National Technical University of Athens
School of Electrical and Computer Engineering
MSc in Data Science & Machine Learning

Audio Signal Processing and Audio Fingerprinting: Implementing a Music Recognition System

Diploma Thesis

of

CHRISTOS NIKOU

Supervisor: Konstantinos Karantzalos
Professor @ NTUA

Co-supervisor: Theodoros Giannakopoulos
Researcher B @ NCSR Demokritos

Athens, November 2023

National Technical University of Athens
School of Electrical and Computer Engineering
MSc in Data Science & Machine Learning

Audio Signal Processing and Audio Fingerprinting: Implementing a Music Recognition System

Diploma Thesis

of

CHRISTOS NIKOU

Supervisor: Konstantinos Karantzalos
Professor @ NTUA

Co-supervisor: Theodoros Giannakopoulos
Researcher B @ NCSR Demokritos

Approved by the examination committee on November 3rd, 2023.

(Signature)	*(Signature)*	*(Signature)*
Konstantinos Karantzalos	Athanasios Boulodimos	Theodoros Giannakopoulos
Professor @ NTUA	Assistant Professor @ NTUA	Researcher B @ NCSR Demokritos

Athens, November 2023

National Technical University of Athens
School of Electrical and Computer Engineering
MSc in Data Science & Machine Learning

DISCLAIMER ON ACADEMIC ETHICS AND INTELLECTUAL PROPERTY RIGHTS

Being fully aware of the implications of copyright laws, I expressly state that this diploma thesis, as well as the electronic files and source codes developed or modified in the course of this thesis, are solely the product of my personal work and do not infringe any rights of intellectual property, personality and personal data of third parties, do not contain work / contributions of third parties for which the permission of the authors / beneficiaries is required and are not a product of partial or complete plagiarism, while the sources used are limited to the bibliographic references only and meet the rules of scientific citing. The points where I have used ideas, text, files and / or sources of other authors are clearly mentioned in the text with the appropriate citation and the relevant complete reference is included in the bibliographic references section. I fully, individually and personally undertake all legal and administrative consequences that may arise in the event that it is proven, in the course of time, that this thesis or part of it does not belong to me because it is a product of plagiarism.

(Signature)

. . . . . .
Christos Nikou
November 3rd, 2023

Abstract

Song identification is one of oldest and perhaps one of the most popular Music Information Retrieval tasks. It has received a lot of attention in both academic research and the industry. One of the most well-known commercial applications for song identification is Shazam. Shazam's algorithm is based on a technique called audio fingerprinting. It extracts a compact representation by hashing the spectral peaks on the spectrogram of the audio fragments. This way, Shazam achieves a representation that is robust to noise distortions that may occur in realistic scenarios where music is captured from portable devices. A non-exhaustive technique based on inverted lists allows for fast and efficient retrievals. However, recently, Google launched a totally different music recognition by utilizing the pioneering ideas of deep learning. Instead of inventing sophisticated audio representations, they trained a deep neural network to automatically extract robust and meaningful audio representations.

In this thesis, we present how these two music recognition systems can be implemented, both theoretically and practically. We compare these two approaches in terms of their performance, storage requirements, and scalability. In the case of the deep learning approach, we introduce a novel data augmentation pipeline that further improves the performance of the overall system prior to already existing systems employing the same approach. This way, our system is ideal for real time applications. The thesis focuses on both the theoretical and practical aspects of audio signal processing. We begin from the fundamental concept of the Fourier Transform and we develop the theory by introducing the most important techniques and features used in audio signal processing nowadays. We explain how the algorithm of Shazam works, and we implement a music recognition using an open source library. In the last chapters, we turn our attention to deep learning. We introduce the basic concepts of deep learning and we develop a music recognition system by training a neural network with contrastive loss. Finally, we experimentally test the performance of these systems. This thesis was meant to be as self-contained as possible. Our code is publicly available (https://github.com/ChrisNick92/deep-audio-fingerprinting).

Keywords

Song identification, Music Information Retrieval, Audio Signal Processing, Deep Learning, Contrastive Learning.

Acknowledgements

Many thanks to Konstantinos Karantzalos for accepting to supervise my thesis, and to Theodoros Giannakopoulos for introducing me to area of audio signal processing. Our collaboration surpassed my expectations as both of them generously provided the freedom to explore and experiment without constraints, offering their assistance whenever needed. Many thanks to my colleagues at NCSR Demokritos for our collaboration during the Museek project. They made me feel like home from the very first moment. A big thanks goes to my colleagues from the Data Science & Machine Learning program at NTUA for making a demanding curriculum seem like a piece of cake. They were always willing to share their thoughts and ideas, and to offer their support. Last but not least, I would like to thank my family, my parents, and my sister for their constant encouragement and support.

Athens, November 2023

Christos Nikou

Table of Contents

Chapter 1

Synopsis

1.1 Ελληνικά

Σε αυτό το πρώτο κεφαλαίο δίνουμε μια σύντομη περιγραφή της παρούσας διπλωματικής εργασίας. Σκοπός αυτού του κεφαλαίου είναι να αποτελέσει ένα πρώτο ανάγνωσμα της εργασίας παρουσιάζοντας συνοπτικά τη δομή, τα περιοχόμενα και τα αποτελέσματα του κάθε κεφαλαίου.

Στο Κεφάλαιο 2 παρουσιάζουμε τον μετασχηματισμό Fourier, το βασικό εργαλείο για τη μελέτη και την επεξεργασία σήματος και αναπτύσσουμε τις βασικές του ιδιότητες. Πιο συγκεκριμένα, η μέλετη ξεκινά με την περίπτωση των περιοδικών συναρτήσεων (ή περιοδικών σημάτων) $f : \mathbb{R} \to \mathbb{R}$. Δηλαδή, των συναρτήσεων για τις οποίες υπάρχει $T \in \mathbb{R}$ τέτοιος ώστε

$$f(x + T) = f(x),$$

για κάθε $x \in \mathbb{R}$. Σε αυτό το πλαίσιο εντάσσονται και οι συναρτήσεις της μορφής $\sin(\pi(nt - \phi_n))$, που αποτελούν τα δομικά στοιχεία της θεωρίας. Ο αριθμός n είναι η συχνότητα του σήματος και μετράται σε Hertz ενώ ο αριθμός ϕ_n αντιστοιχεί στη φάση του σήματος. Η σπουδαιότητα των συναρτήσεων αυτής της μορφής, έγκειται στο γεγονός ότι υπό ορισμένες προϋποθέσεις, κάθε περιοδική συνάρτηση $f : \mathbb{R} \to \mathbb{R}$ μπορεί να προσεγγίστεί από ένα, ενδεχομένως και άπειρο άθροισμα

$$\sum_{n=1}^{\infty} A_n \sin(2\pi(nt - \phi_n)), \tag{1.1.1}$$

τέτοιων συναρτήσεων. Με την αναπαράσταση της (1.1.1) μπορούμε να δούμε τη συνεισφορά κάθε συχνότητας n στο περιοδικό σήμα. Έτσι, στην παράγραφο 2.1 παρουσιάζουμε με αποδείξεις (όπου αυτό είναι εφικτό), πώς και υπό ποιες προύποθέσεις μπορούμε να πετύχουμε αυτή την αναπαράσταση. Στην παράγραφο 2.2 επεκτείνουμε τη θεωρία για σήματα $f : \mathbb{R} \to \mathbb{R}$ που δεν είναι κατ' ανάγκη περιοδικά. Η ανάγκη αυτής της γενίκευσης οφείλεται στο ότι τα περισσότερα ηχητικά σήματα που συναντάμε στην πράξη είναι μη περιοδικά. Έτσι, κρίνεται απαραίτητο να εξετάσει κανείς πώς και με ποιό τρόπο μπορεί κανείς από την κυματομορφή του σήματος να εξάγει την συχνοτική συμπεριφορά του σήματος. Δηλαδή, να διακρίνει ποιες συχνότητες συνεισφέρουν στο σήμα και ποιες όχι. Σε αυτή την περίπτωση, το βασικό εργαλείο

είναι ο μετασχηματισμός Fourier

$$\mathcal{F}f(s) = \int_{-\infty}^{\infty} f(t)e^{-2\pi ist}\,dt. \tag{1.1.2}$$

Έτσι, σε αυτή την παράγραφο, παρουσιάζουμε πώς μπορούμε να επεκτείνουμε την έννοια του μετασχηματισμού Fourier για μη περιοδικές συναρτήσεις. Επίσης, εισάγουμε τον αντίστροφο μετασχηματισμό Fourier

$$\mathcal{F}^{-1}g(t) = \int_{-\infty}^{\infty} g(s)e^{2\pi ist}\,ds, \tag{1.1.3}$$

ο οποίος παίζει τον ρόλο της αντίστροφης απεικόνισης της (1.1.2) που περιγράφεται μέσω των σχέσεων

$$\mathcal{F}\left(\mathcal{F}^{-1}f\right)(t) = f(t), \quad \mathcal{F}^{-1}\left(\mathcal{F}f\right)(s) = f(s). \tag{1.1.4}$$

Η χρησιμότητα των σχέσεων της (1.1.4) γίνεται εμφανής όταν συνδυάζονται με το θέωρημα της συνέλιξης που μας λέει ότι ο μετασχηματισμός Fourier που προκύπτει από τη συνέλιξη

$$(f * g)(t) = \int_{-\infty}^{\infty} f(t-x)g(x)\,dx, \tag{1.1.5}$$

δύο σημάτων f, g ισούται με το γινόμενο

$$\mathcal{F}(f * g)(s) = \mathcal{F}f(s) \cdot \mathcal{F}g(s), \tag{1.1.6}$$

των μετασχηματισμών Fourier $\mathcal{F}f, \mathcal{F}g$ των f, g. Ο τρόπος με τον οποίο χρησιμοποιούνται οι δύο παραπάνω σχέσεις είναι όταν θέλουμε να εφαρμόσουμε κάποιον μετασχηματισμό σε ένα δοσμένο σήμα $f : \mathbb{R} \to \mathbb{R}$. Για παράδειγμα, αν θέλουμε να ¨περικόψουμε¨ τις συχνότητες του σήματος πάνω από ένα κατώφλι $\upsilon_C > 0$, αρκεί να πολλαπλασιάσουμε τον μετασχηματισμό Fourier $\mathcal{F}f(s)$ με τη συνάρτηση

$$H(s) = \begin{cases} 1, & |s| < \upsilon_c \\ 0, & |s| \geq \upsilon_c \end{cases}.$$

Γνωρίζοντας ότι $h(t) = \mathcal{F}^{-1}H(t) = 2\upsilon_C \mathrm{sinc}(2\upsilon_C t)$ και χρησιμοποιώντας τον αντίστροφο μετασχηματισμό μπορούμε να προσδιορίσουμε τον τύπο $y(t) = (f * h)(t)$ του ζητούμενου σήματος. Στην παράγραφο αυτή παρουσιάζουμε και άλλες μορφές μετασχηματισμών που εφαρμόζονται αρκετά στην επεξεργασία σήματος. Η θεωρία αυτή μας βοηθάει να κατανοήσουμε καλύτερα το πως εφαρμόζονται αυτοί οι μετασχηματισμοί, τους οποίους θα χρησιμοποιήσουμε εκτενώς για να εκπαιδεύσουμε ένα νευρωνικό δίκτυο για να εξάγουμε πληροφορία από μικρά ηχητικά αποσπάσματα διαφόρων τραγουδιών. Στις παραγράφους 2.3 & 2.4 επικεντρωνόμαστε στη διακριτή περίπτωση του μετασχηματισμού Fourier που χρησιμοποιείται στην πράξη από τους υπολογιστές για την επεξεργασία σημάτων. Αυτό συμβαίνει διότι ο υπολογιστής μπορεί να διαχειριστεί μόνο ένα πεπερασμένο πλήθος τιμών. Ο τρόπος με τον οποιό περνά κανείς από τη συνεχή περίπτωση του μετασχηματισμού της (1.1.2) στη διακριτή περίπτωση είναι ο εξής: Αρχικά το αναλογικό σήμα $f : \mathbb{R} \to \mathbb{R}$ μετατρέπεται σε ένα διακριτό

σήμα $x : \mathbb{Z} \to \mathbb{R}$ μέσω της

$$x(n) = f(n \cdot T), \quad n \in \mathbb{Z},$$

Ο αριθμός $T > 0$ αντιστοιχεί στην περίοδο δειγματοληψίας, ενώ αντίστροφος αριθμός $F_s = 1/T$ στον ρυθμό δειγματοληψίας που μετράται σε Hertz. Η διαδικασία αυτή αναπαρίσταται γραφικά στο Σχήμα 1.1.

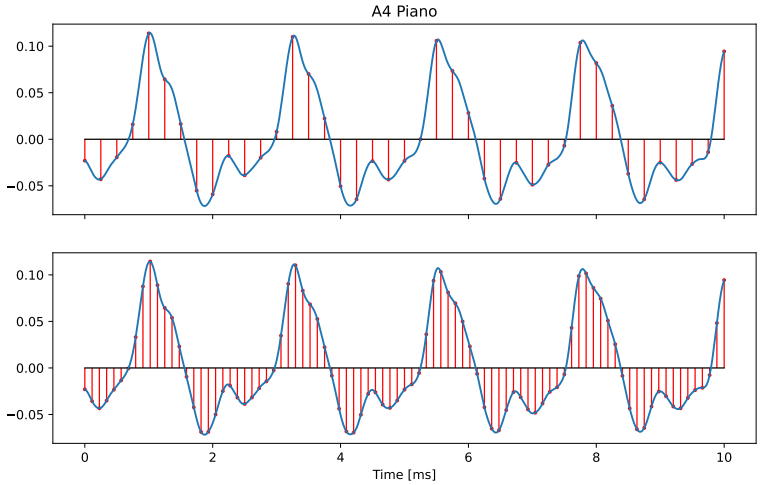

Figure 1.1. *Δειγματοληψία του σήματος που αντιστοιχεί στο A4 του πιάνο με ρυθμούς δειγματοληψίας 4kHz (πάνω γράφημα) και 8kHz (κάτω γράφημα).*

Εν συνεχεία, με τη βοήθεια των αθροισμάτων Riemann, ο μετασχηματισμός της (1.1.2) παίρνει τη μορφή

$$\int_{-\infty}^{\infty} f(t)e^{-2\pi ist}\,dt \approx T \cdot \sum_{n=-\infty}^{\infty} f(nT)e^{-2\pi isnT}. \tag{1.1.7}$$

Εν γένει, για ένα διακριτό σήμα $x : \mathbb{Z} \to \mathbb{R}$, ο μετασχησματισμός Fourier δίνεται μέσω της

$$\hat{x}(s) = \sum_{n=-\infty}^{\infty} x(n)e^{-2\pi isn}. \tag{1.1.8}$$

Σε αυτή την περίπτωση, η συχνότητα $\hat{x}(s)$ συνδέεται με την συχνότητα του αναλογικού σήματος μέσω της

$$\hat{x}(s) = \frac{1}{T}F\left(\frac{s}{T}\right), \tag{1.1.9}$$

όπου F ο μετασχηματισμός Fourier του αναλογικού σήματος $f : \mathbb{R} \to \mathbb{R}$, με $x(n) = f(nT)$. Τώρα, το πρόβλημα με την (1.1.8) είναι το άθροισμα που εμφανίζεται έχει άπειρους όρους και ότι η συχνότητα $s \in \mathbb{R}$ διατρέχει όλους τους πραγματικούς αριθμούς. Έτσι, αφού στην πράξη κάθε σήμα $x : \mathbb{Z} \to \mathbb{R}$ είναι πεπερασμένης διάρκειας, αυτό αναπαρίσταται από N τιμές $x(0), \ldots, x(N-1)$ στον υπολογιστή. Σε επίπεδο υπολογιστών, η (1.1.8) αντικαθίσταται από την

$$X(k) = \hat{x}(k/N) = \sum_{n=0}^{N-1} x(n)e^{-2\pi ikn/N}, \tag{1.1.10}$$

για $k = 0, \ldots, N-1$. Με βάση την (1.1.9) η σχέση μεταξύ του k-οστού συντελεστή Fourier

$X(k)$ και της συχνότητας του αναλογικού σήματος $f : \mathbb{R} \to \mathbb{R}$ δίνεται μέσω της

$$X(k) = \hat{x}(k/N) \approx \frac{1}{T} F\left(\frac{k}{NT}\right). \tag{1.1.11}$$

Τώρα, η σχέση στην (1.1.11) μας δίνει πληροφορία για τη συνεισφορά των συχνοτήτων k/NT για όλη τη διάρκεια του σήματος. Έτσι, με αυτόν τον τρόπο, η χρονική άφιξη αυτών των συχνοτήτων παραμένει κρυφή. Γι' αυτό τον λόγο, εισάγεται η έννοια του βραχυπρόθεσμου μετασχηματισμού Fourier, όπου εφαρμόζεται ο μετασχηματισμός της (1.1.11) σε μικρά παράθυρα του σήματος. Δηλαδή, για δοσμένο μήκος παραθύρου N, που αντιστοιχεί στο πλήθος των δειγμάτων, και για ένα άλμα (hop length) H, εφαρμόζεται ο μετασχηματισμός

$$\mathcal{X}(m, k) = \sum_{n=0}^{N-1} x(n + mH) w(n) e^{-2\pi i n k/N}, \tag{1.1.12}$$

για $m \in \mathbb{Z}$ και $k = 0, \ldots, N-1$. Με αυτόν τον τρόπο, καταλήγουμε σε μια δισδιάστατη αναπαράσταση $\mathcal{X}(m, k)$ ενός αναλογικού σήματος $f : \mathbb{R} \to \mathbb{R}$ η οποία μας δίνει πληροφορία για τις συχνότητες του σήματος, καθώς και σε ποιές χρονικές στιγμές αυτές εμφανίζονται. Δηλαδή, η ποσότητα $\mathcal{X}(m, k)$ περιέχει την πληροφορία για το μέγεθος και τη φάση της συχνότητας k/NT τη χρονική στιγμή mH/F_s, όπου F_s ο ρυθμός δειγματοληψίας του αναλογικού σήματος $f : \mathbb{R} \to \mathbb{R}$. Μέσω της (1.1.12) ορίζεται ή έννοια του φασματογράμματος, ενός από τα πιο διαδεδομένα χαρακτηριστικά που χρησιμοποιούνται στην επεξεργασία ήχου. Το φασματόγραμμα δίνεται από το μέτρο των μιγαδικών αριθμών $|\mathcal{X}(m, k)|^2$, μέσω του οποίου μπορούμε να διακρίνουμε την ενέργεια των συχνοτήτων k/NT τις χρονικές στιγμές mH/F_s. Στο Σχήμα 1.2 φαίνεται η κυματομορφή του σήματος που προκύπτει από το πλήκτρο A4 στο πιάνο (μετά από δειγματοληψία) και το αντίστοιχο φασματόγραμμα. Όπως φαίνεται και στο σχήμα η περισσότερη ενέργεια συσσωρεύεται στη συχνότητα 440 Hz που είναι και η συχνότητα αυτής της νότας.

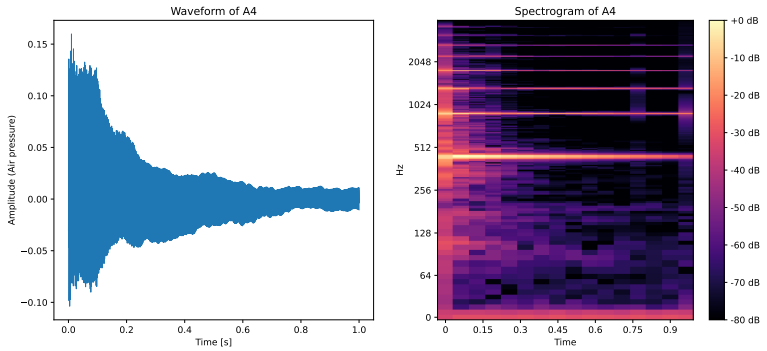

Figure 1.2. *Η κυματομορφή της νότας A4 στο πιάνο και το αντίστοιχο φασματόγραμμα.*

Έχοντας αναπτύξει τα απαραίτητα εργαλεία γύρω από τον μετασχηματισμό Fourier, στο Κεφάλαιο 3 κάνουμε την εισαγωγή στην επεξεργασία ήχου. Στην παράγραφο 3.1 παρουσιάζουμε τους διάφορους τρόπους με τους οποίους μπορεί να αναπαρασταθεί η μουσική πληροφορία. Στη δική μας περίπτωση, η σημαντικότερη αναπαράσταση είναι αυτή που περιγραφεί ένα ηχητικό σήμα ως μια συνάρτηση $f : [0, \infty) \to \mathbb{R}$ η οποία για μια δεδομένη χρονική στιγμή

$t > 0$ μετρά τη πίεση του αέρα σε ένα δεδομένο σημείο. Με αυτόν τον τρόπο, βλέποντας τα ηχητικά σήματα ως συναρτήσεις $f : \mathbb{R} \to \mathbb{R}$ μπορούμε να εκμεταλλευτούμε τη θεωρία του 2ου κεφαλαίου. Στην παράγραφο 3.3 παρουσιάζουμε τα σημαντικότερα χαρακτηριστικά που εξάγονται με χρήση του διακριτού μετασχηματισμού Fourier στην επεξεργασία ήχου. Μια από τις σημαντικότερες αναπαραστάσεις πάνω στην οποία βασιζόμαστε για την ανάπτυξη των μοντέλων βαθιάς μάθησης είναι του mel-φασματογράμματος που παρουσιάζεται στην 3.3.3. Η διαφορά με το φασματόγραμμα είναι ότι αυτή η αναπαράσταση μετατρέπει τον άξονα των συχνοτήτων σε λογαριθμική κλίμακα που εκφράζεται μέσω της σχέσης

$$m = 2595 \cdot \log_{10}\left(1 + \frac{f}{700}\right). \tag{1.1.13}$$

Στην υποπαράγραφο αυτή, παρουσιάζουμε αναλυτικά τα βήματα του πώς μπορεί κανείς να περάσει από το φασματόγραμμα στην αναπαράσταση του mel-φασματογράμματος. Ο λόγος για τον οποίο γίνεται αυτή η μετατροπή στον άξονα των συχνοτήτων είναι διότι έχει αποδειχθεί, μέσω ακουστικών πειραμάτων, ότι ο άνθρωπος αντιλαμβάνεται τις συχνότητες λογαριθμικά και όχι γραμμικά. Αυτό αιτιολογεί και το γεγονός ότι οι νότες του πιάνο που απέχουν μεταξύ τους ακριβώς μια οκτάβα έχουν το ίδιο «χρώμα» ακούσματος. Στο Σχήμα 1.3 βλέπετε το mel-φασματόγραμμα που αντιστοιχεί στις νότες C2, C4, G6 και A6 στο πιάνο.

