks we had to perform would not be connected until they were finished, we decided to work on them in parallel. This allowed us to be more efficient and, most importantly, to save time.

In the initial plan, there was no task related to data verification or debugging any part of the code. Each part of the preprocessing pipeline takes a lot of time to implement and test. We had to spend some time debugging the code and verifying the data.

4 Work accomplished

4.1 Data

Notes: In the section below, we decided not to include some code snippets because they were not very relevant to the report.

4.1.1 The AudioSet

In order to use the Google Audioset dataset, we needed to download the audio files and their associated labels. This first resulted in a file containing the links to the audio files and a file containing the label(s) of the audio files. Their structure is as follows:

```
url_extension, start_seconds, end_seconds, positive_labels
-0SdAVK79lg, 30.000, 40.000, "/m/0155w,/m/01lyv,/m/0342h,/m/042v_gx,/m/04rlf,/m/04szw,/m/07s0s5r,/m/0fx80y,/m/0gg8l"
-0Vl4HyWRk8, 410.000, 420.000, "/m/085jw,/m/0l14l2"
-5xOcMJpTUk, 70.000, 80.000, "/m/018vs,/m/0342h,/m/042v_gx,/m/04rlf,/m/04szw,/m/09x0r,/m/0fx80y"
-6QGvxvaTkI, 280.000, 290.000, "/m/01p970,/m/026t6,/m/0l14md"
-BIMKnb3tlo, 410.000, 420.000, "/m/01p970,/m/026t6,/m/0l14md"
-CUp_Tmg2Y0, 30.000, 40.000, "/m/01qbl,/m/026t6,/m/02hnl,/m/03qtq,/m/03t3fj,/m/0bm02,/m/0l14md"
-DeAdhYKbGE, 290.000, 300.000, "/m/0192l,/m/085jw"
-Dj2PfPmynQ, 30.000, 40.000, "/m/04rlf,/m/0l14jd,/t/dd00003,/t/dd00004"
-FEPOSP7ay0, 260.000, 270.000, "/m/0395lw"
```

Figure 4: Structure and small content of the csv file containing the links to the audio files

Filtering the data by labels

The first column contains the links to the YouTube video, then the start and end time of the audio segment in the video, and finally the labels of the audio segment (there can be multiple labels for one audio segment).

Since we want to train our first model on small scenarios with only 3 classes Animals, Sound of things, and Music, we had to filter the data so that in all the rows that make up the csv file, there is at least one of the 3 classes we wanted to keep, but no repetition between the 3 classes.

Basically, we needed to iterate over the initial csv file to create 3 csv files, each containing only videos with one of the 3 classes we wanted to keep (see `reorder_to_categories.py`).

Also, there might be some repetitions between the labels associated with each audio segment, so we needed to remove those duplicates (see `reorder_btw_categories.py`).

(see our GitHub repository).

Downloading the audio files

Once we had the csv files containing the links to the audio files, we needed to download them. To do this, we started using this Github repository as a base for our work. Since there were some problems using the `youtube-dl` library (that was changes to the `yt-dlp`), we had to modify the code to allow parallelization of the process (crucial for the efficiency of the pipeline if we don't want to wait for days to download the audio files).

We ended up with a code similar to the first one, but with some compatibility, performance and parallelization improvements (see `main_dfs.py`).

Checking the data

Once we had downloaded the audio files, we had to check that all the registered ones were not missing some information, corrupted or of duration less than 10 seconds (see `valid_with_labels`). This part ended-up being a source of problems as it consisted in a test and debug part.

4.2 Study of SNNs

4.2.1 Neurons model

Some frameworks have been created to modelize the dynamics of a neuron. Three of the models that have been created are the Leaky Integrate-and-Fire (LIF) Neuron Model (the most computationnally efficient), the Hodgkin-Huxley Neuron Model (the most biophysically pertinent), and the Izhikevich Neuron Model (a compromise between efficiency and realism).

1. Leaky Integrate-and-Fire (LIF) Neuron:

   The LIF Neuron Model[2] is a basic conceptualization of neurons as leaky integrators of incoming signals. The model is governed by a single differential equation which dictates the membrane potential evolution over time. It is the most used model in spiking neural networks.