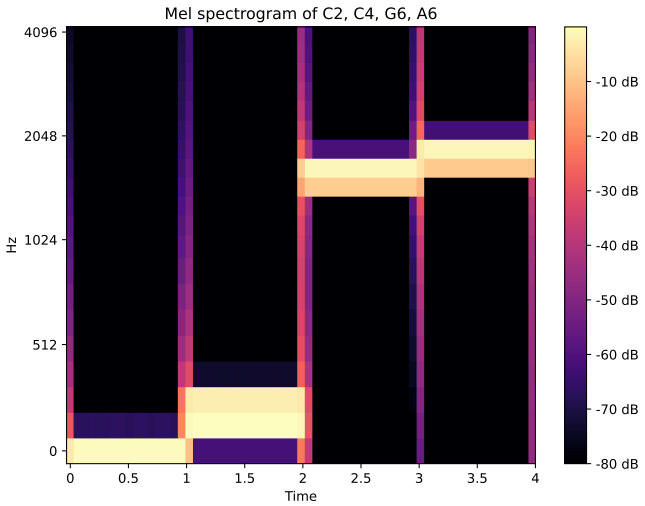

Figure 1.3. *Οι νότες C2, C4, G6, και A6 στο πιάνο.*

Εν συνεχεία, στην παράγραφο 3.4 παρουσιάζουμε και περιγράφουμε μια σειρά από μετασχηματισμούς που χρησιμοποιούνται στην ανάλυση ήχου, όπως για παράδειγμα η προσθήκη θορύβου, η αντήχηση αλλά και το «κλιπάρισμα» συγκεκριμένων συχνοτήτων. Ο λόγος που εστιάζουμε σε αυτούς τους μετασχηματισμούς και προσπαθούμε να τους καταλάβουμε είναι διότι τέτοιες παραμορφώσεις προκαλούνται αρκετά στα ηχητικά σήματα που καταγράφονται από φορητές συσκευές. Έτσι, εάν θέλουμε να αναπτύξουμε ένα αποδοτικό σύστημα αναγνώρισης μουσικής που να λειτουργεί σε πραγματικές συνθήκες θα πρέπει να αναπτύξουμε εργαλεία που να εξουδετερώνουν αυτές τις παραμορφώσεις. Με αυτόν τον τρόπο θα εκπαιδεύσουμε και το μοντέλο βαθιάς μάθησης στο Κεφάλαιο 5. Στην τελευταία παράγραφο αυτού

του κεφαλαίου κάνουμε την εισαγωγή στο πρόβλημα της αναγνώρισης μουσικής (song identification). Αφού παρουσιάσουμε το βασικό πρόβλημα καθώς και τις προκλήσεις που πρέπει να αντιμετωπιστούν κατά τη σχεδίαση ενός συστήματος αναγνωρίσης μουσικής, παρουσιάζουμε αναλυτικά τη μέθοδο που χρησιμοποιεί ο αλγόριθμος του Shazam για την αναγνώριση των κομματιών. Η ιδέα εισήχθη από τους Wang et al. [1] και βασίζεται στην εξαγωγή μουσικής πληροφορίας η οποία είναι όσο το δυνατόν πιο ανθεκτική στις παραμορφώσεις που μπορεί να εμφανιστούν σε ένα πραγματικό σενάριο. Η ιδέα είναι κανείς να εστιάσει στα τοπικά μέγιστα του φασματογράμματος ενός μικρού ηχητικού αποσπάσματος με την ελπίδα ότι αυτές οι τοπικές τιμές θα διατηρηθούν ακόμα και να επέλθουν έντονες παραμορφώσεις. Τα τοπικά μέγιστα είναι γνωστά ως ηχητικά αποσπάσματα (audio fingerprints) και αποτελούν το κύριο συστατικό της ηχητικής αναπαράστασης αυτής της μεθόδου. Στο Σχήμα 1.4 βλέπετε το φασματόγραμμα και τα τοπικά μέγιστα σε ένα απόσπασμα των Τεσσάρων Εποχών - Άνοιξη του Vivaldi χωρίς παραμορφώσεις. Όπως φαίνεται και από το σχήμα, ενώ τα φασματογράμματα παρουσιάζουν μεγάλη απόκλιση, από την άλλη, τα περισσότερα τοπικά μέγιστα εμφανίζονται στις ίδιες θέσεις.

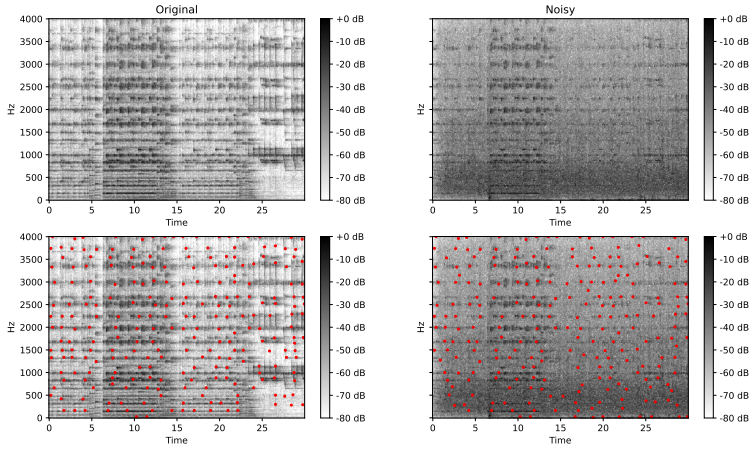

Figure 1.4. *Τα τοπικά μέγιστα απο ένα ηχητικό απόσπασμα των Τεσσάρων Εποχών - Άνοιξη του Vivaldi. Στα αριστερά είναι το φασματόγραμμα και τα τοπικά μέγιστα του καθαρού σήματος, ενώ στα δεξιά βλέπουμε το αντίστοιχο φασματόγραμμα για το ίδιο απόσπασμα με προσθήκη θορύβου με* 0 *SNR (dB).*

Το πρώτο βήμα για την εξαγωγή αυτής της αναπαράστασης είναι η εξαγωγή των τοπικών μεγίστων. Προς τούτο, επιλέγονται δύο σταθερές $k_0, \tau_0 > 0$ που ορίζουν την περιοχή $[-k_0, k_0] \times [-\tau_0, \tau_0]$. Έτσι, ένα σημείο (k^*, m^*) του φασματογράμματος είναι τοπικό μέγιστο αν και μόνο αν

$$\left|\mathcal{X}(k^*, m^*)\right| > \left|\mathcal{X}(k, m)\right|, \tag{1.1.14}$$

για κάθε $(k, m) \in [k^* - k_0, k^* + k_0] \times [m^* - \tau_0, m^* + \tau_0]$. Τώρα, σε αφαιρετικό επίπεδο, μπορούμε να σκεφτόμαστε τη βάση σε αυτό το πρώτο στάδιο έχοντας τοποθετήσει όλα τα φασματογράμματα που αντιστοιχούν στα διαφορετικά κομμάτια το ένα μετά το άλλο σε ένα μεγάλο ενιαίο φασματόγραμμα. Σε αυτή την αναπαράσταση, σε μια δεύτερη δομή μπορούμε να κρατάμε τα χρονικά πλαίσια που τελειώνει το κάθε κομμάτι και αρχίζει το επόμενο. Έτσι,

ακολουθώντας τον συμβολισμό του [2], αν συμβολίσουμε με $\mathcal{D}$ το σύνολο των κομματιών και με $C(\mathcal{D})$ το σύνολο των σημείων (k, m) που αντιστοιχούν στα τοπικά μέγιστα αυτού του ενιαίου φασματογράμματος, μπορούμε σε αυτό το σημείο να περιγράψουμε μια αφελής προσέγγιση για την αναγνώριση ενός ηχητικού αποσπάσματος Q, δοθέντος των $\mathcal{D}$, και $C(\mathcal{D})$. Στο πρώτο βήμα, υπολογίζονται τα τοπικά μέγιστα $C(Q)$ που αντιστοιχούν στο φασματόγραμμα του παραδείγματος (Query). Μια αφελής προσέγγιση για να βρούμε το κομμάτι της βάσης με τη μεγαλύτερη ομοιότητα είναι να θεωρήσουμε τις χρονικές μετατοπίσεις

$$m + C(Q) = \{(k, m + n) : (k, n) \in C(Q)\}. \tag{1.1.15}$$

Τότε, το χρονικό κομμάτι m^* της βάσης μπορεί να προσδιοριστεί μέσω της

$$m^* = \arg\max_{m \in \mathbb{N}} \Delta_Q(m) = \arg\max_{m \in \mathbb{N}} \frac{|(m + C(Q)) \cap C(\mathcal{D})|}{|C(Q|}, \tag{1.1.16}$$

όπου με $|C(Q)|$ συμβολίζουμε τον πληθάριθμο του συνόλου $C(Q)$, αντίστοιχα και για το σύνολο $(m + C(Q)) \cap C(\mathcal{D})$. Με άλλα λόγια, η (1.1.16) μας λέει ότι το κομμάτι της βάσης που έχει τη μεγαλύτερη ομοιότητα με το ηχητικό απόσπασμα είναι αυτό που συμφωνεί στα περισσότερα τοπικά μέγιστα του αντίστοιχου φασματογράμματος. Χρησιμοποιώντας τη δεύτερη δομή, έχοντας το βέλτιστο χρονικό σημείο m^*, μπορούμε να βρούμε το κομμάτι που αντιστοιχεί σε αυτό το σημείο m^*, και κατ' επέκταση να ανακτήσουμε και την πληροφορία για τον καλλιτέχνη του κομματιού. Παρ' όλα αυτά, η προσέγγιση αυτή δεν είναι αποτελεσματική για εφαρμογές πραγματικού χρόνου γιατί ο χρόνος αναζήτησης αυξάνεται γραμμικά με το μέγεθος της βάσης, με αποτέλεσμα η ανάκτηση της πληροφορίας να είναι αργή για βάσεις που περιέχουν χιλιάδες κομμάτια. Έτσι, στην πράξη υλοποιούνται μη εξαντλητικές μέθοδοι αναζήτησης που μειώνουν τον χώρο αναζήτησης επιλέγοντας να ελέγξουν μόνο κάποια συγκεκριμένα «μέρη» της βάσης και όχι όλη τη βάση. Στη περίπτωση του αλγορίθμου του Shazam, η αναζήτηση βασίζεται στις ανεστραμμένες λίστες (inverted lists). Στην υποπαράγραφο 3.5.3 παρουσιάζουμε αυτή την τεχνική καθώς και τη μέθοδο των Wang et al. για να βελτιώσουν περαιτέρω την απόδοση του συστήματος.

Στο κεφάλαιο 4 κάνουμε την εισαγωγή στη βαθιά μάθηση. Σκοπός αυτού του κεφαλαίου είναι να παρουσιάσουμε τις βασικότερες έννοιες που θα χρησιμοποιηθούν για την ανάπτυξη ενός συστήματος αναγνώρισης μουσικής που χρησιμοποιεί ένα νευρωνικό δίκτυο για την εξαγωγή της ηχητικής πληροφορίας. Έτσι, στην παράγραφο 4.1.1 παρουσιάζουμε τη βασική αρχιτεκτονική νευρωνικών δικτύων, τα feed forward neural networks. Σε αφαιρετικό επίπεδο κάθε τέτοιο δίκτυο είναι μια συνάρτηση $f : \mathbb{R}^K \to \mathbb{R}^N$. Η γενική της μορφή εκφράζεται ως μια σύνθεση συναρτήσεων της μορφής

$$f = f^{(n)} \circ f^{(n-1)} \cdots \circ f^{(1)}, \tag{1.1.17}$$

όπου κάθε $f^{(i)} = \sigma_i \circ T_i$, με $T_i : \mathbb{R}^{K_i} \to \mathbb{R}^{K_{i+1}}$ είναι μια γραμμική απεικονίση και $\sigma_i : \mathbb{R} \to \mathbb{R}$ είναι μια μη γραμμική συνάρτηση ενεργοποίησης. Εφόσον κάθε συνάρτηση $T_i : \mathbb{R}^{K_i} \to \mathbb{R}^{K_{i+1}}$ είναι γραμμική, τότε θα γράφεται στη μορφή $T_i(x) = W_i \cdot x$, για κάποιον πίνακα $W_i \in \mathbb{R}^{K_{i+1} \times K_i}$. Οι τιμές των πινάκων W_i συμβολίζονται με $\boldsymbol{\theta}$ και αποτελούν τις παραμέτρους του μοντέλου.

Έτσι, για να δηλώσουμε την εξάρτηση του δικτύου γράφουμε συνήθως $f(\cdot\,;\boldsymbol{\theta})$ αντί για $f(\cdot)$. Στην ενότητα 4.2, αφού παρουσιάσουμε τα διάφορα είδη προβλημάτων που εμφανίζονται στη βαθιά μάθηση, όπως την επιβλεπόμενη, τη μη επιβλεπόμενη και την ήμι-επιβλεπόμενη μάθηση, εισάγουμε το βασικό θεωρητικό πλαίσιο που βασίζεται η εκπαίδευση των νευρωνικών δικτύων. Στόχος σε κάθε πρόβλημα βαθιάς μάθησης είναι δεδομένου μιας συνάρτησης κόστους J, να βρεθούν οι κατάλληλοι παράμετροι του μοντέλου $\boldsymbol{\theta}$ που ελαχιστοποιούν τη J. Για παράδειγμα, στα προβλήματα επιβλεπόμενης μάθησης μας δίνεται ένα σύνολο δεδομένων $\{(x_1, y_1), \ldots, (x_n, y_n)\}$, όπου κάθε δείγμα x_i ανήκει σε μια από τις K κατηγορίες. Στόχος είναι προσδιορίσουμε τις παραμέτρους $\boldsymbol{\theta}^*$ ώστε $f(x_i\,;\boldsymbol{\theta}^*) = y_i$. Σε αυτή την περίπτωση η συνάρτηση J που επιλέγεται είναι η cross entropy που δίνεται μέσω της

$$J(f(x), y_x) = -\sum_{i=1}^{K} \log \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \cdot \mathbb{1}_{i=y_x}, \tag{1.1.18}$$

όπου $f(x) = (z_1, \ldots, z_n)$ και $y_x \in \{1, \ldots, K\}$. Τώρα, η εύρεση των κατάλληλων παραμέτρων γίνεται μέσω των αλγορίθμων που βασίζονται στη κατάβαση κλίσης (gradient based algorithms). Η ιδέα όλων αυτών των αλγορίθμων βασίζεται στην επαναληπτική ανανεώση των βαρών προς την αντίθετη κατεύθυνση της κλίσης $\nabla_{\boldsymbol{\theta}} J$. Ο πιο βασικός αλγόριθμος που εμπίπτει σε αυτή την κατηγορία είναι ο αλγόριθμος στοχαστικής κατάβασης (stochastic gradient descent). Σε αυτόν τον αλγόριθμο τα δείγματα (x_i, y_i) εξετάζονται ένα-ένα και για κάθε τέτοιο ζευγάρι έχουμε μια ανανέωση των παραμέτρων. Έτσι, σε αυτή την περίπτωση η ανανέωση των παραμετρών περιγράφεται από την εξίσωση

$$\boldsymbol{\theta}^{(i+1)} = \boldsymbol{\theta}^{(i)} - \eta \cdot \nabla_{\boldsymbol{\theta}} J\left(f(x^{(i)}; \boldsymbol{\theta}^{(i)}), y^{(i)}\right). \tag{1.1.19}$$

Τώρα, στην πράξη χρησιμοποιούνται παραλλαγές της στοχαστικής κατάβασης κλίσης που επιτυγχάνουν καλύτερα αποτελέσματα. Στην ενότητα 4.3 εξετάζουμε κάποιες από αυτές τις παραλαγγές παρουσιάζοντας τα πλεονεκτήματα και μειονεκτήματά τους. Στην ενότητα 4.4 παρουσιάζουμε τη βασική δομή των Συνελικτικών Νευρωνικών Δικτύων (Convolutional Neural Networks), τα οποία χρησιμοποιούνται εκτενώς στην επεξεργασία δεδομένων που προέρχονται από εικόνες και ήχο. Η βασική πράξη που χαρακτηρίζει αυτά τα δίκτυα είναι αυτή της συνέλιξης που παρουσιάζεται στην παράγραφο 4.4.2. Μέσω της συνέλιξης τα δίκτυα αυτά έχουν τη δυνατότητα να εξάγουν χρήσιμη τοπική πληροφορία από δισδιάστατα αντικείμενα, όπως π.χ. εικόνες, φασματογράμματα, κτλ, μειώνοντας συγχρόνως και τις διαστάσεις τους. Στη δική μας περίπτωση, βασιζόμαστε σε αυτή την αρχιτεκτονική αυτή για να εξάγουμε μια συμπαγή αναπαράσταση από τα ηχητικά αποσπάσματα των τραγουδιών που θα χρησιμοποιήσουμε για το σύστημα αναγνώρισης.