   Equation:

$$\tau_m \frac{du}{dt} = -[u(t) - u_{\text{rest}}] + RI(t)$$

   τ_m: Membrane constant
   u: Membrane potential
   u_{rest}: Resting membrane potential
   R: Membrane resistance
   $I(t)$: Current

2. Hodgkin-Huxley (HH) Neuron:

   The Hodgkin-Huxley Neuron model is the most biophysically realist, but is more complex. This model delves into the intricate details of neuronal behavior, incorporating ion channels and their conductances to mimic biological processes. This Model is built upon on a set of nonlinear differential equations.

   Equations:

$$\begin{aligned}
C_m \frac{dV}{dt} &= -I_{\text{Na}} - I_{\text{K}} - I_{\text{L}} + I_{\text{ext}} \\
I_{\text{Na}} &= g_{\text{Na}} m^3 h (V - E_{\text{Na}}) \\
I_{\text{K}} &= g_{\text{K}} n^4 (V - E_{\text{K}}) \\
I_{\text{L}} &= g_{\text{L}} (V - E_{\text{L}}) \\
\frac{dm}{dt} &= \alpha_m (1 - m) - \beta_m m \\
\frac{dh}{dt} &= \alpha_h (1 - h) - \beta_h h \\
\frac{dn}{dt} &= \alpha_n (1 - n) - \beta_n n
\end{aligned}$$

   C_m**:** Membrane capacitance

   V**:** Membrane potenti

   I_***:** Sodium, Potassium, leakage, and external current

   g_***:** Maximum sodium, potassium, and leakage conductance

   E_***:** Nernst potential for sodium, potassium, and leakage (the equilibrium potential)

   m, h, n**:** Gating variables that represent the fraction of channels in the open state for sodium, potassium, and leakage channels, respectively

   $\alpha_m, \beta_m, \alpha_h, \beta_h, \alpha_n, \beta_n$**:** Voltage-dependent rate constants governing the kinetics of the opening and closing of ion channels

3. Izhikevich Neuron:

   The Izhikevich neuron model [**?**] is a two-dimensional model designed to be as biologically plausible as the Hodgkin–Huxley model and as computationally efficient as the integrate-and-fire model. Depending on the parameters, the model reproduces spiking and bursting behavior of known types of neurons.

   Equations:

$$\frac{dv}{dt} = 0.04v^2 + 5v + 140 - u + I_{\text{syn}}$$
$$\frac{du}{dt} = a(bv - u)$$

   v: Membrane potential

   u: Membrane recovery variable

   a, ba, b: Parameters determining the neuron's behavior

   I_{syn}: Synaptic current

   Although it is not the one which represents reality the most, LIF model is the best compromise if we take into account computation complexity.

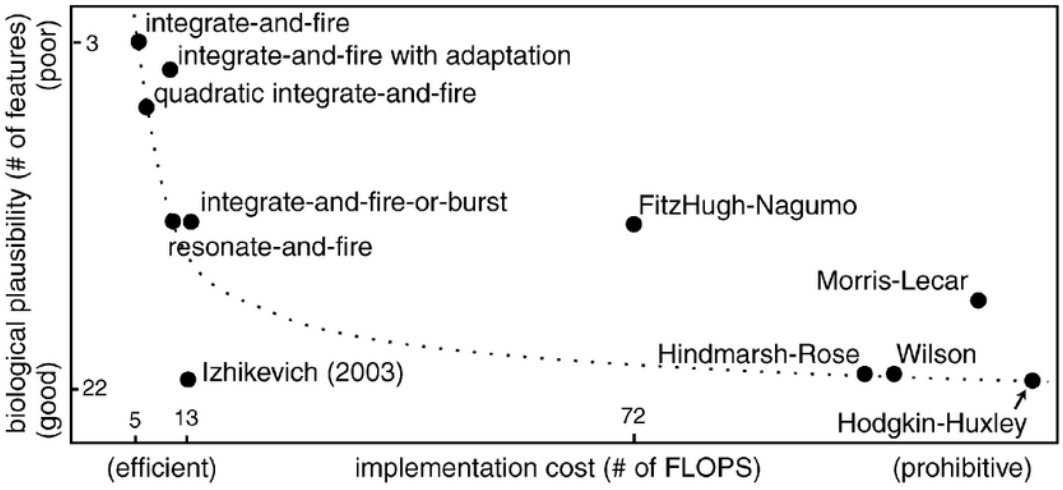


Figure 5: Comparison of different neuron model's complexity and biological similarity

4.2.2 Spike encoding

Encoding in SNNs refers to how information is represented and conveyed by the spikes of the SNN. In our study, we focus primarily on two of them:

- Rate encoding:

  In rate encoding, information is represented by the firing rate of neurons. The more spikes a neuron produces in a given time window, the higher the firing rate and the stronger the encoded information.

- Latency coding:

  Temporal encoding focuses on the precise timing of spikes. The timing information of individual spikes is critical to the representation of the input data.

The exact timing of spikes can convey additional information, and the temporal patterns of spikes are used to encode features or events in the input signal.

4.3 Spectrograms, Mel spectrograms, MFCC

4.3.1 Spectrograms

Spectrograms are essential graphic tools in audio analysis. They offer a visual representation of the frequency spectrum of a sound signal as a function of time, providing detailed information on the frequency composition and temporal dynamics of an audio signal. This report explores the use of spectrograms in various contexts and highlights their importance in the analysis and understanding of audio signals.

A spectrogram is generated by applying a short-time Fourier transform (STFT) to an audio signal. This technique divides the signal into small time windows, then applies a Fourier transform to each window to obtain the frequency distribution at that particular moment. The results of these transformations are then represented graphically, using colors to indicate the intensity of frequencies at different periods.

$$X_k = \sum_{n=0}^{N-1} x_n e^{-i 2\pi k n / N} \tag{3}$$

Figure 6: Discrete Fourier Transform

Spectrograms enable in-depth analysis of the temporal characteristics of audio signals. Events such as transients, attacks and decays can be clearly identified, which is essential for understanding dynamic variations in music, speech and other forms of sound.

By examining the color and intensity of areas in a spectrogram, it's easy to identify the dominant frequencies present in an audio signal. This is particularly useful for detecting anomalies, characterizing musical instruments and separating sound sources.

Spectrograms are widely used in fields such as professional audio, music and linguistics. In particular, we are interested in Mel spectrograms, as well as MFCCs as input data for our neural networks.

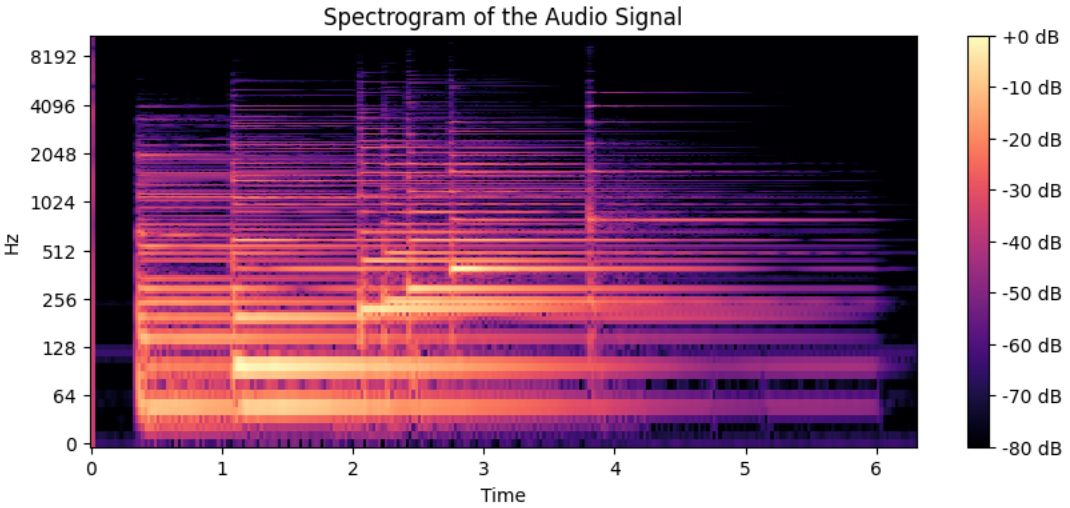


Figure 7: Spectrogram of a slowly ascending piano arpeggio, we can notice the new notes with higher frequencies appearing over time, where the image is brighter

4.3.2 Mel Spectrograms

While spectrograms offer a detailed view of the frequency spectrum of an audio signal, a significant evolution in audio analysis occurred with the introduction of Mel spectrograms.