Στο κεφάλαιο 5 παρουσιάζουμε το σύστημα αναγνωρίσης μουσικής που βασίζεται στις ιδέες της βαθιάς μάθησης. Η βασική διαφορά σε σχέση με το σύστημα που ακολουθεί τη προσέγγιση του Shazam, είναι ότι σε αυτή την περίπτωση εκπαιδεύουμε ένα νευρωνικό δίκτυο το οποίο αναλαμβάνει στη συνέχεια την εξαγωγή των χαρακτηριστικών από τα τραγούδια. Με αυτόν τον τρόπο, η προσοχή στρέφεται από την εύρεση εξειδικευμένων αλγορίθμων εξαγωγής χαρακτηριστικών στην εύρεση αποτελεσματικών αλγορίθμων και τεχνικών εκπαίδευσης. Στη συγκεκριμένη περίπτωση, η εκπαίδευση του δικτύου γίνεται μέσω της τεχνικής του contrastive

learning. Η ιδέα αυτής της τεχνικής είναι να αναπαραστήσει τα παραμορφωμένα ηχητικά αποσπάσματα από το ίδιο κομμάτι όσον το δυνατόν πλησιέστερα στα πραγματικά ηχητικά αποσπάσματα σε έναν Ευκλείδειο χώρο χαμηλότερης διάστασης από τον χώρο εισόδου των δεδομένων. Για αν επιτευγχθεί αυτό, απαραίτητη προϋπόθεση είναι η επιλογή κατάλληλων τεχνικών παραμορφώσης που ανταποκρίνονται σε ρεαλιστικά σενάρια. Έτσι, αφού παρουσιάσουμε την αρχιτεκτονική του συνελικτικού δικτύου στην παράγραφο 5.2.1, στην παράγραφο 5.2.2 κάνουμε την εισαγωγή στη μέθοδο του contrastive learning. Ιδιαίτερη σημασία έχει η επιλογή των μεθόδων εμπλουτισμού (data augmentation) που επιλέγονται για την εκπαίδευση του δικτύου, όπως η προσθήκη θορύβου με διάφορα SNR, η αντήχηση, η χρονική μετατόπιση αλλά και το «κλιπάρισμα» των χαμηλών/υψηλών συχνότητων. Ο συνδυασμός αυτών των παραμορφώσεων οδηγεί στην ανάπτυξη ενός μοντέλου ικανού να πραγματοποιεί ιδιαίτερα ακριβείς προβλέψεις και σε περιπτώσεις που ο ήχος καταγράφεται από μικρόφωνα φορητών συσκευών. Σε μαθηματικό επίπεδο, το νευρωνικό δίκτυο αποτελείται από δύο σκέλη: α) τον κωδικοποιητή (encoder) $f : \mathbb{R}^{256\times32} \to \mathbb{R}^{1024}$, οποίος είναι ουσιαστικά ένα συνελικτικό δίκτυο που δρα σε mel-φασματογράμματα διαστάσεων $\mathbb{R}^{256\times32}$ απεικονίζοντάς τα σε διανύσματα διαστάσεων 1024, και (β) τη συνάρτηση προβολής $g : \mathbb{R}^{1024} \to \mathcal{S}^{d-1}$, η οποία είναι ένα feed forward νευρωνικό δίκτυο που απεικονίζει τα διανύσματα εισόδου πάνω στη μοναδιαία σφαίρα

$$\mathcal{S}^{d-1} = \{z \in \mathbb{R}^d : \|z\|_2 = 1\},$$

στον $\mathbb{R}^d$. Στην περίπτωσή μας, οι διαστάσεις $\mathbb{R}^{256\times32}$ που προκύπτουν από την εφαρμογή του βραχυπρόθεσμου μετασχηματισμού Fourier αντιστοιχούν σε 1 δευτερόλεπτο ηχητικού αποσπάσματος. Επίσης, η διάσταση προβολής είναι ίση με $d = 128$. Έτσι, σε αδρές γραμμές, το νευρωνικό δίκτυο $g \circ f$ απεικονίζει ηχητικά αποσπάσματα του ενός δευτερολέπτου σε διανύσματα 128 διαστάσεων. Κατ' αναλογία με την περίπτωση του κλασικού fingerprinting, κάθε διάνυσμα $g \circ f(x)$ καλείται αποτύπωμα (fingerprint) του $x \in \mathbb{R}^{256\times32}$. Τώρα, συμβολίζοντας με $\mathcal{M}(x)$ το δείγμα που αντιστοιχεί στο φασματόγραμμα x αφού έχει υποστεί τις προαναφερθείσες παραμορφώσεις, τότε η ιδέα κατά τη διάρκεια εκπαίδευσης είναι φέρουμε όσο το δυνατό πιο κοντά το παραμορφωμένο δείγμα $\mathcal{M}(x)$ με το x, μεγιστοποιώντας τη μεταξύ τους γωνία

$$\langle g \circ f(\mathcal{M}(x)), g \circ f(x) \rangle,$$

στον Ευκλείδειο χώρο $\mathbb{R}^{128}$. Πιο τυπικά, αυτό επιτυγχάνεται μέσω του contrastive loss που ορίζεται ως εξής: Έστω το δείγμα $\{x_1, \ldots, x_N\}$ που αποτελείται από τα καθαρά φασματογράμματα από N διαφορετικά τραγούδια. Θεωρούμε το σύνολο $\{x_{N+1}, \ldots, x_{2N}\}$ που αντιστοιχεί στα παραμορφωμένα δείγματά τους. Δηλαδή, $x_{i+N} = \mathcal{M}(x_i), i = 1, \ldots, N$. Η ποσότητα που μετρά πόσο κοντά είναι το x_i στο x_j στον χώρο εμφύτευσης $\mathcal{S}^{d-1}$ δίνεται από την

$$\ell(i,j) = -\log \frac{e^{\langle g\circ f(x_i), g\circ f(x_j)\rangle/T}}{\sum_{k=1}^{2N} \mathbb{1}_{k\neq i} \cdot e^{\langle g\circ f(x_i), g\circ f(x_k)\rangle/T}}, \tag{1.1.20}$$

όπου $T > 0$ η υπερπαράμετρος θερμοκρασίας. Με βάση την (1.1.20) το contrastive loss δίνεται

μέσω της

$$\mathcal{L} = \frac{1}{2N} \sum_{k=1}^{N} [\ell(k, N+k) + \ell(N+k, k)]. \tag{1.1.21}$$

Έχοντας εκπαιδεύσει το δίκτυο με αυτόν τον τρόπο, στην παράγραφο 5.3 χρησιμοποιούμε το δίκτυο για να εξάγουμε τα fingerprints από τα 26016 κομμάτια που χρησιμοποιήσαμε και στην περίπτωση της dejavu και εξηγούμε πώς μπορούμε να φτιάξουμε τη βάση με τέτοιο τρόπο ώστε να είναι η εφικτή η γρήγορη ανάκτηση των αποτελεσμάτων. Η εξαγωγή των fingerprints γίνεται για κάθε ένα δευτερόλεπτο των τραγουδιών και με άλμα εξαγωγής που αντιστοιχεί σε 0.5 ms. Έτσι, για παράδεγμα, ένα κομμάτι διάρκειας 180 δευτερολέπτων μας δίνει 359 αποτυπώματα. Εν συνεχεία, για να μειώσουμε τη χωρητικότητα της βάσης, χρησιμοποιούμε τη μέθοδο του product quantization οδηγώντας μας σε μια αναπαράσταση που καταλαμβάνει μόλις 400 MB, σε αντίθεση με την dejavu των 6 GB. Όπως και στην περίπτωση της dejavu, χρησιμοποιούμε μια μη εξαντλητική μέθοδο αναζήτησης για τη γρήγορη ανάκτηση των αποτελεσμάτων. Οι τεχνικές αυτές παρουσιάζονται αναλυτικά στην πάραγραφο 5.3.

Στο Κεφάλαιο 6 παρουσιάζουμε τις λεπτομέρειες που αφορούν το πειραματικό μέρος της εργασίας. Πιο συγκεκριμένα, στην ενότητα 6.1 παρουσιάζουμε το σύνολο δεδομένων που αναπτύχθηκε για τις ανάγκες των πειραμάτων. Στην ενότητα 6.2 παρουσιάζουμε δύο διαφορετικά πειράματα που σχεδιάστηκαν για την αξιολόγηση και σύγκριση των δύο συστημάτων. Στην πρώτη περίπτωση εξετάζουμε την απόδοση των συστημάτων προσθέτοντας θόρυβο σε 17 τραγούδια με συγκεκριμένο SNR από το σύνολο $\{0, 5, 10, 15\}$ (σε dB). Επίσης, εξετάζουμε και πως η χρονική διάρκεια του ηχητικού αποσπάσματος προς αναγνώριση επηρεάζει την απόδοση των συστημάτων. Η μετρική που χρησιμοποιούμε για την αξιολόγηση σε αυτή την περίπτωση είναι το Top-1-hit-rate, μετρική με την οποία μας ενδιαφέρει η σωστή πρόβλεψη του τραγουδιού αλλά συγχρόνως και η χρονική πρόβλεψη σε σχέση με το πραγματικό κομμάτι με ένα όριο ανοχής αντίστοιχο των ±500 ms. Ο λόγος για τον οποίο διεξάγεται αυτό το πείραμα είναι για να μπορούμε να αξιολογήσουμε την ανοχή της κάθε προσέγγισης ως προς το SNR. Παρ′ όλα αυτά, σε ένα πραγματικό σενάριο, το ηχητικό σήμα υποκείται και σε μια σειρά από άλλες παραμορφώσεις. Γι′ αυτόν τον λόγο, στην παράγραφο 6.2.2 παρουσιάζουμε ένα πείραμα που προσομοιάζει καλύτερα αυτές τις συνθήκες. Πιο συγκεκριμένα, σε αυτό το πείραμα 15 τραγούδια παίζονται το ένα μετά το άλλο από τα ηχεία ενός φορητού υπολογιστή. Από ένα άλλο ηχείο παίζεται ένα ηχητικό απόσπασμα που περιέχει θόρυβο (ομιλίες σε εξωτερικό χώρο). Η διαφορά με το πρώτο πείραμα είναι ότι η καταγραφή γίνεται από το μικρόφωνο του φορητού υπολογιστή αντικατοπτρίζοντας με αυτόν τον τρόπο ένα πραγματικό σενάριο όπου οι χρήστες χρησιμοποιούν τις φορητές τους συσκευές για την ηχογράφηση των ηχητικών αποσπασμάτων προς αναγνώριση. Έτσι, με αυτόν τον τρόπο παράγουμε 3 διαφορετικά αρχεία (low.wav, mid.wav, high.wav) ηχογραφήσεων ανάλογα με την απόσταση του ηχείου θορύβου από το μικρόφωνο του φορητή υπολογιστή. Το αρχείο low.wav αντιστοιχεί στην μεγαλύτερη απόσταση (άρα και στον λιγότερο θόρυβο) ενώ το αρχείο high.wav στη μικρότερη απόσταση (περισσότερος θόρυβος).

Εν συνεχεία, στην ενότητα 6.3, χρησιμοποιώντας τη βιβλιοθήκη ανοικτού κώδικα dejavu[1] υλοποιούμε το σύστημα αναγνώρισης μουσικής που αξιοποιεί τις ιδέες του αλγορίθμου του

[1] https://github.com/worldveil/dejavu

Shazam. Αφού παρουσιάσουμε μερικές τεχνικές λεπτομέρειες που αφορούν τη βιβλιοθήκη στην ενότητα 6.3.1, στην ενότητα 6.3.2 παρουσιάζουμε τα αποτελέσματα αυτής της προσέγγισης στο πείραμα που ο θόρυβος προστίθεται με συγκεκριμένο SNR. Στον πίνακα 1.1 βλέπουμε τα αποτελέσματα αυτής της προσέγγισης για όλους τους συνδυασμούς SNR και διάρκειας του ηχητικού αποσπάσματος.

Table 1.1. *Απόδοση της dejavu.*

Metrics	**Query length (s)**	0 dB	5 dB	10 dB	15 dB
Accuracy		3.91%	5.91%	7.84 %	8.87 %
F1 score	1	6.86%	11.01%	14.12%	15.92%
Top-1 hit rate		**3.39%**	**4.72%**	**6.89%**	**7.30%**
Accuracy		28.06%	36.66%	42.45%	48.23&
F1 score	2	41.05%	52.94%	59.06%	64.61%
Top-1 hit rate		**24.27%**	**33.32%**	**38.48%**	**44.02%**
Accuracy		41.38%	60.70%	68.10%	73.78&
F1 score	3	57.36%	74.53%	80.50%	84.54%
Top-1 hit rate		**38.65%**	**56.82%**	**63.79%**	**69.32%**
Accuracy		52.06%	71.91%	81.30%	86.77&
F1 score	4	66.22%	82.83%	89.19%	92.79%
Top-1 hit rate		**48.32%**	**67.11%**	**75.84%**	**81.59%**
Accuracy		60.02%	79.83%	87.52%	91.60&
F1 score	5	73.31%	88.38%	93.23%	95.39%
Top-1 hit rate		**55.22%**	**75.15%**	**82.11%**	**86.55%**
Accuracy		82.85%	94.69%	98.07%	98.79&
F1 score	10	90.08%	97.25%	99.05%	99.40%
Top-1 hit rate		**77.78%**	**90.34%**	**93.48%**	**94.44%**

Στην ενότητα 6.4 συγκρίνουμε τις δύο μεθόδους. Πιο αναλυτικά, στην παράγραφο 6.4.1 επικεντρωνόμαστε στην πρόσεγγιση που αξιοποιεί τις ιδέες της βαθιάς μάθησης. Αφού παρουσιάσουμε μερικές τεχνικές λεπτομέρειες σχετικά με την υλοποίηση της μεθόδου, στη συνέχεια εξετάζουμε την απόδοση του συστήματος στο πείραμα με το σταθερό SNR θορύβου. Στον πίνακα 1.2 φαίνονται τα αποτελέσματα αυτού του πειράματος. Συγκρίνοντας τους πίνακες 1.1, 1.2 βλέπουμε ότι η πρόσεγγιση της βαθιάς μάθησης υπερτερεί της κλασικής μεθόδου για μικρά ηχητικά αποσπάσματα ($<$ 10 sec), κάνοντάς την έτσι κατάλληλη για εφαρμογές με απαιτήσεις πραγματικού χρόνου. Στην παράγραφο 6.4.2 παρουσιάζουμε και τα αποτελέσματα του πειράματος όπου η ηχογράφηση γίνεται από το μικρόφωνο φορητού υπολογιστή. Επίσης, εξετάζουμε και πόσο οι παραμορφώσεις που αφορούν το «κλιπάρισμα» των χαμηλών/υψηλών συχνοτήτων βοηθούν στην ανοχή του μοντέλου ως προς τον θόρυβο. Τα αποτελέσματα γι' αυτό το πείραμα φαίνονται στον πίνακα 1.3. Τέλος, στο κεφάλαιο 7 κλείνουμε αναφέροντας κάποια συμπεράσματα αλλά και πιθανές μελλοντικές κατευθύνσεις έρευνας για το πρόβλημα αναπάπτυξης αποτελεσματικών συστημάτων αναγνώρισης μουσικής.

Table 1.2. *Απόδοση του deep fingerprinting.*

Metrics	**Query length (s)**	0 dB	5 dB	10 dB	15 dB
Accuracy		56.80%	81.19%	91.73 %	94.78 %
F1 score	1	70.79%	88.73%	95.10%	96.84%
Top-1 hit rate		**45.30%**	**66.52%**	**78.33%**	**84.48%**
Accuracy		78.11%	92.54 %	97.56 %	98.61 &
F1 score	2	86.80 %	95.54 %	98.43 %	99.06 %
Top-1 hit rate		**63.77 %**	**78.35 %**	**87.09 %**	**89.63 %**
Accuracy		83.41 %	96.12 %	98.49 %	99.07 &
F1 score	3	90.25 %	97.55 %	99.04 %	99.33 %
Top-1 hit rate		**69.25 %**	**83.41 %**	**89.44 %**	**92.82 %**
Accuracy		87.73 %	97.32 %	98.75 %	98.95 &
F1 score	4	92.92 %	98.39 %	99.20 %	99.29 %
Top-1 hit rate		**73.44 %**	**85.52 %**	**91.75 %**	**92.62 %**
Accuracy		89.32 %	97.48 %	98.56 %	99.16 &
F1 score	5	93.58 %	98.35 %	99.07 %	99.39 %
Top-1 hit rate		**75.99 %**	**85.52 %**	**89.68 %**	**93.88 %**
Accuracy		95.65 %	98.07 %	98.55 %	98.31 &
F1 score	10	97.27 %	98.86 %	99.06 %	98.97 %
Top-1 hit rate		**78.50 %**	**84.30 4%**	**88.89 %**	**92.51 %**

Table 1.3. *Επίδοση των συστημάτων αναγνώρισης σε καταγραφή από μικρόφωνο.*

Metrics	**Query length (s)**	low	mid	high
Deep Audio		**85.03%**	**62.65%**	**41.92%**
Deep Audio (No Filters)	2	67.22%	49.02%	19.47%
Dejavu		12.63%	6.50%	10.73%
Deep Audio		**92.66%**	**80.06%**	**62.52%**
Deep Audio (No Filters)	5	80.38%	65.39%	31.10%
Dejavu		59.21%	40.94%	47.24%
Deep Audio		**94.52%**	**90.32%**	**74.19%**
Deep Audio (No Filters)	10	84.19%	78.39%	41.29%
Dejavu		83.60%	70.98%	71.29%
Deep Audio		**97.56%**	**90.24%**	80.98%
Deep Audio (No Filters)	15	86.34%	80.98%	48.29%
Dejavu		93.33%	77.14%	**83.33%**

1.2 English

In this opening chapter, we offer a brief overview of the present dissertation. The aim of this chapter is to provide a first look into the contents of the thesis, by presenting the basic structure, the goals and the outcomes of each chapter.

In Chapter 2, we introduce the Fourier Transform, which is the fundamental tool for studying and processing signals. We delve into its basic properties. Specifically, our exploration begins with the case of periodic functions (or periodic signals) $f : \mathbb{R} \to \mathbb{R}$. These are functions for which there exists $T \in \mathbb{R}$ such that

$$f(x + T) = f(x),$$

for every $x \in \mathbb{R}$. This framework includes also the functions of the form $\sin(\pi(nt - \phi_n))$, called *sinusoids*, which are the building blocks of the theory. The number n represents the frequency of the signal, measured in Hertz, while the number ϕ_n corresponds to the phase of the signal. The significance of functions of this form lies in the fact that under certain conditions, every periodic function $f : \mathbb{R} \to \mathbb{R}$ can be approximated by a, possibly infinite, sum

$$\sum_{n=1}^{\infty} A_n \sin(2\pi(nt - \phi_n)), \tag{1.2.1}$$

of such functions. Using the representation in Eq. (1.2.1), we can discern the contribution of each frequency n to the periodic signal. Thus, in Section 2.1, we present with proofs (where feasible) how and under which conditions we can achieve this representation. In Section 2.2, we extend the theory to signals $f : \mathbb{R} \to \mathbb{R}$ that are not necessarily periodic. This generalization is necessary because most audio signals encountered in practice are non-periodic. Therefore, it becomes essential to examine how we can deduce the frequency behavior of a signal from its waveform. In this case, the key tool is the Fourier Transform

$$\mathcal{F}f(s) = \int_{-\infty}^{\infty} f(t)e^{-2\pi ist}\, dt. \tag{1.2.2}$$

Hence, in this section, we demonstrate how we can extend the concept of the Fourier Transform to non-periodic functions. Additionally, we introduce the inverse Fourier Transform

$$\mathcal{F}^{-1}g(t) = \int_{-\infty}^{\infty} g(s)e^{2\pi ist}\, ds, \tag{1.2.3}$$

which acts as the inverse mapping of Eq. (1.2.2) and is related through the relationships

$$\mathcal{F}\left(\mathcal{F}^{-1}f\right)(t) = f(t), \quad \mathcal{F}^{-1}\left(\mathcal{F}f\right)(s) = f(s). \tag{1.2.4}$$

The utility of the relationships in Eq. (1.1.4) becomes evident when combined with the convolution theorem, which states that the Fourier Transform resulting from the convolution

$$(f * g)(t) = \int_{-\infty}^{\infty} f(t - x)g(x)\, dx, \tag{1.2.5}$$

of two signals f, g is equal to the product

$$\mathcal{F}(f * g)(s) = \mathcal{F}f(s) \cdot \mathcal{F}g(s), \tag{1.2.6}$$

of the Fourier Transforms $\mathcal{F}f, \mathcal{F}g$ of f, g. These relationships come into play when we wish to apply a transformation to a given signal $f : \mathbb{R} \to \mathbb{R}$. For instance, if we want to "filter out" frequencies in the signal above a certain threshold $v_C > 0$, we can achieve this by multiplying the Fourier Transform $\mathcal{F}f(s)$ with the function

$$H(s) = \begin{cases} 1, & |s| < v_c \\ 0, & |s| \geq v_c \end{cases}.$$

Knowing that $h(t) = \mathcal{F}^{-1}H(t) = 2v_C\text{sinc}(2v_C t)$, and utilizing the inverse transform, we can determine the formula $y(t) = (f * h)(t)$ for the desired signal. In this section, we also present various other forms of transformations widely applied in signal processing. This theory helps us better understand how these transformations are applied, which will be extensively employed in training a neural network to extract information from short audio excerpts of different songs. In Sections 2.3% 2.4, we focus on the discrete case of the Fourier Transform which is utilized by computer machines processing audio signals. This occurs because a computer can only handle a finite number of values. The process of transitioning from the continuous case of the transform in Eq. (1.1.2) to the discrete case is as follows: Initially, the analog signal $f : \mathbb{R} \to \mathbb{R}$ is transformed into a discrete signal $x : \mathbb{Z} \to \mathbb{R}$ through

$$x(n) = f(n \cdot T), \quad n \in \mathbb{Z},$$

via sampling. The value $T > 0$ corresponds to the sampling period, while its inverse $F_s = 1/T$ represents the sampling rate measured in Hertz. This process is graphically represented in Figure 1.5.