These represent an adaptation of traditional spectrograms using a frequency scale based on the Mel scale, which is more in line with human auditory perception. This transition to Mel Spectrograms has broadened the possibilities of analysis, offering a more realistic representation of the auditory characteristics perceived by the human ear.

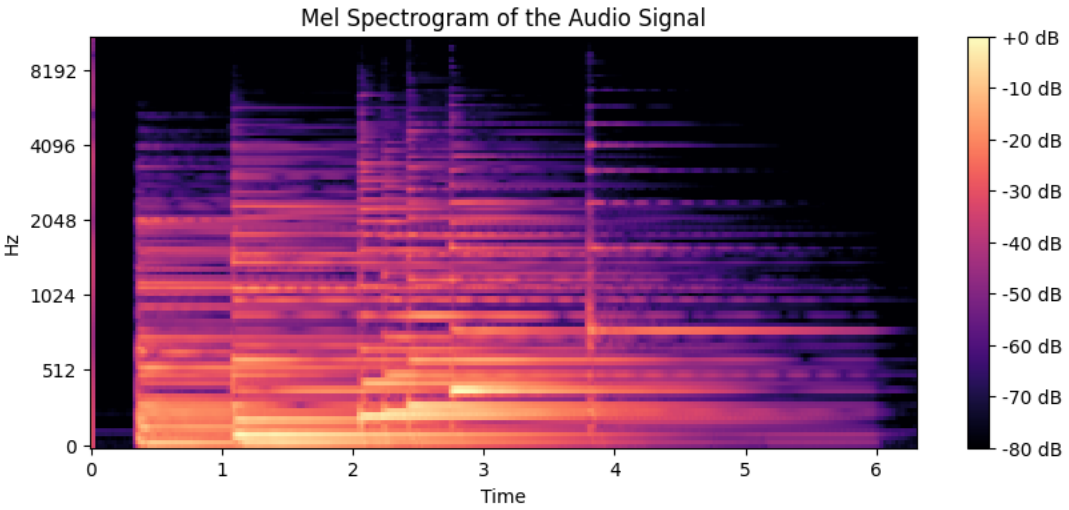


Figure 8: Mel Spectrogram

One of the main differences is that the bands corresponding to a note of the arpeggio are closer than in the spectrogram, which represents better the human auditory system

4.3.3 Mel-frequency cepstral coefficients (MFCC)

MFCCs are derived directly from Mel spectrograms and are computed by applying a discrete cosine transform to the log power of Mel spectrograms. This approach captures information specific to human auditory characteristics while reducing data redundancy. MFCCs thus encapsulate frequency variations over time in a compact way, producing a set of cepstral coefficients that are widely used for automatic speech recognition and other audio signal processing tasks.

Two of the main advantages of MFCCs are compactness and information discrimination. MFCCs condense information while preserving the signal's distinctive characteristics. Compact representation facilitates storage, transmission, and processing of large amounts of data. By focusing on perceptual features rather than raw frequency, MFCCs are less sensitive to pitch variations, improving the robustness of sound recognition. The calculation of MFCCs involves several steps, including Mel scale transformation, calculation of the logarithm of spectral powers, discrete cosine transformation, and selection of relevant coefficients.

To summarize, to obtain an MFCC from an audio signal

- Take the signal's Fourier transform (STFT).
- Map the powers of the resulting spectrum to the Mel scale, multiplying by overlapping window functions (triangular or cosine).
- Take the logarithm of the amplitudes at each mel frequency.
- Take the discrete cosine transform (DCT) of the list of logarithmic powers of the mel frequencies, as if it were a signal. The MFCCs are the amplitudes of the resulting spectrum.