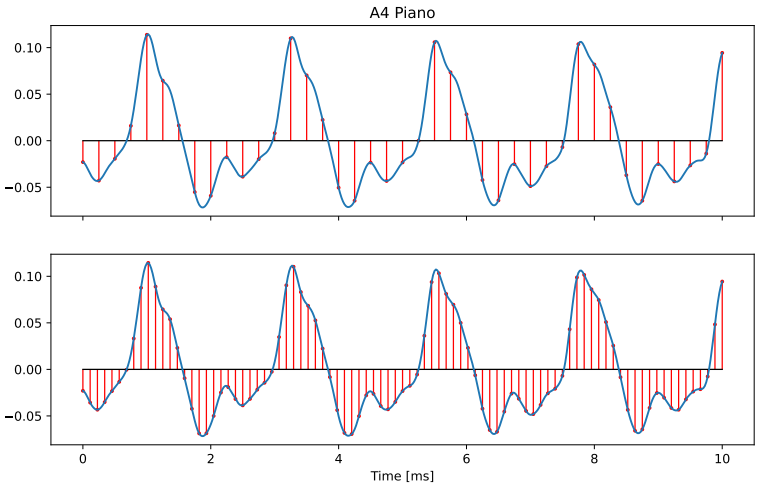

Figure 1.5. *Equidistant sampling of Piano A4 with sampling rates 4KHz (top) and 8KHz (bottom).*

Next, with the aid of Riemann sums, the transformation in Eq. (1.2.2) takes the form

$$\int_{-\infty}^{\infty} f(t)e^{-2\pi ist}\,dt \approx T \cdot \sum_{n=-\infty}^{\infty} f(nT)e^{-2\pi isnT}. \tag{1.2.7}$$

In general, for a discrete signal $x : \mathbb{Z} \to \mathbb{R}$, the Fourier Transform is given by

$$\hat{x}(s) = \sum_{n=-\infty}^{\infty} x(n)e^{-2\pi isn}. \tag{1.2.8}$$

In this case, the frequency $\hat{x}(s)$ is related to the frequency of the analog signal through

$$\hat{x}(s) = \frac{1}{T}F\left(\frac{s}{T}\right), \tag{1.2.9}$$

where F is the Fourier Transform of the analog signal $f : \mathbb{R} \to \mathbb{R}$, with $x(n) = f(nT)$. However, the issue with Eq.(1.2.8) is that the sum involved has an infinite number of terms, and the frequency $s \in \mathbb{R}$ spans all real numbers. Therefore, in practice, each signal $x : \mathbb{Z} \to \mathbb{R}$ is of finite duration, and it is represented by N values $x(0), \ldots, x(N-1)$ on the computer. Thus, on a computational level, Eq.(1.2.8) is replaced by

$$X(k) = \hat{x}(k/N) = \sum_{n=0}^{N-1} x(n)e^{-2\pi ikn/N}, \tag{1.2.10}$$

for $k = 0, \ldots, N-1$. Based on Eq. (1.1.9), the relationship between the k^{th} Fourier coefficient $X(k)$ and the frequency of the analog signal $f : \mathbb{R} \to \mathbb{R}$ is given by

$$X(k) = \hat{x}(k/N) \approx \frac{1}{T}F\left(\frac{k}{NT}\right). \tag{1.2.11}$$

The relation in Eq.(1.2.11) provides us with information about the contribution of frequencies k/NT over the entire duration of the signal. Thus, in this manner, the temporal occurrence of these frequencies remains hidden. For this reason, the concept of the short-time Fourier Transform is introduced, where the transformation of Eq.(1.2.11) is applied to small windows of the signal. Specifically, for a given window length N, corresponding to the number of samples, and a hop length H, the transformation is applied as

$$\mathcal{X}(m,k) = \sum_{n=0}^{N-1} x(n+mH)w(n)e^{-2\pi ink/N}, \tag{1.2.12}$$

for $m \in \mathbb{Z}$ and $k = 0, \ldots, N-1$. In this way, we arrive at a two-dimensional representation $\mathcal{X}(m,k)$ of an analog signal $f : \mathbb{R} \to \mathbb{R}$, which provides us with information about the frequencies in the signal as well as the time instants at which they appear. In other words, the quantity $\mathcal{X}(m,k)$ encompasses information about the magnitude and phase of the frequency k/NT at the time instant mH/F_s, where F_s is the sampling rate of the analog signal $f : \mathbb{R} \to \mathbb{R}$. The concept of the spectrogram is defined through Eq.(1.2.12), which is one of the most commonly used features in audio signal processing. The spectrogram is given by the magnitude of the complex numbers $|\mathcal{X}(m,k)|^2$, allowing us to discern the energy of frequencies k/NT at time instants mH/F_s. Figure1.6 displays

the waveform of the signal resulting from pressing the A4 key on a piano (after sampling) and its corresponding spectrogram. As shown in the figure, most of the energy is concentrated at the frequency of 440 Hz, which is the frequency of this note.

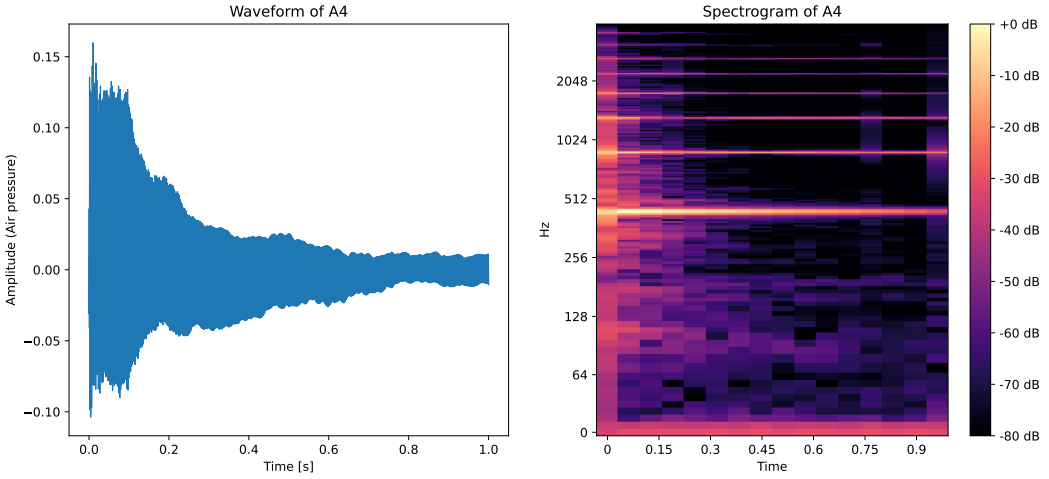

Figure 1.6. *The waveform of A4 key along with its spectrogram representation. The A4 played on piano contains a high degree of the frequency component corresponding to* 440 *Hz. The other contributions visible in the spectrogram correspond to the harmonics of this fundamental frequency, which are integer multiples of* 440 *Hz (i.e.,* 880 *Hz,* 1320 *Hz, etc.).*

Having developed the necessary tools around the Fourier Transform, in Chapter 3, we dwelve into the theory of audio signal processing. In Section 3.1, we present various ways in which musical information can be represented. In our case, the most important representation is the one that describes an audio signal as a function $f : [0, \infty) \to \mathbb{R}$, which, for a given time instant $t > 0$, measures the air pressure at a given point. In this way, by viewing audio signals as functions $f : \mathbb{R} \to \mathbb{R}$, we can leverage the theory from Chapter 2. In Section 3.3, we present the most important audio features extracted using the discrete Fourier Transform in audio signal processing. One of the most critical representations on which we rely for the development of deep learning models is the mel-spectrogram, presented in 3.3.3. The difference from the spectrogram is that this representation converts the frequency axis into a logarithmic scale expressed through the equation

$$m = 2595 \cdot \log_{10}\left(1 + \frac{f}{700}\right). \tag{1.2.13}$$

In this paragraph, we cover in detail all the processing steps required to transition from the spectrogram to the representation of the mel-spectrogram. The reason behind this frequency axis conversion is because it has been shown, through psychoacoustic experiments, that humans perceive frequencies logarithmically rather than linearly. This also explains why piano notes that are exactly an octave apart have the same tonal "color." In Figure 1.7, you can observe the mel-spectrogram corresponding to the piano notes C2, C4, G6, and A6.

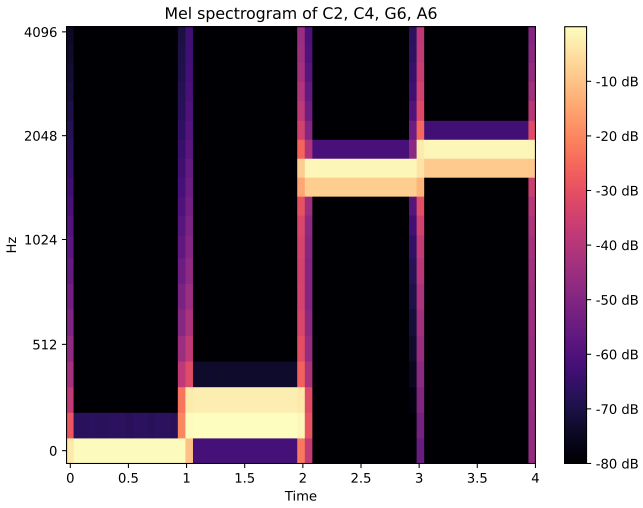

Figure 1.7. *Mel spectrogram of the notes C2, C4, G6, and A6 with 20 frequency bins.*

In Section 3.4, we present and describe a series of transformations used in audio signal processing, such as adding background noise, simulating reverbaration effects, and clipping specific frequencies. The reason we focus on these transformations and strive to understand them is that such distortions are often encountered in audio signals recorded by portable devices. Therefore, if we aim to develop an efficient music recognition system that operates under real-world conditions, we need to develop tools that counteract these distortions. This approach will also be applied to training the deep learning model in Chapter 5. In the last paragraph of this chapter, we introduce the problem of music recognition (song identification). After presenting the core problem and the challenges that must be addressed when designing a music recognition system, we provide a detailed overview of the method used by the Shazam algorithm for song identification. The idea was introduced by Wang et al. [1] and is based on extracting music information that is as robust as possible against the distortions that can occur in a real scenario. The concept is to focus on the local maxima of the spectrogram of a small audio fragment, with the hope that these local values will be preserved even in the presence of significant distortions. These local maxima are known as audio fingerprints and constitute the main ingredient of this method. In Figure 1.8, you can see the spectrogram and the local maxima in a segment of Vivaldi's "Four Seasons - Spring" without distortions. As it is evident from the figure, while the spectrograms exhibit significant variation, most of the local maxima appear in the same positions. The first step in extracting this representation is to find the local maxima. To do this, we select two constants k_0 and τ_0, both greater than zero, which define the region $[-k_0, k_0] \times [-\tau_0, \tau_0]$. Consequently, a point (k^*, m^*) in the spectrogram is a local maximum if and only if

$$\left|\mathcal{X}(k^*, m^*)\right| > \left|\mathcal{X}(k, m)\right|, \tag{1.2.14}$$

for all $(k, m) \in [k^* - k_0, k^* + k_0] \times [m^* - \tau_0, m^* + \tau_0]$. At an abstract level, we can think of the basis at this stage as having placed all the spectrograms corresponding to different segments one after the other in a large unified spectrogram.

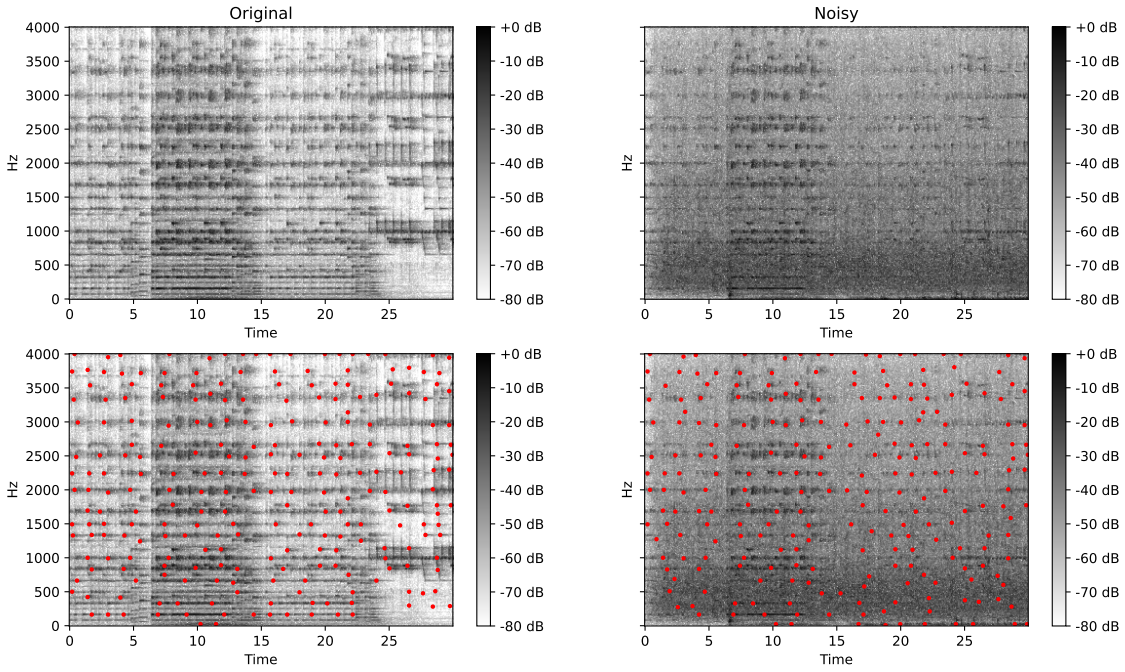

Figure 1.8. *Spectrograms corresponding to 30-sec audio fragment from The Four Seasons - Spring of Vivaldi. On the left is the clean audio, and on the left is the distorted signal with background noise added (SNR = 0 in dB).*

In this representation, we can maintain the temporal frames that mark the end of each segment and the beginning of the next one. Thus, following the notation of [2], if we denote by $\mathcal{D}$ the set of segments and by $C(\mathcal{D})$ the set of points (k, m) corresponding to the local maxima of this unified spectrogram, we can describe a naive approach for recognizing an audio fragment Q , given $\mathcal{D}$ and $C(\mathcal{D})$. In the first step, we calculate the local maxima $C(Q)$ corresponding to the spectrogram of the query example. A straightforward approach to find the database segment with the highest similarity is to consider the temporal shifts

$$m + C(Q) = \{(k, m + n) : (k, n) \in C(Q)\}. \tag{1.2.15}$$

Then, the temporal segment m^* of the database can be determined using

$$m^* = \arg\max_{m \in \mathbb{N}} \Delta_Q(m) = \arg\max_{m \in \mathbb{N}} \frac{|(m + C(Q)) \cap C(\mathcal{D})|}{|C(Q|}, \tag{1.2.16}$$

where $|C(Q)|$ denotes the cardinality of the set $C(Q)$, and likewise for the set $(m+C(Q))\cap C(\mathcal{D})$. In other words, Eq. (1.2.16) tells us that the database segment with the highest similarity to the audio snippet is the one that matches the most local maxima of the corresponding spectrogram. Utilizing the second structure, with the optimal temporal point m^*, we can find the segment corresponding to this point m^* and consequently retrieve information about the artist of the segment. However, this approach is not efficient for real-time applications, as the search time increases linearly with the size of the database, making information retrieval slow for databases containing thousands of

segments. Thus, in practice, alternative approaches are adopted that use non-exhaustive search techniques, which reduce the search space by examining only specific "parts" of the database rather than the entire database. In the case of the Shazam algorithm, the non-exhaustive technique is based on inverted lists. In paragraph 3.5.3, we present this technique as well as the method introduced by Wang et al. to further enhance the system's performance.

In Chapter 4, we introduce deep learning. The purpose of this chapter is to present the fundamental concepts that will be used in the subsequent chapters. In paragraph 4.1.1, we introduce the basic architecture of neural networks, the feed-forward neural networks. In an abstract sense, each such network is a function $f : \mathbb{R}^K \to \mathbb{R}^N$. Its general form can be expressed as a composition of functions:

$$f = f^{(n)} \circ f^{(n-1)} \cdots \circ f^{(1)}, \tag{1.2.17}$$

where each $f^{(i)} = \sigma_i \circ T_i$, with $T_i : \mathbb{R}^{K_i} \to \mathbb{R}^{K_{i+1}}$ being a linear transformation and $\sigma_i : \mathbb{R} \to \mathbb{R}$ a nonlinear activation function. Since each function $T_i : \mathbb{R}^{K_i} \to \mathbb{R}^{K_{i+1}}$ is linear, it can be written as $T_i(x) = W_i \cdot x$, where $W_i \in \mathbb{R}^{K_{i+1} \times K_i}$ is a matrix. The values of matrices W_i are represented by $\boldsymbol{\theta}$ and constitute the model's parameters. Thus, to denote the dependency on the network's parameters, we typically write $f(\cdot; \boldsymbol{\theta})$ instead of $f(\cdot)$. In Section 4.2, after presenting the various problem types in deep learning, such as supervised, unsupervised, and semi-supervised learning, we introduce the basic theoretical framework underlying the training of neural networks. The goal in each deep learning problem is to find the suitable model parameters $\boldsymbol{\theta}$ that minimize a cost function J. For instance, in supervised learning problems, we are given a dataset $(x_1, y_1), \ldots, (x_n, y_n)$, where each sample x_i belongs to one of K categories. The objective is to find the parameters $\boldsymbol{\theta}^*$ such that $f(x_i; \boldsymbol{\theta}) = y_i$. In this case, the chosen function J is the cross-entropy, defined as follows:

$$J(f(x), y_x) = -\sum_{i=1}^{K} \log \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \cdot \mathbb{1}_{i=y_x}, \tag{1.2.18}$$

where $f(x) = (z_1, \ldots, z_n)$ and $y_x \in 1, \ldots, K$. The search for appropriate parameters is carried out by gradient-based algorithms. The central idea of these algorithms is to iteratively update the weights in the opposite direction of the gradient $\nabla_{\boldsymbol{\theta}} J$. The fundamental algorithm in this category is the stochastic gradient descent (SGD). In this algorithm, the samples (x_i, y_i) are examined one by one, and for each such pair, the parameters are updated according to the equation:

$$\boldsymbol{\theta}^{(i+1)} = \boldsymbol{\theta}^{(i)} - \eta \cdot \nabla_{\boldsymbol{\theta}} J\left(f(x^{(i)}; \boldsymbol{\theta}^{(i)}), y^{(i)}\right). \tag{1.2.19}$$

In practice, variations of stochastic gradient descent are often used, achieving better results. In Section 4.3, we examine some of these variations, presenting their advantages and disadvantages. In section 4.4, we introduce the basic structure of Convolutional Neural Networks (CNNs), extensively used in processing data originate from images and audio signals. The fundamental operation characterizing CNNs is convolution, which is presented in paragraph 4.4.2. Through convolution, these networks are capable of extracting useful local information from two-dimensional objects, such as images, spectrograms, etc., while simultaneously reducing their dimensions. In our case, we rely on this architecture to extract a compact representation from the audio fragments of songs, which will be used for the recognition system.

In Chapter 5, we present the music recognition system based on the deep learning concepts. The main difference compared to the approach followed by Shazam is that in this case, we train a neural network that takes over the feature extraction from the songs. This way, the focus shifts from finding sophisticated feature extraction algorithms to finding effective training algorithms and techniques. In this case, the network training is done by utilizing the contrastive learning framework. The idea behind this technique is to map the distorted audio fragments as close as possible to the actual audio fragments in a lower-dimensional Euclidean space compared to the input data space. To achieve this, a prerequisite is to select suitable distortion techniques that correspond to realistic scenarios. After presenting the architecture of the convolutional network in paragraph 5.2.1, we introduce the contrastive learning method in paragraph 5.2.2. The choice of data augmentation methods is crucial for the training of the network, including adding noise with different SNR, reverberation, time shifting, and low/high-frequency clipping. The combination of these distortions leads to the development of a model capable of making highly accurate predictions even in cases where the audio is captured by microphones of portable devices. Mathematically, the neural network consists of two parts: (a) the encoder $f : \mathbb{R}^{256\times32} \to \mathbb{R}^{1024}$, which is essentially a convolutional network that operates on mel-spectrograms of dimensions $\mathbb{R}^{256\times32}$, mapping them to vectors of dimensions 1024, and (b) the projection function $g : \mathbb{R}^{1024} \to \mathcal{S}^{d-1}$, which is a feedforward neural network that maps input vectors onto the unit sphere

$$\mathcal{S}^{d-1} = \{z \in \mathbb{R}^d : \|z\|_2 = 1\},$$

in $\mathbb{R}^d$. In our case, the dimensions $\mathbb{R}^{256\times32}$ resulting from applying the short-time Fourier Transform correspond to 1 second of audio segment. Also, the projection dimension is $d = 128$. Thus, in broad terms, the neural network $g \circ f$ maps audio fragments of one second into vectors of 128 dimensions. Analogous to the classical fingerprinting approach, each vector $g \circ f(x)$ is called a fingerprint of $x \in \mathbb{R}^{256\times32}$. During training, the goal is to bring the distorted sample $\mathcal{M}(x)$ as close as possible to x, maximizing their cosine similarity

$$\ell(i,j) = -\log \frac{e^{\langle g\circ f(x_i), g\circ f(x_j)\rangle/T}}{\sum_{k=1}^{2N} \mathbb{1}_{k\neq i} \cdot e^{\langle g\circ f(x_i), g\circ f(x_k)\rangle/T}}, \tag{1.2.20}$$

in the Euclidean space $\mathbb{R}^{128}$. This is achieved through the contrastive loss, which is defined by comparing each clean sample to its corresponding distorted version. After training the network this way, in Section 5.3, we use the network to extract fingerprints from the 26016 songs we used and explain how to build the database allowing for fast and efficient retrievals. The extraction is done for every one-second segment of the songs with a step size corresponding to 0.5 milliseconds. To reduce the capacity of the database, the product quantization method is employed, resulting in a representation occupying only 400 MB, in contrast to dejavu's 6 GB. Similarly to the dejavu case, a non-exhaustive search method is utilized for fast result retrieval. These techniques are discussed in detail in Section 5.3.

In Chapter 6, we present the details related to the experimental part of the thesis. More specifically, in Section 6.1, we introduce the dataset developed for the experiments' needs. In Section 6.2, we present two different experiments designed for the evaluation and comparison of the two systems. In the first case, we examine the performance of the systems by adding noise to 17 songs with

specific SNR values from the set $\{0, 5, 10, 15\}$ (in dB). Additionally, we investigate how the duration of the audio fragment for recognition affects the systems' performance. The metric we use for evaluation in this case is the Top-1-hit-rate, a metric that measures both correct song prediction and temporal alignment with respect to the original song allowing a tolerance of ±500 ms. This experiment is conducted to assess the tolerance of each approach with respect to SNR. However, in a real scenario, the audio signal is subject to various other distortions. For this reason, in paragraph 6.2.2, we present an experiment that better simulates these conditions. Specifically, in this experiment, 15 songs are played sequentially from the speakers of a laptop. From another speaker, an audio excerpt containing noise (speech in an outdoor environment) is played. The difference from the first experiment is that the recording is done through the laptop's microphone, thus mimicking a real scenario where users use their portable devices to record audio fragments for recognition. In this way, we generate three different audio files (low.wav, mid.wav, high.wav) based on the distance of the noise source from the laptop's microphone, with 'low.wav' corresponding to the farthest distance (thus less noise) and 'high.wav' to the closest distance (more noise).

Subsequently, in Section 6.3, using the open-source library dejavu[2], we implement the music recognition system that incorporates the ideas of the Shazam algorithm. After presenting some technical details regarding the library in Section 6.3.1, in Section 6.3.2, we present the results of this approach in the experiment where noise is added with specific SNR. In Table 1.1, we see the results of this approach for all combinations of SNR and audio query duration.

Table 1.4. *Performance of dejavu.*

Metrics	**Query length (s)**	0 dB	5 dB	10 dB	15 dB
Accuracy	1	3.91%	5.91%	7.84 %	8.87 %
F1 score		6.86%	11.01%	14.12%	15.92%
Top-1 hit rate		**3.39%**	**4.72%**	**6.89%**	**7.30%**
Accuracy	2	28.06%	36.66%	42.45%	48.23&
F1 score		41.05%	52.94%	59.06%	64.61%
Top-1 hit rate		**24.27%**	**33.32%**	**38.48%**	**44.02%**
Accuracy	3	41.38%	60.70%	68.10%	73.78&
F1 score		57.36%	74.53%	80.50%	84.54%
Top-1 hit rate		**38.65%**	**56.82%**	**63.79%**	**69.32%**
Accuracy	4	52.06%	71.91%	81.30%	86.77&
F1 score		66.22%	82.83%	89.19%	92.79%
Top-1 hit rate		**48.32%**	**67.11%**	**75.84%**	**81.59%**
Accuracy	5	60.02%	79.83%	87.52%	91.60&
F1 score		73.31%	88.38%	93.23%	95.39%
Top-1 hit rate		**55.22%**	**75.15%**	**82.11%**	**86.55%**
Accuracy	10	82.85%	94.69%	98.07%	98.79&
F1 score		90.08%	97.25%	99.05%	99.40%
Top-1 hit rate		**77.78%**	**90.34%**	**93.48%**	**94.44%**

[2] https://github.com/worldveil/dejavu

In Section 6.4, we compare the two methods. More specifically, in paragraph 6.4.1, we focus on the approach that utilizes deep learning concepts. After presenting some technical details regarding the implementation of this method, we then examine the system's performance in the experiment with constant SNR noise. The results of this experiment are shown in Table 1.2. Comparing Tables 1.4 and 1.5, we observe that the deep learning approach outperforms the classical method for short audio excerpts (< 10 seconds), making it suitable for real-time applications. In paragraph 6.4.2, we present the results of the experiment where the recording is done through the laptop's microphone. Additionally, we investigate how distortions related to 'clipping' of low/high frequencies contribute to the model's tolerance to noise. The results for this experiment are shown in Table 1.6. Finally, in Chapter 7, we conclude by summarizing some findings and suggesting potential future research directions for the problem of developing effective music recognition systems.

Table 1.5. *Performance of Deep Audio Fingerprinting System.*

Metrics	**Query length (s)**	0 dB	5 dB	10 dB	15 dB
Accuracy		56.80%	81.19%	91.73 %	94.78 %
F1 score	1	70.79%	88.73%	95.10%	96.84%
Top-1 hit rate		**45.30%**	**66.52%**	**78.33%**	**84.48%**
Accuracy		78.11%	92.54 %	97.56 %	98.61 &
F1 score	2	86.80 %	95.54 %	98.43 %	99.06 %
Top-1 hit rate		**63.77 %**	**78.35 %**	**87.09 %**	**89.63 %**
Accuracy		83.41 %	96.12 %	98.49 %	99.07 &
F1 score	3	90.25 %	97.55 %	99.04 %	99.33 %
Top-1 hit rate		**69.25 %**	**83.41 %**	**89.44 %**	**92.82 %**
Accuracy		87.73 %	97.32 %	98.75 %	98.95 &
F1 score	4	92.92 %	98.39 %	99.20 %	99.29 %
Top-1 hit rate		**73.44 %**	**85.52 %**	**91.75 %**	**92.62 %**
Accuracy		89.32 %	97.48 %	98.56 %	99.16 &
F1 score	5	93.58 %	98.35 %	99.07 %	99.39 %
Top-1 hit rate		**75.99 %**	**85.52 %**	**89.68 %**	**93.88 %**
Accuracy		95.65 %	98.07 %	98.55 %	98.31 &
F1 score	10	97.27 %	98.86 %	99.06 %	98.97 %
Top-1 hit rate		**78.50 %**	**84.30 4%**	**88.89 %**	**92.51 %**

Table 1.6. *Performance (Accuracy) of the Music Recognition Systems on Microphone Experiment.*

Metrics	**Query length (s)**	low	mid	high
Deep Audio	2	**85.03%**	**62.65%**	**41.92%**
Deep Audio (No Filters)		67.22%	49.02%	19.47%
Dejavu		12.63%	6.50%	10.73%
Deep Audio	5	**92.66%**	**80.06%**	**62.52%**
Deep Audio (No Filters)		80.38%	65.39%	31.10%
Dejavu		59.21%	40.94%	47.24%
Deep Audio	10	**94.52%**	**90.32%**	**74.19%**
Deep Audio (No Filters)		84.19%	78.39%	41.29%
Dejavu		83.60%	70.98%	71.29%
Deep Audio	15	**97.56%**	**90.24%**	80.98%
Deep Audio (No Filters)		86.34%	80.98%	48.29%
Dejavu		93.33%	77.14%	**83.33%**

Chapter 2

Elements of Fourier Analysis

The purpose of this chapter is to develop all the necessary theoretical tools used in audio signal processing. We introduce one of the most important concepts, the Fourier Transform and develop some basic properties and theorems. Much of the content in this chapter is based on the mathematical area of Fourier analysis. However, it should be noted that the exposition in this chapter is not meant to cover all aspects and concepts of the theory in depth, but rather to serve as a building block for developing and explaining the features used in audio signal processing in Chapter 3. For more comprehensive expositions of the theory, readers may refer to [3, 4, 5].

As we will see in Chapter 3, an audio signal can be represented as a real function of one independent variable. The horizontal axis corresponds to the time elapsed from the first pulse of the audio signal, while the vertical axis measures the difference of air pressure from its average value at a certain point. Therefore, an audio signal can be represented by a *pressure-time plot*, also referred to as the *waveform* of the sound. Figure 2.1 shows a waveform representation of the first 10 seconds from the song "Someone Like you" by Adele[1].

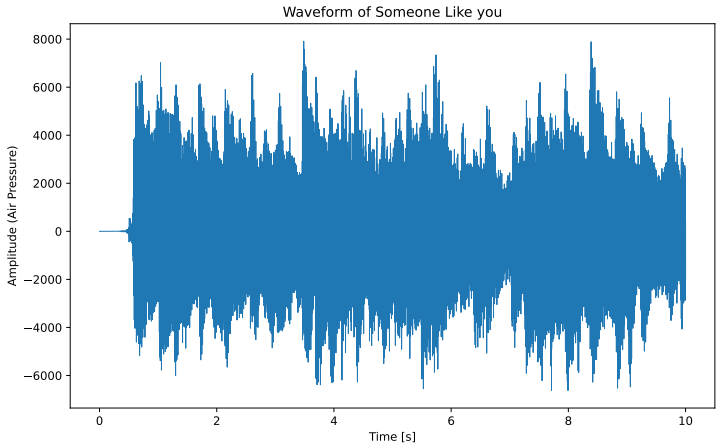

Figure 2.1. *The Waveform of the first 10 seconds of Someone Like You by Adele.*

By looking at the waveform in Figure 2.1 we can see that some peeks are arriving at equally spaced time intervals. As we shall shortly see, the number of repetitions of a certain peek at some fixed

[1] You can hear in: https://www.youtube.com/watch?v=z7GCiVTlv04.

time interval constitutes an important feature of a function, and in particular, of an audio signal. The tool that is used to determine the "energy" presence of each of these repetitions is the Fourier Transform. In the next section, we start by developing the theory for periodic functions, and we extend these ideas to the non-periodic case as well.

2.1 Fourier Series

We say that a function $f : \mathbb{R} \to \mathbb{R}$ is *periodic* if there exists $T \in \mathbb{R}$ such that

$$f(x + T) = f(x), \tag{2.1}$$

for all $x \in \mathbb{R}$. Any T satisfying (2.1) is a *period* of f. The smallest possible number for which (2.1) is satisfied is called the *fundamental period* of the function f. The reciprocal of the fundamental period is the *fundamental frequency* and it is measured in Hertz (Hz). The fundamental frequency is the number of oscillations in a time interval of length T. The most simple, and perhaps the most important family of periodic functions are the sinusoids. Mathematically, a sinusoid can be described by a function $f : \mathbb{R} \to \mathbb{R}$ such that $f(t) = A \cdot \sin(2\pi(ft - \phi))$ for $t \in \mathbb{R}$. The parameter A corresponds to the *amplitude*, the parameter f to the *frequency*, and the parameter ϕ to the *phase* (measured in normalized radians with 1 corresponding to an angle of 360°). Figure 2.2 shows the waveform of the sinusoids $2\sin(2\pi \cdot 5t)$, $1.2\sin(2\pi \cdot 2t)$ as well as the superposition $2\sin(2\pi \cdot 5t) + 1.2\sin(2\pi \cdot 2t)$.

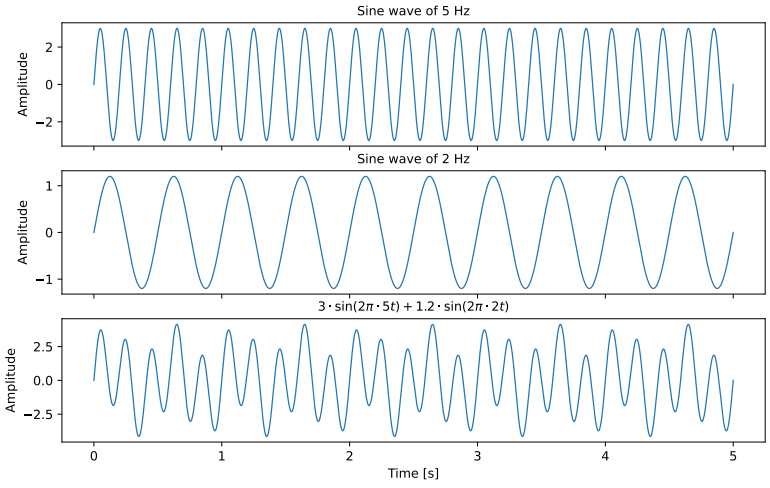

Figure 2.2. *Sinusoids of 5 Hz and 2 Hz.*

2.1.1 The Building Blocks

The sinusoids are the building blocks of Fourier analysis. The idea is to describe any periodic function $f : \mathbb{R} \to \mathbb{R}$ of period 1[2] as a superposition of sinusoids of the form

$$\sum_{n=1}^{N} A_n \sin(2\pi(nt - \phi_n)). \tag{2.1.1}$$

To accomplish this, one needs to define a measure of similarity between functions. This measure will be used to compare the periodic function $f : \mathbb{R} \to \mathbb{R}$ with a sinusoid $\sin(\pi(nt - \phi_n))$ of fixed frequency and phase. Higher values of the measure would imply higher similarity with the sinusoid $\sin(\pi(nt - \phi_n))$. The individual terms in (2.1.1) are called *harmonics*, terminology that comes from the mathematical representation of musical pitch (see chapter 3). This form of a general trigonometric sum has the advantage of displaying explicitly the amplitude and phase of each harmonic, but in theory, for calculation purposes as well as for the ability of generalization to complex functions an alternative form is used by using the trigonometric identity

$$\sin(\alpha - \beta) = \sin\alpha\cos\beta - \cos\alpha\sin\beta, \tag{2.1.2}$$

and the complex exponentials

$$e^{int} = \cos(nt) + i\sin(nt). \tag{2.1.3}$$

Letting $u_n = 2\pi\phi_n$ we can write (2.1.1) as $\sum_{n=1}^{N} A_n \sin(2\pi nt - u_n)$. Now, by (2.1.2), (2.1.3) we have that

$$\begin{aligned}\sum_{n=1}^{N} A_n \sin(2\pi nt - u_n) &= \sum_{n=1}^{N} \left(\underbrace{-A_n \sin u_n}_{\alpha_n} \cos(2\pi nt) + \underbrace{A_n \cos u_n}_{\beta_n} \sin(2\pi nt) \right) \\ &= \sum_{n=1}^{N} \left(\alpha_n \cos(2\pi nt) + \beta_n \sin(2\pi nt)\right),\end{aligned}$$

and, if we include a constant term (corresponding to $n = 0$) the trigonometric sum in (2.1.1) becomes

$$\frac{a_0}{2} + \sum_{n=1}^{N} \left(\alpha_n \cos(2\pi nt) + \beta_n \sin(2\pi nt)\right). \tag{2.1.4}$$

Now, it is easy to verify that

$$\cos t = \frac{e^{it} + e^{-it}}{2}, \quad \sin t = \frac{e^{it} - e^{-it}}{2i}.$$

Hence

$$\cos(2\pi nt) = \frac{e^{2\pi int} + e^{-2\pi int}}{2}, \quad \sin(2\pi nt) = \frac{e^{2\pi int} - e^{-2\pi int}}{2i}.$$

[2] For the moment we restrict to the case of period 1. The generalization to other periods poses no difficulty.

Using this, we have that

$$\begin{aligned}\frac{a_0}{2}+\sum_{n=1}^{N}\left(a_n\cos(2\pi nt)+\beta_n\sin(2\pi nt)\right) &= \frac{a_0}{2}+\sum_{n=1}^{N}\left(a_n\cdot\frac{e^{2\pi int}+e^{-2\pi int}}{2}+\beta_n\cdot\frac{e^{2\pi int}-e^{-2\pi int}}{2i}\right)\\ &= \frac{a_0}{2}+\sum_{n=1}^{N}\left(a_n\cdot\frac{e^{2\pi int}+e^{-2\pi int}}{2}-i\beta_n\cdot\frac{e^{2\pi int}-e^{-2\pi int}}{2}\right)\\ &= \frac{a_0}{2}+\sum_{n=1}^{N}\left(\frac{a_n-i\beta_n}{2}\right)\cdot e^{2\pi int}+\sum_{n=1}^{N}\left(\frac{a_n+i\beta_n}{2}\right)\cdot e^{-2\pi int}.\end{aligned}$$

Now, for $c_0=\frac{a_0}{2}$, $c_n=\frac{a_n-i\beta_n}{2}$ and $c_{-n}=\frac{a_n+i\beta_n}{2}$, we conclude that (2.1.4) can be written equivalently as

$$\sum_{n=-N}^{N}c_ne^{2\pi int}. \tag{2.1.5}$$

Therefore, the goal is to approximate a function $f:\mathbb{R}\to\mathbb{R}$ of period 1 by a trigonometric sum $\sum_{n=-N}^{N}c_ne^{2\pi int}$,for appropriate $c_n\in\mathbb{C}$ and $N\in\mathbb{N}\cup\{\infty\}$ [3]. At this point, two problems arise:

(i) How one can determine the complex coefficients c_n ?

(ii) How one can define a similarity measure for two functions f,g ?

Luckily, for the first problem we can hack the solution with reverse engineering: Assuming that the function f can be written as

$$f(t)=\sum_{n=-\infty}^{\infty}c_ne^{2\pi int}, \tag{2.1.6}$$

and ignoring matters of convergence, if we fix a $k\in\mathbb{Z}$ and multiply (2.1.6) by $e^{-2\pi ikt}$ we get that

$$\begin{aligned}f(t)e^{-2\pi ikt} &= \sum_{n=-\infty}^{\infty}c_ne^{2\pi int}\cdot e^{-2\pi ikt}\\ &= \sum_{n=-\infty}^{\infty}c_ne^{2\pi i(n-k)t}.\end{aligned}$$

Integrating on the interval $[-1,1]$ and taking a creative detour around the math police for the second time (changing the order of integration with summation), we get

$$\int_0^1 f(t)e^{-2\pi ikt}\,dt=\sum_{n=-\infty}^{\infty}c_n\int_0^1 e^{2\pi i(n-k)t}\,dt. \tag{2.1.7}$$

Now for $k\neq n$ we have that

$$\begin{aligned}\int_0^1 e^{2\pi i(n-k)t}\,dt=\int_0^1\left[\frac{e^{2\pi i(n-k)t}}{2\pi i(n-k)}\right]'dt &= \frac{e^{2\pi i(n-k)t}}{2\pi i(n-k)}\Big|_0^1\\ &= \frac{1}{2\pi i(n-k)}\left(e^{2\pi i(n-k)}-e^0\right)\\ &= \frac{1}{2\pi i(n-k)}(1-1)=0,\end{aligned}$$

[3] We will see that in some cases the sum is an infinite series.

since $n - k \neq 0$ and $n - k \in \mathbb{Z}$. For $k = n$ we get that

$$\int_0^1 e^{2\pi i(n-k)t} = 1.$$

Hence, we conclude that

$$\int_0^1 e^{2\pi i(n-k)t} = \begin{cases} 1, & n = k \\ 0, & n \neq k \end{cases}.$$

Using this in (2.1.7) we get

$$c_n = \int_0^1 f(t)e^{-2\pi int}\, dt, \tag{2.1.8}$$

for $n \in \mathbb{Z}$. The coefficients in (2.1.8) are the *Fourier coefficients* of f. What we have achieved so far is that *if* we assume that the function f can be represented as a sum of trigonometric functions, then the coefficients c_n must be given by the integral $\int_0^1 f(t)e^{-2\pi int}\, dt$. What is left open is *when* and *how*, or in what sense a periodic function $f : \mathbb{R} \to \mathbb{R}$ can be represented by such a sum. This question requires more sophisticated mathematical tools, such as the Lebesgue integral, which is a part of measure theory. While we will not use any terms from measure theory, we will state some important results that are relevant to Fourier analysis. Some good references on measure theory are [6, 7, 5, 8]. Before we address the second problem mentioned before, we treat the case where the function $f : \mathbb{R} \to \mathbb{R}$ has period $T \neq 1$.

2.1.2 The Case of Period $T \neq 1$

Suppose that the function $f : \mathbb{R} \to \mathbb{R}$ is periodic with fundamental period $T \neq 1$. Then, the function $g : \mathbb{R} \to \mathbb{R}$ given by $g(t) = f(tT)$ is periodic with period 1 since for every $t \in \mathbb{R}$ we have

$$g(t + 1) = f((t + 1)T) = f(tT + T) = f(tT) = g(t).$$

The Fourier coefficients of g are given by

$$c_n = \int_0^1 g(t)e^{-2\pi int}\, dt.$$

Using $g(t) = f(tT)$ and the substitution $u = tT$ we get that

$$c_n = \int_0^1 g(t)e^{-2\pi int}\, dt = \int_0^1 f(tT)e^{-2\pi int}\, dt = \frac{1}{T}\int_0^T f(u)e^{-2\pi inu/T}\, du.$$

Therefore, if g can be expressed as $\sum_{n=-\infty}^{\infty} c_n e^{2\pi int}$, then f would be equal to $\sum_{n=-\infty}^{\infty} c_n e^{2\pi int/T}$, where $c_n = \frac{1}{T}\int_0^T f(t)e^{-2\pi int/T}$. These observations allow us to restrict to the case where the period of the function is equal to 1. Summarizing all the above we formulate the following definition. Additionally, we allow complex functions and include sufficient conditions for the existence of the Fourier coefficients.

Definition 2.1 (Fourier coefficients)**.** *Let $f : \mathbb{R} \to \mathbb{C}$ be a periodic function with fundamental period $T \in \mathbb{R}$ such that $\int_0^T |f(t)|\, dt < \infty$. Then, the* Fourier coefficients *of f are the complex*

numbers given by

$$\hat{f}(n) = \frac{1}{T}\int_0^T f(t)e^{-2\pi int/T}\,dt, \quad n \in \mathbb{Z}$$

A useful observation is that the coefficients $\hat{f}(n)$ can be calculated by integrating the function f in any interval of length T. To see this, let $a \in \mathbb{R}$. Then,

$$\frac{\partial}{\partial a}\left(\frac{1}{T}\int_a^{a+T} f(t)e^{-2\pi int/T}\,dt\right) = \frac{1}{T}\left(f(a+T)e^{-2\pi in(a+T)/T} - f(a)e^{-2\pi ina/T}\right),$$

but since $f(a+T) = f(a)$ and $e^{-2\pi in(a+T)/T} = e^{-2\pi ina/T}$ we get that

$$\frac{\partial}{\partial a}\left(\frac{1}{T}\int_a^{a+T} f(t)e^{-2\pi int/T}\,dt\right) = 0,$$

and since for $a = 0$ the integral is equal to $\hat{f}(n)$ we conclude that

$$\hat{f}(n) = \frac{1}{T}\int_a^{a+T} f(t)e^{-2\pi int}\,dt, \quad a \in \mathbb{R}.$$

An approach to define a measure of similarity between functions is to extend the notion of the usual inner product between vectors in $\mathbb{R}^n$ to more general spaces, where the space may be infinite-dimensional and the elements in that space may be functions. These spaces are called *Hilbert spaces*.