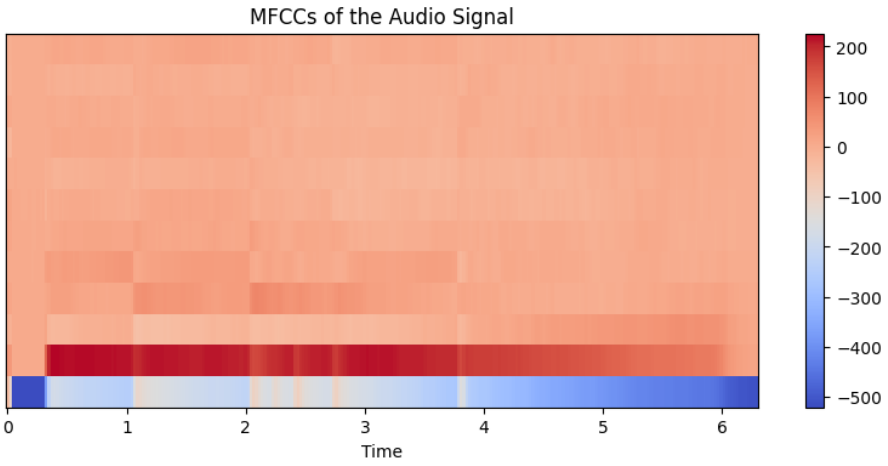


Figure 9: MFCC

$$X_k = \sum_{n=0}^{N-1} x_n \cos\left[\frac{\pi}{N}\left(n+\frac{1}{2}\right)k\right] \tag{4}$$

4.4 Signal reconstruction

To compare the most commonly used coding methods, we used a SNN neuron to encode and then decode MFCCs extracted from an audio signal. The two most commonly used coding approaches were compared: frequency coding and latency coding.

RESULTS: Reconstruction Quality

The results show that latency encoding (when normalized and linearized and with a sufficient number of time steps) leads to a better reconstruction quality of the audio signal than frequency encoding, which is much more subject to small errors that propagate during the reconstruction, errors due to the probability of emitting a pulse at each time step that does not perfectly represent the pixel intensity. This improvement can be explained by the ability of latency coding to capture fine temporal variations in the signal. However, due to the loss of information during signal transformation in MFCC, and more specifically in DCT, no reconstruction reaches the quality of the original signal.

In addition, with latency encoding, a pulse carries much more information, which can lead to less robustness in the face of noise, which can be very present in our dataset. We will therefore try to compare the two approaches in our SNN.