2.1.3 Hilbert Spaces

For two vectors $x = (x_1, \ldots, x_n)$, $y = (y_1, \ldots, y_n)$ in $\mathbb{R}^n$ the inner product is given by

$$\langle x, y\rangle = x_1y_1 + \cdots + x_ny_n. \tag{2.1.9}$$

The inner product is closely linked to the angle between the vectors x, y. In $\mathbb{R}^2$ and $\mathbb{R}^3$ this comes from the formula $\langle x, y\rangle = \|x\|_2 \cdot \|y\|_2 \cdot \cos\theta$, where θ is the angle between x, y. The generalization to dimensions $n \geq 4$ comes from the Cauchy-Schwarz inequality:

$$|\langle x, y\rangle| \leq \|x\|_2 \, \|y\|_2 \,. \tag{2.1.10}$$

The fact that $-1 \leq \frac{\langle x,y\rangle}{\|x\|_2\|y\|_2} \leq 1$ allows us to define $\cos\theta = \frac{\langle x,y\rangle}{\|x\|_2\|y\|_2}$. Perpendicularity (or orthogonality) can be expressed with the inner product via the relation $\langle x, y\rangle = 0$. Observe that this implies that $\cos\theta = 0$ which is equivalent to $\theta = 0$ or $\theta = 3\pi/2$ for $\theta \in [0, 2\pi)$. The inner product does more than identifying orthogonal vectors, however. When it's nonzero it tells us how much of one vector is in the direction of another. That is, the vector

$$\frac{\langle x, y\rangle}{\|y\|_2}\frac{y}{\|y\|_2},$$

is the projection of x onto the unit vector $y/\|y\|_2$. An alternative approach would be to think of the inner product as measuring how much one vector "similar" is to the other. Below we list some important properties of the inner product:

1. $\langle x, x\rangle \geq 0$ and $\langle x, x\rangle = 0 \iff x = 0$ (positive definiteness)

2. $\langle x, y\rangle = \langle y, x\rangle$ (symmetry)

3. $\langle ax, y\rangle = a\langle x, y\rangle$ for any scalar a (homogeneity)

4. $\langle x + w, y\rangle = \langle x, y\rangle + \langle w, y\rangle$ (additivity)

These properties allow us to define the inner product in a more general setting. Moreover, by allowing the inner product to take complex values, we can incorporate both the magnitude and phase information of the projection. Specifically, since any complex number $c \in \mathbb{C}$ can be expressed in polar form as $c = |c|e^{i\vartheta}$, where $|c|$ represents the magnitude and ϑ represents the phase angle, we can incorporate both pieces of information into the inner product. Therefore, we have the following definition:

Definition 2.2 (Inner product)**.** *Let H be a vector space over $\mathbb{C}$. An inner product is any function $\langle\cdot,\cdot\rangle : H \times H \to \mathbb{R}$ satisfying the properties:*

1. *For all $x \in H$, $\langle x, x\rangle \geq 0$ and $\langle x, x\rangle = 0 \iff x = 0$ (positive definiteness).*

2. *For all $x, y \in H$, $\langle x, y\rangle = \overline{\langle y, x\rangle}$.*

3. *For all $x, y, z \in H$, $\langle x + z, y\rangle = \langle x, y\rangle + \langle z, y\rangle$ and $\langle x, z + y\rangle = \langle x, z\rangle + \langle x, y\rangle$.*

4. *For all $x, y \in H$ and $t \in \mathbb{C}$, $\langle tx, y\rangle = t\langle x, y\rangle$ and $\langle x, ty\rangle = \overline{t}\langle x, y\rangle$.*

An inner product on a vector space naturally defines the notion of a norm, which in turn allows to define the distance between pair of points in the vector space.

Definition 2.3 (norm)**.** *Let H be a vector space over $\mathbb{C}$. A norm is a function $\|\cdot\| : H \to \mathbb{R}$ such that*

1. *$\|x\| \geq 0$ and $\|x\| = 0 \iff x = 0$ for all $x \in H$.*

2. *$\|tx\| = |t|\,\|x\|$ for all $t \in \mathbb{C}$ and $x \in H$.*

3. *$\|x + y\| \leq \|x\| + \|y\|$ for $x, y \in H$ (triangle inequality).*

It is easy to verify that $\|x\| = \langle x, x\rangle^{1/2}$ is a norm on H. Now, the distance between any points $x, y \in H$ can be defined by $d(x, y) = \|x - y\|$. As the notion of the inner product allows us to define distance, angle measurements, and calculate projections it is easy to generalize the notion of the orthonormal sequence.

Definition 2.4 (orthonomal sequence)**.** *Let H be a vector space over $\mathbb{C}$ and $\langle\cdot,\cdot\rangle$ an inner product. An orthonormal sequence on H is a sequence of vectors $\{e_i\}_{i=1}^{\infty}$ such that $\|e_i\| = 1$ for all i and $\langle e_i, e_j\rangle = 0$ for every $i \neq j$.*

Dealing with finite-dimensional spaces is simpler than dealing with infinite-dimensional spaces because in the finite case, there are fewer things to consider. For instance, given a vector $x =$

$(x_1, \ldots, x_n)$, the closest point in the subspace generated by the first $k < n$ orthonormal vectors $e_1, \ldots, e_k$ is

$$\sum_{i=1}^{k} \langle x, e_i \rangle e_i.$$

This follows from the fact that $\langle x, e_i \rangle$ is the projection of x onto e_i, since $\|e_i\|_2 = 1$. In the infinite-dimensional case, if $\{e_i\}_{i=1}^{\infty}$ is an orthonormal sequence, the closest point in the subspace generated by the vectors $\{e_k :, k \geq 2\}$ would be $\sum_{k=2}^{\infty} \langle x, e_k \rangle e_k$. However, to ensure that the element $\sum_{k=2}^{\infty} \langle x, e_k \rangle e_k$ is well-defined, the sequence of partial sums $s_n = \sum_{k=2}^{n} \langle x, e_k \rangle$ must converge. The notion of completeness is essential for ensuring the convergence of such sequences.

Definition 2.5 (Cauchy sequence)**.** *Let H be a vector space over $\mathbb{C}$ and $\langle \cdot, \cdot \rangle$ an inner product on H. We say the sequence $\{x_n\}_{n=1}^{\infty}$ is a Cauchy sequence if for every $\epsilon > 0$ there exists an index $n_0 \in \mathbb{N}$ such that $\|x_n - x_m\| < \epsilon$ for every $n, m \geq n_0$.*

In general, we say that a normed space X is complete[4] if every cauchy sequence $\{x_n\}_{n=1}^{\infty}$ is convergent, i.e. if there exists a point $x \in H$ such that for every $\epsilon > 0$ there exists $n_0 \in \mathbb{N}$ such that $\|x_n - x\| < \epsilon$ for every $n \geq n_0$. Summarizing all the above we have the following definition.

Definition 2.6 (Hilbert space)**.** *Let H be a vector space over $\mathbb{C}$ and $\langle \cdot, \cdot \rangle$ an inner product on H. We say that H is a Hilbert space if the norm $\|x\| = \langle x, x \rangle^{1/2}$ induced by the inner product is complete. Equivalently, every Cauchy sequence with respect to $\langle x, x \rangle^{1/2}$ is convergent.*

In the case of the $\mathbb{R}^n$, every vector $x \in \mathbb{R}^n$ can be uniquely expressed as a linear combination of the standard basis vectors $\{e_i\}_{i=1}^{n}$, where e_i is a vector whose i-th component is 1 and all other components are 0. Similarly, we say that an orthonormal sequence $\{e_i\}_{i=1}^{\infty}$ is an orthonormal basis for H if every element $x \in H$ can be uniquely expressed as an infinite linear combination of the basis vectors, i.e., $x = \sum_{i=1}^{n} \langle x, e_i \rangle e_i$. This is equivalent to saying that the sequence of partial sums $s_n = \sum_{i=1}^{n} \langle x, e_i \rangle e_i$ converges to x in H.
All this theory was presented in order to introduce the space of square-integrable functions $\mathcal{L}_2([0, 1])$ on the closed interval $[0, 1]$, which is the most important example of a Hilbert space for Fourier analysis. To this end, we let

$$\mathcal{L}_2([0, 1]) = \left\{ f : [0, 1] \to \mathbb{C} : f \text{ integrable and } \int_0^1 |f(t)|^2 \, dt < \infty \right\}, \tag{2.1.11}$$

The integral in 2.1.11 is with respect to the Lebesgue measure (notion from measure theory). However, in the upcoming examples, we will only restrict to functions that are also integrable in the standard Riemann sense. Of course, it is still left to show that $\mathcal{L}_2([0, 1])$ is a Hilbert space. The easy part is to show that $\mathcal{L}_2([0, 1])$ is a vector space. To see this, let $f, g \in \mathcal{L}_2([0, 1])$ and

[4] These spaces are called Banach spaces.

observe that since $2|f(t)| \cdot |g(t)| \leq |f(t)|^2 + |g(t)|^2$ we have that

$$\begin{aligned} |f(t) + g(t)|^2 &\leq (|f(t)| + |g(t)|)^2 \\ &= |f(t)|^2 + |g(t)|^2 + 2|f(t)||g(t)| \\ &\leq 2|f(t)|^2 + 2|g(t)|^2. \end{aligned}$$

Therefore, if $f, g \in \mathcal{L}_2([0,1])$ then

$$\int_0^1 |f(t) + g(t)|^2 \, dt \leq 2\int_0^1 |f(t)|^2 \, dt + 2\int_0^1 |g(t)|^2 \, dt < \infty,$$

which implies that $f + g \in \mathcal{L}_2([0,1])$. Similarly, it easily verified that $af \in \mathcal{L}_2([0,1])$ for every $f \in \mathcal{L}_2([0,1])$ and $a \in \mathbb{C}$. The inner product that is defined on $\mathcal{L}_2([0,1])$ is given by

$$\langle f, g \rangle = \int_0^1 f(t)\overline{g(t)} \, dt, \tag{2.1.12}$$

for every $f, g \in \mathcal{L}_2([0,1])$. The properties listed in Definition 2.2 can be verified by a straightforward calculation. The hardest part that requires additional theory is to prove that

$$\|f\|_2 = \left(\int_0^1 |f(t)|^2 \, dt\right)^{1/2}$$

is a norm and the space $\mathcal{L}_2([0,1])$ is complete. We take that for granted as well as that the complex exponentials $\{e^{2\pi int}\}_{n=1}^{\infty}$ form an orthonormal basis of $\mathcal{L}_2([0,1])$. For example, to see that the complex exponentials are an orthonormal sequence for $k \neq m$, write

$$\begin{aligned} \langle e^{2\pi ikt}, e^{2\pi imt} \rangle &= \int_0^1 e^{2\pi ikt}\overline{e^{imt}} \, dt = \int_0^1 e^{2\pi ikt} e^{-2\pi imt} \, dt \\ &= \int_0^1 e^{2\pi i(k-m)t} \, dt = \frac{1}{2\pi i(k-m)} e^{2\pi i(k-m)t}\Big|_0^1 \\ &= \frac{1}{2\pi i(k-m)} \left(e^{2\pi i(k-m)} - 1\right) = 0, \end{aligned}$$

since $e^{2\pi i(k-m)} = 1$. Now, for $k = m$ since $e^{2\pi ikt}e^{-2\pi ikt} = 1$ it follows that $\langle e^{2\pi ikt}, e^{2\pi ikt} \rangle = 1$. Therefore,

$$\langle e^{2\pi ikt}, e^{2\pi imt} \rangle = \begin{cases} 1, & k = m \\ 0, & k \neq m \end{cases}.$$

Having in mind the meaning of the inner product in $\mathbb{R}^n$, similarly, in this case, we may ask what is the component of a function $f(t)$ "in the direction" $e_n(t) = e^{2\pi nt}$? By analogy to the Euclidean case, it is given by the inner product

$$\langle f, e_n \rangle = \int_0^1 f(t)\overline{e_n(t)} \, dt = \int_0^1 f(t)e^{-2\pi int} \, dt,$$

which is precisely the n-th Fourier coefficient $\hat{f}(n)$. Thus writing the Fourier series

$$f(t) = \sum_{n=-\infty}^{\infty} \hat{f}(n) e^{2\pi i n t},$$

as we did in (2.1.6), is exactly like the decomposition in terms of an orthonormal basis and the associated inner product:

$$f = \sum_{n=-\infty}^{\infty} \langle f, e_n \rangle e_n. \tag{2.1.13}$$

This is exactly the equation resulting from the fact that the complex exponentials are an orthonormal basis. However, the above equation is with respect to the $\mathcal{L}_2$-norm. This is formally expressed as

$$\left\| f - \sum_{n=-N}^{N} \langle f, e_n \rangle e_n \right\|_2 = \left(\int_0^1 \left| f(t) - \sum_{n=-N}^{N} \langle f, e_n \rangle e_n(t) \right|^2 dt \right)^{1/2} \longrightarrow 0,$$

as $N \to \infty$. Of course, the interesting part is when (2.1.13) holds in the pointwise sence, i.e.

$$f(t) = \sum_{n=-\infty}^{\infty} \langle f, e_n \rangle \cdot e^{2\pi i n t},$$

for every $t \in \mathbb{R}$. We take a break from the theoretical results and postpone this question for Subsection 2.1.6.

2.1.4 Sinusoids and Fourier Coefficients - The Role of Phase

While introducing complex numbers enables us to elegantly handle algebraic operations, it can also detract from our original objective of representing a periodic function $f : \mathbb{R} \to \mathbb{R}$ with fundamental period 1 as a sum of sinusoids of the form $A_n \sin(2\pi(nt - \phi_n))$. In this paragraph, we explain the relation between the Fourier coefficients

$$c_n = \int_0^1 f(t) e^{-2\pi i n t} \, dt,$$

and the sinusoids $A_n \sin(2\pi(nt - \phi_n))$ as well as the role of phase ϕ_n, and its relation with the polar angle $\arg(c_n)$ of the complex number $c_n = |c_n| e^{i \arg c_n}$.

To begin with, we remind that any complex number $z = a + i\beta$ can be expressed in polar form as $z = |z| e^{i \arg(z)}$ (see Figure (2.3)), where $\arg(z) \in [0, 2\pi)$ is the angle of the vector (a, β) with the unit vector $e_1 = (1, 0)$. Furthermore, for a complex function $f : \mathbb{R} \to \mathbb{C}$, and any real numbers $a < \beta$ we have that (by definition)

$$\int_a^\beta f(t) \, dt = \int_a^\beta \mathrm{Re}(f(t)) \, dt + i \int_a^\beta \mathrm{Im}(f(t)) \, dt.$$

Therefore, since the function $f(t) e^{-2\pi n t}$ takes complex values the number c_n is a complex number, and as such it has a polar form $|c_n| e^{i \arg(c_n)}$.

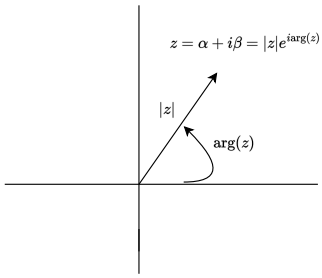

Figure 2.3. *Polar representation of complex numbers.*

In Subsection (2.1.3) we saw that every $f \in \mathcal{L}([0,1])$ can be expressed as

$$f = \sum_{n=-\infty}^{\infty} \langle f, e_n \rangle e_n,$$

in the $\mathcal{L}_2$-sense, where $e_n(t) = e^{2\pi i n t}$. Suppose now that $f : \mathbb{R} \to \mathbb{R}$ is periodic with fundamental period equal to 1 such that

$$\int_a^{a+1} |f(t)|^2 \, dt < \infty,$$

for some $a \in \mathbb{R}$. By letting $g(t) = f(t + a)$ for $t \in [0,1]$ we obtain a function in $\mathcal{L}_2([0,1])$. Conversely, if $f \in \mathcal{L}_2([0,1])$ we can extend this function to a periodic function $\tilde{f} : \mathbb{R} \to \mathbb{R}$ of fundamental period 1 by letting $\tilde{f}(t) = f(t - [t])$. [5] In other words, there is a one to one and onto correspondence between the functions in $\mathcal{L}_2([0,1])$ and the periodic functions of fundamental period 1 which are square integrable in some interval of length 1. Summarizing, we obtain the following theorem.

Theorem 2.1 (Convergence of the Fourier series in $\mathcal{L}_2$)**.** *Let $f : \mathbb{R} \to \mathbb{R}$ be a periodic function with fundamental period equal to 1 such that $\int_a^{a+1} |f(t)|^2 \, dt < \infty$ for some $a \in \mathbb{R}$. Then if $f_a(t) = f(t + a)$,*

$$\left\| f_a - \sum_{n=-N}^{N} \langle f_a, e_n \rangle e_n \right\|_2 = \left(\int_0^1 \left| f(t+a) - \sum_{n=-N}^{N} \langle f_a, e_n \rangle e_n(t) \right|^2 dt \right)^{1/2} \longrightarrow 0, \tag{2.1.14}$$

as $N \to \infty$.

The preceding theorem provides insight into how a periodic function $f : \mathbb{R} \to \mathbb{R}$ can be expressed as a trigonometric sum. Our original objective, however, was to identify the A_n and ϕ_n values that would enable us to construct the sum

$$\sum_{n=1}^{N} A_n \sin(2\pi(nt - \phi_n))$$

that best approximates f. The relative information about the optimal A_n and ϕ_n values are hidden in the Fourier coefficients $\hat{f}(n)$ and $\hat{f}(-n)$. Indeed, in (2.1.4) we saw that $A_n \sin(2\pi(nt - \phi_n)$ can be written as $a_n \cos(2\pi n t) + \beta_n \sin(2\pi n t)$, where $a_n = -A_n \sin u_n$, $\beta_n = A_n \cos u_n$, and $u_n = 2\pi\phi_n$.

[5] $[t]$ is the integer part of $t \in \mathbb{R}$.

To obtain (2.1.5) we saw that $\hat{f}(n) = \frac{a_n - i\beta_n}{2}$ and $\hat{f}(-n) = \frac{a_n + i\beta_n}{2}$. Therefore, we have that

$$\hat{f}(n) = -\frac{A_n}{2}\left(\sin u_n - i\cos u_n\right).$$

Using the identities $\cos(u_n - \frac{\pi}{2}) = \sin u_n$ and $\sin(u_n - \frac{\pi}{2}) = -\cos u_n$ we get

$$\hat{f}(n) = -\frac{A_n}{2}\left(\cos(u_n - \frac{\pi}{2}) + i\sin(u_n - \frac{\pi}{2})\right) = -\frac{A_n}{2}e^{i(u_n - \frac{\pi}{2})}.$$

From the above identity, we see that the magnitude A_n is given by $2|\hat{f}(n)|$. For the phase we have that

$$\begin{aligned} u_n - \frac{\pi}{2} &= \arg(\hat{f}(n)) \iff \\ 2\pi\phi_n - \frac{\pi}{2} &= \arg(\hat{f}(n)) \iff \\ \phi_n &= \frac{\arg(\hat{f}(n))}{2\pi} + \frac{1}{4}. \end{aligned}$$

The magnitude A_n indicates the amplitude of the sinusoid $A_n \sin(2\pi(nt - \phi_n))$ in the original function f, while the phase ϕ_n corresponds to the horizontal shift of the sinusoid that best aligns with the function f. To better understand the role of phase consider as an example the waveform of the first 1 second of a A4 note played on piano in Figure 2.4.

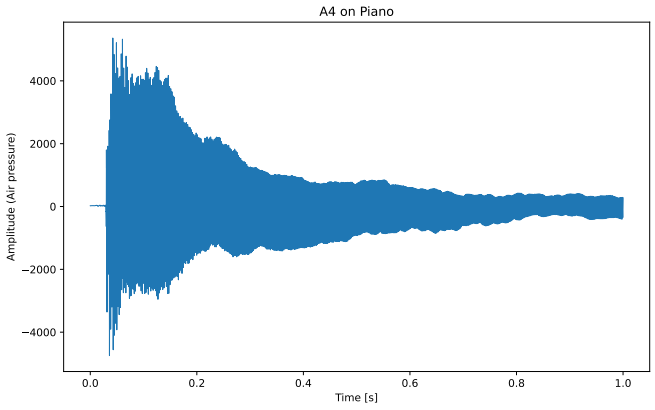

Figure 2.4. *The waveform of A4 on piano.*

Since we are analyzing the waveform of A4 over an interval of length 1, we can assume that the function f representing the waveform is periodic with period 1 by simply repeating the function for values $|t| > 1$. Our objective is to determine which exponential function $e^{2\pi int}$ contributes the most to the projection of f. As we have seen, we can obtain the projections using the inner product:

$$\langle f, e^{2\pi int}\rangle \cdot e_n(t) = \left(\int_0^1 f(t)\overline{e^{2\pi int}}\,dt\right) \cdot e_n(t) = \hat{f}(n) \cdot e_n(t).$$

This can be expressed as the product of the Fourier coefficient of f with the exponential function $e^{2\pi int}$, i.e., $\hat{f}(n) \cdot e^{2\pi int}$. By examining the magnitudes of the Fourier coefficients $|\hat{f}(n)|$, we can identify the value of n that corresponds to the exponential function $e^{2\pi int}$ that provides the highest

contribution to f. The value of n determines the sinusoid $A_n \sin(2\pi(nt-\phi_n)$ that best approximates f, or equivalently, the *harmonic* with the highest contribution to f. To determine the harmonic with the highest contribution to f in the waveform of A4, we zoom in to the waveform in an interval corresponding to 10 ms. By looking at Figure 2.5 we can see that in 10 ms there are approximately 4.5 crests. This corresponds to the sinusoid with fundamental frequency of 450 Hz. As we will see in chapter 3, each piano note is determined by its fundamental frequency. In the case of the A4 piano key, this frequency is 440 Hz, which is very close to what we see in the figure!

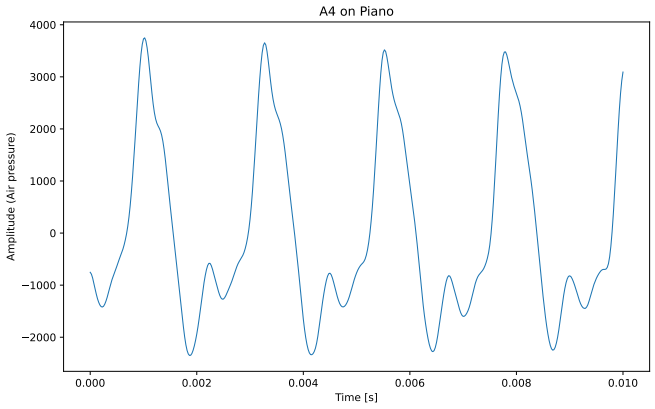

Figure 2.5. *The waveform of A4 on piano zoomed in 1ms.*

Hence, the index of the harmonic contributing the most is $n = 440$. In Figure 2.6 we see the A4 piano plotted with the sinusoid $\sin(2\pi(440t))$. For simplicity, we have rescaled the waves to have the same amplitudes.

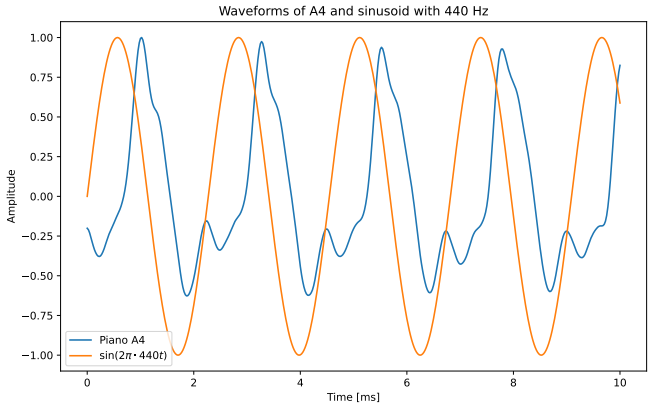

Figure 2.6. *The A4 piano note corresponding to 440 Hz and the sinusoid* $\sin(2\pi \cdot 440t)$.

It is apparent that the sinusoid $\sin(2\pi \cdot 440t)$ closely matches the waveform of A4 within that time interval. Nevertheless, we can improve the fit by adjusting the phase parameter ϕ accordingly. By Figure 2.6 we see that we need to shift the sinusoid to the right. This corresponds to $\sin(2\pi(440t-\phi))$. In Figure 2.7 we see the same graph for $\phi = 0.218$, which is calculated numerically by

$$\phi^* = \arg\min_{\phi \in [0,1)} \int_0^1 |f(t) - \sin(2\pi(440t - \phi))|^2 \, dt.$$

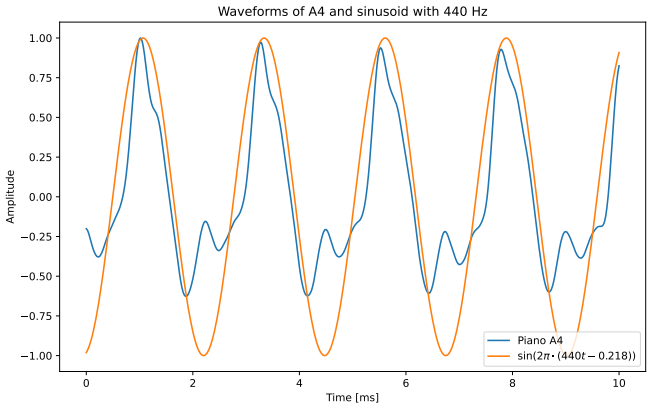

Figure 2.7. *The A4 piano note and the sinusoid* $\sin(2\pi \cdot (440t - 0.218))$.

2.1.5 The Spectrum

The reason why the approximation in Figure 2.7 is not perfect, is because we need more harmonics to accurately approximate the waveform of the A4 note. This is due to the fact that the A4 key it is synthesized from many harmonics and not just once. This means that there are also $n \neq 440$ such that $\hat{f}(n) \neq 0$. In the preceding paragraphs, we have often used the more geometric term *period* instead of the more natural term *frequency.* It's natural to talk about the period for a Fourier series representation of $f(t)$,

$$f(t) = \sum_{n=-\infty}^{\infty} \hat{f}(n) e^{2\pi i n t}.$$

The period is 1. However, each exponential $e^{2\pi i n t}$ is periodic with period 1 and $1/n$ at the same time. Therefore, the function f is synthesized from many harmonics, many frequencies, positive and negative, and perhaps an infinite number. The set of frequencies present in a given periodic signal is the *spectrum* of the signal. Graphically the spectrum is represented as shown in Figure 2.8.

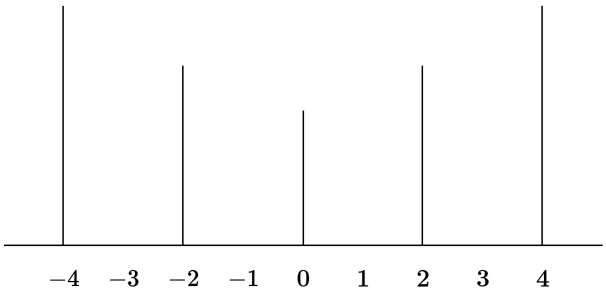

Figure 2.8. *The spectrum of a periodic function.*

In Figure 2.8, it is important to note that the magnitudes $|\hat{f}(0)|^2, |\hat{f}(\pm 1)|^2, |\hat{f}(\pm 2)|^2, \ldots$, are depicted, rather than the coefficients themselves, $\hat{f}(0), \hat{f}(\pm 1), \hat{f}(\pm 2), \ldots$. These square magnitudes, $|\hat{f}(n)|^2$, represent the energy of the (positive and negative) harmonics $e^{\pm 2\pi i n t}$, and together they form the sequence of squared magnitudes $|\hat{f}(0)|^2, |\hat{f}(\pm 1)|^2, |\hat{f}(\pm 2)|^2, \ldots$, known as the *power spectrum* or *energy spectrum*.

2.1.6 Pointwise convergence and examples

In the previous subsections we proved that any integrable periodic $f : \mathbb{R} \to \mathbb{R}$ can be expressed as a sum of trigonometric functions in the $\mathcal{L}_2$-sense. However, there is an analog theorem in the pointwise sense:

Theorem 2.2 (Pointwise convergence)**.** *Suppose that f is continuous with a continuous derivative except at perhaps a finite number of points (in a period). Then, for each $a \in [0, 1]$,*

$$\sum_{n=-N}^{N} \hat{f}(n) e^{2\pi i n a} \longrightarrow \frac{1}{2}\left(\lim_{t \to a^-} f(t) + \lim_{t \to a^+} f(t)\right), \tag{2.1.15}$$

as $N \to \infty$.

In the case where f is continuous at a, since $\lim_{t \to a^-} f(t) = \lim_{t \to a^+} f(t)$ we simply have that the Fourier series of f coincides with f. This is a powerful result which we illustrate with the following examples: Consider the function

$$f(t) = \begin{cases} \frac{1}{2} + t, & -\frac{1}{2} \le t \le 0 \\ \frac{1}{2} - t, & 0 \le t \le \frac{1}{2} \end{cases},$$

We can extend f to a periodic function $\tilde{f} : \mathbb{R} \to \mathbb{R}$ with period 1 by repeating f. The general formula of $\tilde{f}$ is then given by

$$\tilde{f}(t) = \begin{cases} \frac{1}{2} - (t - [t]), & 0 \le t - [t] \le \frac{1}{2} \\ t - [t] - \frac{1}{2}, & \frac{1}{2} < t - [t] \le 1 \end{cases}.$$

Since f is an even function we can easily calculate the Fourier coefficients as follows:

$$\begin{aligned} \hat{f}(n) &= \int_{-1/2}^{1/2} f(t) e^{-2\pi i n t}\, dt = 2\int_0^{1/2} \left(\frac{1}{2} - t\right) \cos(2\pi n t)\, dt \\ &= \frac{2}{2\pi n} f(t) \cancel{\sin(2\pi n t)}\Big|_0^{1/2} \cancel{\to 0} + \frac{1}{\pi n}\int_0^{1/2} \sin(2\pi n t)\, dt \\ &= -\frac{1}{2(\pi n)^2} \cos(2\pi n t)\Big|_0^{1/2} = \frac{1}{2(\pi n)^2}\left(1 - \cos(\pi n)\right). \end{aligned}$$

Since for every $n = 2k$ we have that $\cos(\pi n) = 1$ we conclude that

$$\hat{f}(k) = \begin{cases} \frac{1}{(\pi n)^2}, & n = 2k + 1 \\ 0, & n = 2k \end{cases}.$$

Furthermore, for $n = 0$ we have that

$$\hat{f}(0) = \int_{-1/2}^{1/2} f(t)\, dt = 2\int_0^{1/2} \left(\frac{1}{2} - t\right) dt = \frac{1}{2} - t^2\Big|_0^{1/2} = \frac{1}{2} - \frac{1}{4} = \frac{1}{4}.$$

Since $\hat{f}(n) = \hat{f}(-n)$ we conclude that the fourier series of f is given by

$$\begin{aligned}\sum_{n=-\infty}^{\infty} \hat{f}(n)e^{2\pi int} &= \frac{1}{4} + \sum_{n=1}^{\infty} \hat{f}(n)\left(e^{2\pi int} + e^{-2\pi int}\right) \\ &= \frac{1}{4} + \sum_{n=1}^{\infty} \frac{2}{\pi^2(2n+1)^2} \cos(2\pi(2n+1)t).\end{aligned}$$

By the continuity of f and Theorem 2.2, it follows that

$$f(t) = \frac{1}{4} + \sum_{n=1}^{\infty} \frac{2}{\pi^2(2n+1)^2} \cos(2\pi(2n+1)t), \tag{2.1.16}$$

for every $-\frac{1}{2} \leq t \leq \frac{1}{2}$. As we can see the function f has infinitely many harmonics despite that the function is continuous. This is due the fact of the discontinuity of the derivative. Now, an interesting byproduct of (2.1.16) is that we can calculate the infinite sum $\sum_{n=1}^{\infty} \frac{1}{n^2}$. Indeed, plugging $t = 0$ we get that

$$\begin{aligned}\frac{1}{2} &= \frac{1}{4} + \sum_{n=1}^{\infty} \frac{2}{\pi^2(2n+1)^2} \iff \\ \frac{\pi^2}{8} &= \sum_{n=1}^{\infty} \frac{1}{(2n+1)^2}.\end{aligned}$$

On the other hand,

$$\begin{aligned}\sum_{n=1}^{\infty} \frac{1}{n^2} &= \sum_{n=1}^{\infty} \frac{1}{(2n)^2} + \sum_{n=1}^{\infty} \frac{1}{(2n+1)^2} \iff \\ \sum_{n=1}^{\infty} \frac{1}{n^2} &= \frac{1}{4}\sum_{n=1}^{\infty} \frac{1}{n^2} + \frac{\pi^2}{8} \iff \\ \frac{3}{4}\sum_{n=1}^{\infty} \frac{1}{n^2} &= \frac{\pi^2}{8},\end{aligned}$$

where we deduce that $\sum_{n=1}^{\infty} \frac{1}{n^2} = \frac{\pi^2}{6}$. We conclude this paragraph with an example from which the pointwise convergence fails at exactly one point. To this end, consider the function $f(t) = 1$, for $0 \leq t < 1/2$, and $f(t) = -1$, for $1/2 \leq t < 1$. Figure 2.9 shows the graph of this function. Now, since f is odd, we conclude that $\hat{f}(0) = 0$. For $n \neq 0$ we have that

$$\begin{aligned}\hat{f}(n) &= \int_{-1/2}^{1/2} f(t)e^{-2\pi int}\, dt \\ &= \int_{-1/2}^{0} e^{-2\pi int}\, dt + \int_{0}^{1/2} e^{2\pi int}\, dt \\ &= -\frac{1}{2\pi in} e^{-2\pi int}\Big|_0^{1/2} + \frac{1}{2\pi in} e^{-2\pi int}\Big|_{1/2}^{1} = \frac{1}{\pi in}\left(1 - e^{-\pi in}\right).\end{aligned}$$

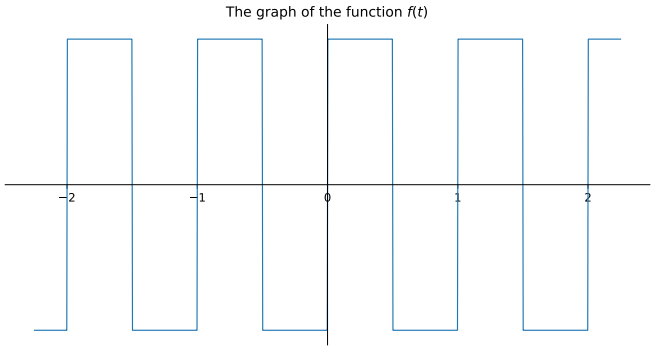

Figure 2.9. *The graph of function $f(t)$ on the interval* $[-2.25, 2.25]$

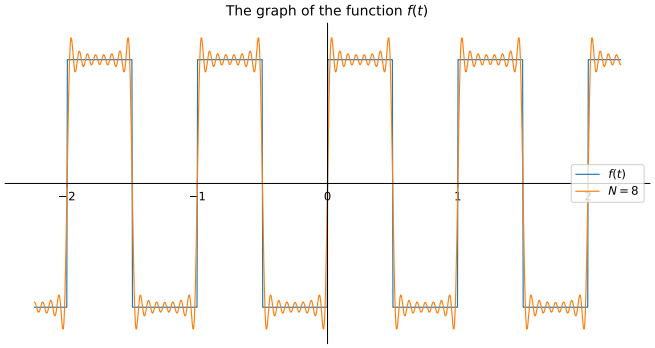

Figure 2.10. *The graph of function $f(t)$ and its fourier series up to the 8th frequency.*

Therefore, the Fourier series of f is

$$\sum_{n\neq 0} \frac{1}{\pi i n} \left(1 - e^{-\pi i n}\right) e^{2\pi i n t}.$$

Using the relation

$$1 - e^{-\pi i n} = \begin{cases} 0, & n \text{ even} \\ 2, & n \text{ odd} \end{cases},$$

the series becomes

$$\sum_{n \text{ odd}} \frac{2}{2\pi i n} e^{2\pi i n t}.$$

Combining the positive and negative terms together with the identity

$$e^{2\pi i n t} - e^{-2\pi i n t} = 2i \sin 2\pi n t,$$

we get the following simplified form of the Fourier series:

$$\frac{4}{\pi} \sum_{k=0}^{\infty} \frac{1}{2k+1} \sin\left(2\pi(2k+1)t\right).$$

Now, by Theorem 2.2 we know that

$$f(t) = \frac{4}{\pi} \sum_{k=0}^{\infty} \frac{1}{2k+1} \sin\left(2\pi(2k+1)t\right),$$

every $-\frac{1}{2} \leq t \leq \frac{1}{2}$ with $t \neq 0$. Figure 2.10 shows the graph of the function f along with its fourier series up to the 8th frequency.

2.2 The Fourier Transform

2.2.1 Motivation and Definition

In the previous subsection we focused on periodic functions $f : \mathbb{R} \to \mathbb{R}$. We saw that apart from its fundamental frequency $1/T$, these functions may have other frequencies as well, corresponding to multiples $\pm n/T$, for $n \in \mathbb{Z}$. These frequencies are referred to as the *harmonics* of f. However, having a theory that only relies on periodic functions is restrictive. For instance, when we analyzed the waveform of the A4 piano in Figure 2.4 we had to zoom in the order of milliseconds in order to observe its periodic nature. Until now, the way that we tackled the problem was to use periodization. To this end, since any audio signal will have a finite duration we assumed that is a periodic function by defining it on the whole $\mathbb{R}$ by repeating the signal on the rest of the intervals. We encountered this approach also in the two examples presented in paragraph 2.1.6. However, this approach is restrictive in audio signal processing where we want to analyze different audio signals for a specific task at the same time. These signals may have different duration, and as such, we would have to periodize with the maximum period of these signals, an approach which may lead to artifacts. To better understand the dependency of the period T let us consider the following example inspired by [4]. Consider the "rect" function ("rect" for "rectangle") defined by

$$\Pi(t) = \begin{cases} 1, & |t| < 1/2 \\ 0, & |t| \geq 1/2 \end{cases}.$$

This function is not periodic but we make the assumption that all its information lies on the interval $[-1/2, 1/2]$, corresponding to its non-zero part, and use periodization to make it periodic. Let's assume that $\Pi(t)$ is periodic with period $T > 1$. In this case, the function will be zero on the intervals $(-\frac{T}{2}, -\frac{1}{2}] \cup [\frac{1}{2}, \frac{T}{2})$. In Figure 2.11 you can see the graph of the periodized rect function.

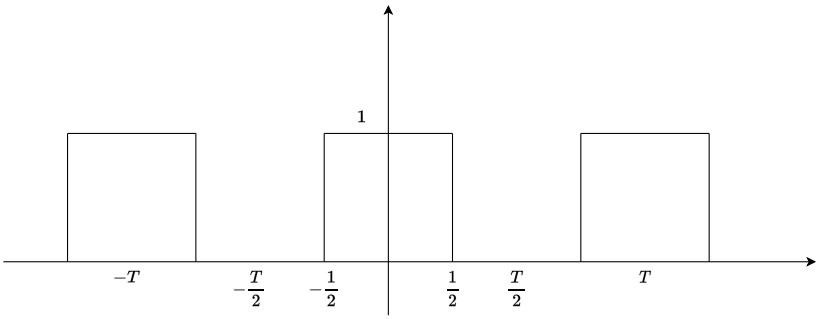

Figure 2.11. *Periodization of the rect function with fundamental period T.*

We denote by f_T the periodization of Π and calculate its Fourier coefficients. Since f_T is even we have that $\hat{f_T}(0) = 1$. Now, again by using the fact that f_T is even, for any $n \neq 0$ we have that

$$\begin{aligned}\hat{f_T}(n) &= \frac{1}{T}\int_{-\frac{T}{2}}^{\frac{T}{2}} f_T(t)e^{-2\pi int/T}\,dt = \frac{2}{T}\int_0^{\frac{T}{2}} \cos(2\pi nt/T)\,dt \\ &= \frac{2}{T}\int_0^{1/2} \cos(2\pi nt/T)\,dt = \frac{\sin(2\pi nt/T)}{\pi n}\Big|_0^{\frac{1}{2}} = \frac{\sin(\pi n/T)}{\pi n}.\end{aligned}$$

Therefore, the Fourier series of f_T is given by

$$\sum_{n=-\infty}^{\infty} \hat{f_T}(n)e^{2\pi int/T} = \sum_{n=-\infty}^{\infty} \frac{\sin(\pi n/T)}{\pi n}e^{2\pi int/T}.$$

In Figures 2.12, 2.13, 2.14 we see the Fourier coefficients corresponding for $T = 2, 6, 16$[6].

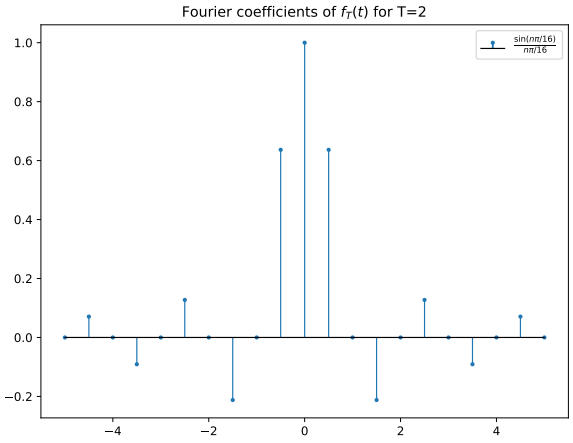

Figure 2.12. *Normalized Fourier coefficients of f_T for $T = 2$.*

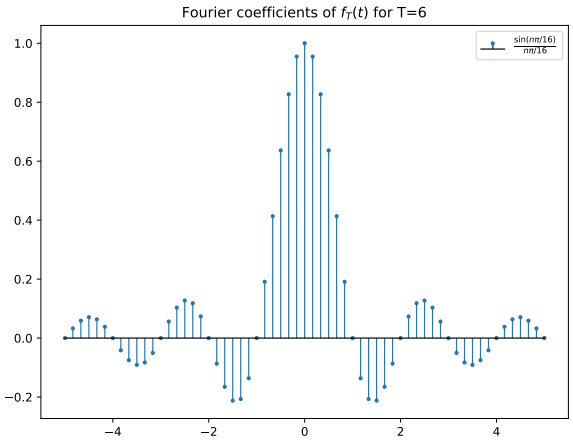

Figure 2.13. *Normalized Fourier coefficients of f_T for $T = 6$.*

[6] Actually these are the normalized coefficients $\frac{\sin(\pi nT)}{n\pi/T}$.