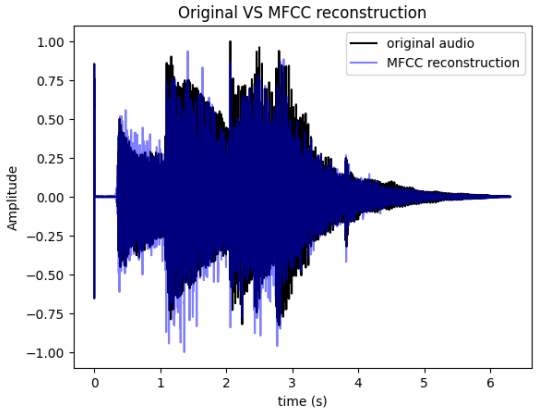


Figure 10: Original VS MFCC reconstruction

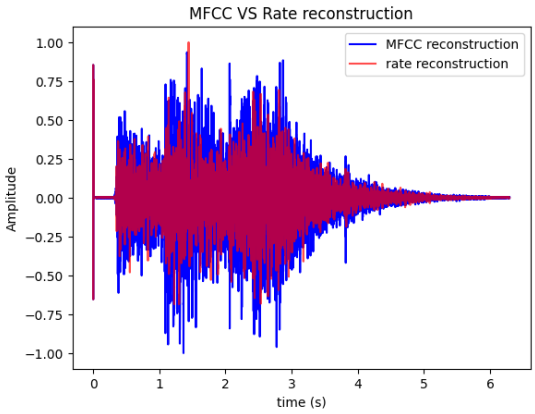


Figure 11: MFCC VS Rate reconstruction

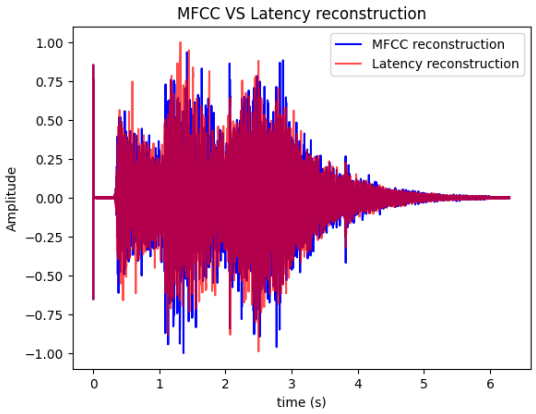


Figure 12: MFCC VS Latency reconstruction

5 Challenges and Solutions

Pipeline parallelization

The first challenge we faced was the incompatibility of the `youtube-dl` library with the `ffmpeg` library, which can vary depending on the Python version you use. For stability reasons, we decided to use the `yt-dlp` Python implementation instead of calling a shell command to perform operations (problems with the conda environment).

Data Verification

Another challenge was to verify the integrity of the data (not corrupted, no missing information, exactly 10 seconds long) and the download process (checking the existence of the audio url to download).

In addition, since we could not predict all possible outcomes that might occur, we had to go back and forth between data verification and data download to correct the problems that occurred (it took quite a while).

Finding the relevant number of MFCCs features

One of the issues was to find the relevant number of MFCCs features.

Increasing the number of MFCC coefficients increases the dimensionality of the feature vectors. While a higher dimensional feature space can capture more information, it also results in increased computational cost and memory requirements. Due to our limited resources, we had to find a compromise between the accuracy of the MFCC and the complexity.

It seems that the first few MFCC coefficients capture the most relevant information for audio processing (see figure 9). These coefficients represent the signal's spectral envelope. As you go down the list of coefficients, they may contain less critical information and be more susceptible to noise.

The number of coefficients is also related to the diversity of our data; a more diverse dataset may require more coefficients as more features will be recognizable.

Reconstruct the audio

To reconstruct the audio, the data had to be properly normalized to create the MFCC and then denormalized to extract good audio. We used sklearn's MinMaxScaler fitted to the data (which can apply both normalization and inverse normalization) to achieve our results.

Latency encoding issues

In latency encoding, there is a logarithmic dependence between input feature intensity and spiking timing, which can be problematic for reconstruction when the number of time steps is insufficient. To mitigate this problem, we applied a linear latency coding instead of a logarithmic one, and normalized it so that each spike occurs within our time steps.

6 Conclusion and Future Perspectives

In conclusion, we have made some progress in the implementation of a data preprocessing pipeline, the theoretical study of SNNs, and the reconstruction of audio signals. Our exploration of different data augmentation methods, neural models, and coding schemes gave us an insight into the possibilities and, most importantly, the challenges of SNNs in audio classification tasks. Our study of signal reconstruction highlighted the trade-offs between frequency and latency coding. While latency coding showed promise in capturing fine temporal variations. Comparing different encoding methods seems to be crucial for determining the most appropriate approach for our SNN-based audio classification.

This also gave us some lines of improvement:

1. Combining our efforts to design, implement, and train SNN models.
2. Implement SNN models for audio classification using the knowledge gained from our study.
3. Compare different models of SNNs with different types of encoding.
4. Complete the implementation of the existing ANN models.
5. Perform performance evaluation.

Audio samples

Figure 13: Original Audio

Figure 14: MFCC Reconstruction

Figure 15: Latency Reconstruction

Figure 16: Rate Reconstruction

7 Technical environment

7.1 Computational tools

We worked on our personal computers and we used the following computational tools:

Softwares	Visual Studio Code Jupyter Notebook Git Github Google Colab Anaconda Overleaf
Programming languages	Python Latex
Libraries	Pytorch Librosa Numpy Pandas Matplotlib sox Youtube-dl
Frameworks	SnnTorch

Figure 17: Computational tools

(Our Github repository)

References

[1] L. Deng, Y. Wu, X. Hu, L. Liang, Y. Ding, G. Li, G. Zhao, P. Li, and Y. Xie, *Rethinking the performance comparison between SNNS and ANNS*. https://www.sciencedirect.com/science/article/pii/S0893608019302667?ref=pdf_download&fr=RR-2&rr=829f60cc982b9a09, 2020.

[2] R. N. Wulfram Gerstner, Werner M. Kistler and L. Paninski, *Neuronal Dynamics, from single neurons to networks and models of cognition*. https://neuronaldynamics.epfl.ch/index.html, 2014.