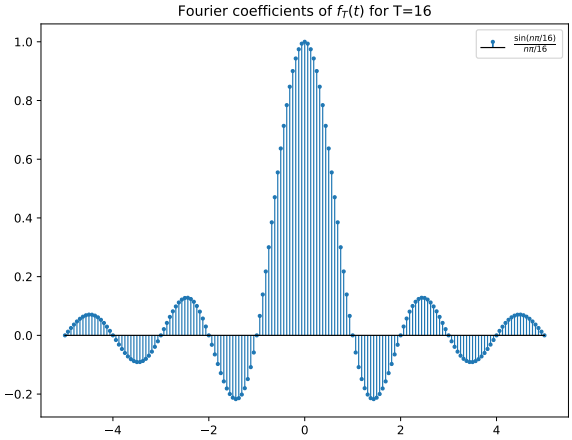

Figure 2.14. *Normalized Fourier coefficients of f_T for $T = 16$.*

The previous figures show that as T grows large then the graph of the mapping $n/T \mapsto \frac{\sin(n\pi/T)}{n\pi/T}$ tends to a function defined on the whole $\mathbb{R}$. This reflects the fact that the harmonics n/T are inversely proportional to the fundamental period T, and as T approaches infinity these harmonics are getting closer and closer. Therefore, if we wanted to burden ourselves from the imposed period T an idea is to view the function $\Pi(t)$ as the limit of $f_T(t)$ for $T \to \infty$. Then, in this case the function $\Pi(t)$ will have a "harmonic" for each variable $s \in \mathbb{R}$ for which $s \approx n/T$, for large values of T. The value of the Fourier coefficient for s will be equal to $\frac{\sin(\pi s)}{\pi s}$, which is exactly the graph of the function that $\frac{\sin(n\pi/T)}{n\pi/T}$ approaches as T grows large. To obtain a formula to calculate the contribution of the frequency $s \in \mathbb{R}$ to the function Π we may start from the definition for the Fourier coefficient of f_T for the frequency $s \approx n/T$. This is given by

$$\hat{f}(n/T) = \hat{f}_T(n) = \frac{1}{T}\int_{-\frac{T}{2}}^{\frac{T}{2}} f_T(t)e^{-2\pi int/T}\,dt. \tag{2.2.1}$$

Provided that the integral on the right-hand side of (2.2.1) is well defined for every $T > 0$ we can use Riemann's sums to see what is happening when $T \to \infty$. To this end, we partition the interval $[-\frac{T}{2}, \frac{T}{2}]$ to equally space subintervals of length $1/T$. For simplicity, to avoid any remainders that may occur during the partition of the interval $\left[-\frac{T}{2}, \frac{T}{2}\right]$ we assume that T^2 divisible by n. Mathematically, the partition can be described as the union

$$\left[-\frac{T}{2}, \frac{T}{2}\right] = \bigcup_{k=0}^{\frac{T^2}{n}-1} \left[-\frac{T}{2} + k\cdot\frac{n}{T}, -\frac{T}{2} + (k+1)\cdot\frac{n}{T}\right]. \tag{2.2.2}$$

If we denote by I_n the intervals on the right hand side of (2.2.2) and let $c_n \in I_n$ be the middle point, the integral on (2.2.1) can be approximated by the sum

$$\int_{-\frac{T}{2}}^{\frac{T}{2}} f_T(t)e^{-2\pi int/T}\,dt \approx \frac{n}{T}\sum_{n=1}^{\frac{T^2}{n}-1} f_T(c_n)e^{-2\pi inc_n/T}. \tag{2.2.3}$$

Now, if we let $s = \frac{n}{T}$ we can view the right-hand side of (2.2.3) again as a Riemann sum of the integral of the function $f_T(t)e^{-2\pi ist}$ on the interval $\left[-\frac{T}{2}, \frac{T}{2}\right]$, i.e.

$$\int_{-T/2}^{T/2} f_T(t)e^{-2\pi ist}\,dt \approx \frac{n}{T}\sum_{n=1}^{\frac{T^2}{n}-1} f_T(c_n)e^{-2\pi inc_n/T}. \tag{2.2.4}$$

Since for large values of T we have that

$$\int_{-T/2}^{T/2} f_T(t)e^{-2\pi ist}\,dt \approx \int_{-\infty}^{\infty} \Pi(t)e^{-2\pi ist}\,dt$$

we can define the continuous analog of the coefficient $\hat{f}_t(n)$ for the non periodic function $\Pi : \mathbb{R} \to \mathbb{R}$ on the point $s \in \mathbb{R}$ to be

$$\hat{\Pi}(s) = \int_{-\infty}^{\infty} \Pi(t)e^{-2\pi ist}\,dt. \tag{2.2.5}$$

Solving the above integral we get $\hat{\Pi}(s) = \frac{\sin(\pi s)}{\pi s}$. The equation (2.2.5) is exactly the Fourier Transform (FT) of $\Pi(t)$ at s and is denoted by $\mathcal{F}(\Pi)(s)$. As in the case of the periodic functions, below we state the formal definition along with the sufficient condition for the existence of the Fourier Transform.

Definition 2.7 (Fourier Transform)**.** *Let $f : \mathbb{R} \to \mathbb{R}$ be a function such that $\int_{-\infty}^{\infty} |f(t)|\,dt < \infty$. The Fourier Transform of f is the function $\mathcal{F}(f) : \mathbb{R} \to \mathbb{R}$ given by*

$$\mathcal{F}f(s) = \int_{-\infty}^{\infty} f(t)e^{-2\pi ist}\,dt, \tag{2.2.6}$$

for $s \in \mathbb{R}$.

As opposed to the case of periodic functions where the spectrum was the discrete set $\{n/T : n \in \mathbb{Z}\}$, in the continuous case one has to deal with the continuous spectrum $\{s : s \in \mathbb{R}\}$. The set $\{|\mathcal{F}f(s)|^2 : s \in \mathbb{R}\}$ is referred to as the *power spectrum* of f. In engineering and especially in signal processing they usually say that the original signal (function) $f(t)$ is defined on the *time domain* and the Fourier Transform $\mathcal{F}(f)$ is defined on the *frequency domain* . We will say more about signals and in particular audio signals in Chapter 3. For our purposes, it is beneficial to think of the theory of the Fourier Transform in the context of signals. In engineering an *analog signal* is simply the same entity as a function $f : \mathbb{R} \to \mathbb{R}$. A *digital signal* is a sequence $x : \mathbb{Z} \to \mathbb{R}$. We will use these terms (mathematical and engineering) interchangeably throughout the rest of this chapter.

The theory of the Fourier Transform is quite extensive with important mathematical implications. However, since we are interested in audio signal processing, there is no need to develop the theory in great detail, as we will work mainly with digital signals. In this case, we use the discrete analog of the Fourier Transform, the *Discrete Fourier Transform* (DFT). Therefore, in the following paragraphs, we present without any proofs the most important results and concepts related to audio signal processing.

2.2.2 The Inverse Fourier Transform

There is a one-to-one and onto correspondence between the function f and its Fourier Transform $\mathcal{F}f$. This means, that the knowledge of $\mathcal{F}F$ allows us to retrieve f and vice versa. To introduce the Inverse Fourier Transform (IFT) we follow a heuristic approach with reverse engineering by avoiding the mathematical formality. To this end, suppose that f is zero outside some interval. We can periodize f to have period T in such a way that the interval $[-T/2, T/2]$ contains the interval where f is non-zero. In this case, f will have a Fourier series expansion

$$f(t) = \sum_{n=-\infty}^{\infty} c_n e^{2\pi i n t/T}, \tag{2.2.7}$$

where the coefficients are given by

$$c_n = \frac{1}{T}\int_{-T/2}^{T/2} f(t)e^{-2\pi i n t/T}\, dt.$$

Comparing the previous integral and the integral in (2.2.6) we get that

$$\frac{1}{T}\mathcal{F}f(n/T) = c_n.$$

Substituting this equation in (2.2.7) we get that

$$f(t) = \sum_{n=-\infty}^{\infty} \frac{1}{T}\mathcal{F}f(n/T)e^{2\pi i n t/T}.$$

The previous sum can be viewed as a Riemann sum of the integral $\int_{-T/2}^{T/2} \mathcal{F}f(s)e^{2\pi i s t}\, ds$. Therefore, using the fact that f is zero outside the interval $[-T/2, T/2]$ we conclude that

$$\begin{aligned} f(t) &= \sum_{n=-\infty}^{\infty} \frac{1}{T}\mathcal{F}f(n/T)e^{2\pi i n t/T} \\ &\approx \int_{-T/2}^{T/2} \mathcal{F}f(s)e^{2\pi i s t}\, ds \\ &= \int_{-\infty}^{\infty} \mathcal{F}f(s)e^{2\pi i s t}\, ds. \end{aligned}$$

Since any function $f : \mathbb{R} \to \mathbb{R}$ can be approximated by a periodic function (possibly with large period T) on the limit we expect to have

$$f(t) = \int_{-\infty}^{\infty} \mathcal{F}f(s)e^{2\pi i s t}\, ds,$$

for non-periodic functions as well. The previous formula defines the Inverse Fourier Transform which is summarized in the following definition.

Definition 2.8 (Inverse Fourier Transform)**.** *Let $g : \mathbb{R} \to \mathbb{R}$ be a function such that $\int_{-\infty}^{\infty} |g(t)|\, dt < \infty$. The Inverse Fourier Transform of g is the function $\mathcal{F}^{-1} : \mathbb{R} \to \mathbb{R}$ given by*

$$\mathcal{F}^{-1}g(t) = \int_{-\infty}^{\infty} g(s)e^{2\pi ist}\, ds, \tag{2.2.8}$$

for every $t \in \mathbb{R}$.

$\mathcal{F}f$ and $\mathcal{F}^{-1}f$ are closely related through the following identities referred to as the Fourier Inversion Theorem :

$$\mathcal{F}\left(\mathcal{F}^{-1}f\right)(t) = f(t), \quad \mathcal{F}^{-1}\left(\mathcal{F}f\right)(s) = f(s). \tag{2.2.9}$$

We conclude this paragraph by stating Parseval's identity which relates the energy of the signal in the time domain and the energy spectrum in the frequency domain. Parseval's identity says that these two are equal. In other words,

$$\int_{-\infty}^{\infty} |f(t)|^2\, dt = \int_{-\infty}^{\infty} |\mathcal{F}f(s)|^2\, ds. \tag{2.2.10}$$

2.2.3 Examples

In this paragraph we calculate the Fourier Transform of some basic functions used in signal processing.

The Rect function

We have already seen that the Fourier Transform of the rect function given by

$$\Pi(t) = \begin{cases} 1, & |t| \leq 1/2 \\ 0, & |t| > 1/2 \end{cases},$$

is $\mathcal{F}\Pi(s) = \sin(\pi s)/\pi s$.

The Triangle function

The Triangle function is defined by

$$\Lambda(t) = \begin{cases} 1 - |t|, & |t| \leq 1 \\ 0, & |t| > 1 \end{cases}.$$

In Figure 2.15 you can see the graph of this function.

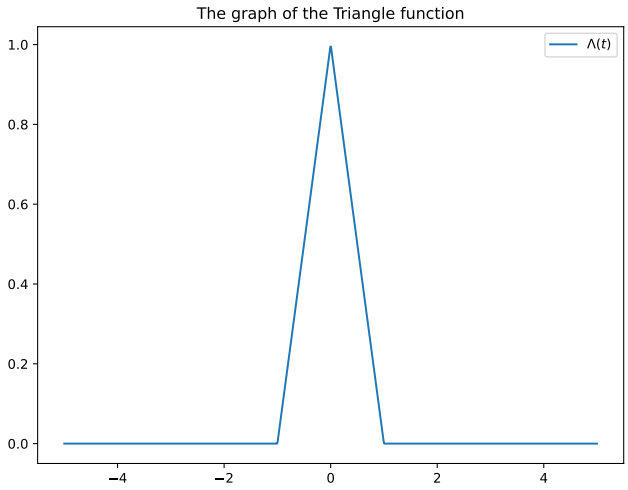

Figure 2.15. *The graph of the Triangle function on the interval* $[-5, 5]$.

Since Λ is even we can easily calculate the Fourier Transform as follows:

$$\begin{aligned}
\mathcal{F}\Lambda(s) &= \int_{-\infty}^{\infty} \Lambda(t) e^{-2\pi i s t}\, dt = \int_{-1}^{1} (1-|t|)\, e^{-2\pi i s t}\, dt \\
&= 2\int_{0}^{1} (1-t)\cos(2\pi s t)\, dt \\
&= \frac{2}{2\pi s}\Big[(1-t)\sin(2\pi s t)\Big]_0^1 + \frac{1}{\pi s}\int_0^1 \sin(2\pi s t)\, dt \\
&= \frac{1}{2(\pi s)^2}\Big[-\cos(2\pi s t)\Big]_0^1 = \frac{1}{(\pi s)^2}\frac{(1-\cos(2\pi s))}{2}.
\end{aligned}$$

Using the identity $\frac{1-\cos(2\pi s)}{2} = \sin^2(\pi s)$ we conclude that

$$\mathcal{F}\Lambda(s) = \frac{1}{(\pi s)^2}\sin^2(\pi s).$$

Usually $\sin(\pi s)/\pi s$ is denoted as $\mathrm{sinc}(s)$. With this notation, we can write

$$\mathcal{F}\Pi(s) = \mathrm{sinc}(s), \quad \mathcal{F}\Lambda(s) = \mathrm{sinc}^2(s),$$

for the Fourier Transform of the rect and triangle function, respectively. In Figure 2.16 you can see the graph of $\mathrm{sinc}^2(s)$.

Exponential decay

The Exponential decay is defined by

$$f(t) = \begin{cases} e^{-at}, & t > 0 \\ 0, & t \le 0 \end{cases},$$

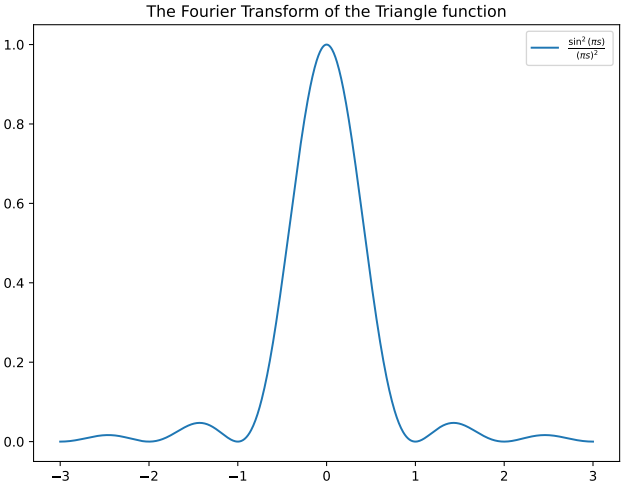

Figure 2.16. *The Fourier Transform of the Triangle function.*

for $a > 0$. In Figure 2.17 you can see the graph of the Exponential decay for $a = 0.25, 0.5, 1, 2$. As we can see large values for a result in a more rapid decrease as t tends to infinity. To calculate the Fourier Transform we have that

$$\mathcal{F}f(s) = \int_{-\infty}^{\infty} e^{-at} e^{-2\pi i s t}\, dt = \int_{0}^{\infty} e^{-(a+2\pi i s)t}\, dt$$
$$-\frac{1}{a+2\pi i s}\left[e^{-(a+2\pi i s)t}\right]_0^{\infty} = \frac{1}{a+2\pi i s}.$$

We observe that in this case, the values $\mathcal{F}f(s)$ are complex for $s \neq 0$. In Figure 2.18 we graphically present the power spectrum $|\mathcal{F}f(s)|^2 = \frac{1}{a^2+4\pi^2 s^2}$ of f for each $a = 0.25, 0.5, 1, 2$.

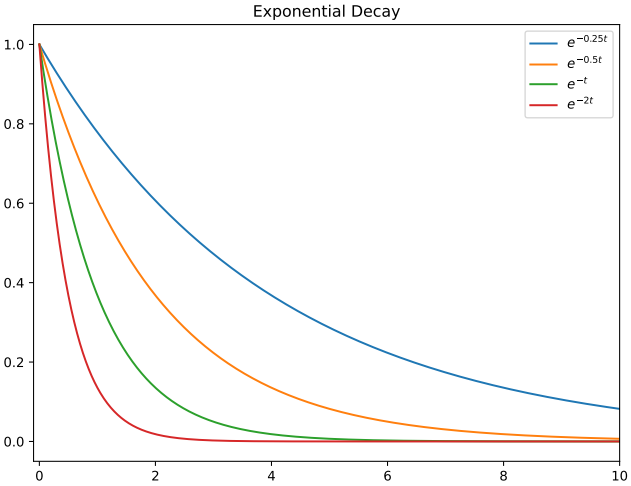

Figure 2.17. *The graph of the Exponential decay for* $a = 0.25, 0.5, 1, 2$.

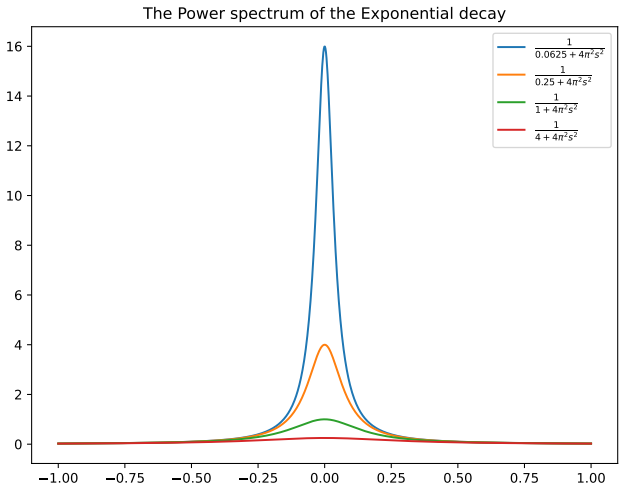

Figure 2.18. *The Power spectrum of the Exponential decay for* $a = 0.25, 0.5, 1, 2$.

2.2.4 Properties and Convolution

In this paragraph we introduce some basic properties of the Fourier Transform as well as the notion of convolution which is extensively used in signal processing.

Duality

The first property is referred to as the *duality* of the Fourier Transform. To this end, calculating the Fourier Transform on $-s$ we get that

$$\mathcal{F}f(-s) = \int_{-\infty}^{\infty} f(t)e^{-2\pi i(-s)t}\,dt = \int_{-\infty}^{\infty} f(t)e^{2\pi ist}\,dt.$$

The last integral is exactly the Inverse Fourier Transform of f on s, i.e. $\mathcal{F}f(-s) = \mathcal{F}^{-1}f(s)$. Similarly, calculating the Inverse Fourier Transform of f at $-t$ we get that

$$\mathcal{F}^{-1}f(-t) = \int_{-\infty}^{\infty} f(s)e^{2\pi is(-t)}\,ds = \int_{-\infty}^{\infty} f(s)e^{-2\pi ist}\,ds.$$

Since the last integral is equal to $\mathcal{F}f(t)$ we conclude that $\mathcal{F}^{-1}f(-t) = \mathcal{F}f(t)$. Summarizing the above equations we get the *duality* of the Fourier Transform:

$$\mathcal{F}f(-s) = \mathcal{F}^{-1}f(s), \quad \mathcal{F}^{-1}f(-t) = \mathcal{F}f(t). \tag{2.2.11}$$

The duality is a useful property that can be used in some cases to determine the Fourier Transform. For instance, we know that $\mathcal{F}\Pi = \text{sinc}$, and hence $\mathcal{F}^{-1}\text{sinc} = \Pi$. By duality we have that

$$\mathcal{F}\text{sinc}(t) = \mathcal{F}^{-1}\text{sinc}(-t) = \Pi(-t).$$

Since $\Pi(-t) = \Pi(t)$ we get that $\mathcal{F}\text{sinc} = \Pi$. The next property involves the reversed $f^-(t) = f(-t)$

signal of f. Its Fourier Transform is given by

$$\mathcal{F}f^{-}(s) = \int_{-\infty}^{\infty} f^{-}(t)e^{-2\pi ist}\,dt = \int_{-\infty}^{\infty} f(-t)e^{-2\pi ist}\,dt \overset{u=-t}{=} \int_{-\infty}^{\infty} f(u)e^{2\pi isu}\,du.$$

But since the last integral is equal to $\mathcal{F}^{-1}f(s)$ we get that $\mathcal{F}f^{-}(s) = \mathcal{F}^{-1}f(s)$.

Shifting and stretching

The next two properties involve the behavior of the Fourier Transform with respect to shifting and stretching operations on a function. For $a \in \mathbb{R}$, we consider the translation f_a of f given by $f_a(t) = f(t + a)$. Stretching f by a corresponds to the function $f^a(t) = f(at)$. The Fourier Transform of f_a is given by

$$\mathcal{F}f_a(s) = \int_{-\infty}^{\infty} f_a(t)e^{-2\pi ist}\,dt = \int_{-\infty}^{\infty} f(t + a)e^{-2\pi ist}\,dt.$$

Substituting $u = t + a$ we can write

$$\int_{-\infty}^{\infty} f(t + a)e^{-2\pi ist}\,dt = \int_{-\infty}^{\infty} f(u)e^{-2\pi is(u-a)}\,du = e^{2\pi ias}\int_{-\infty}^{\infty} f(u)e^{-2\pi ius}\,du.$$

Observing that the last integral is the Fourier Transform of f at s we get that

$$\mathcal{F}f_a(s) = e^{2\pi ias}\mathcal{F}f(s). \tag{2.2.12}$$

To calculate the Fourier Transform of $f^a(t) = f(at)$ we first assume that $a > 0$ and use the substitution $u = at$. Therefore, we have that

$$\begin{aligned}\mathcal{F}f^a(s) &= \int_{-\infty}^{\infty} f^a(t)e^{-2\pi ist}\,dt = \int_{-\infty}^{\infty} f(at)e^{-2\pi ist}\,dt \\ &= \frac{1}{a}\int_{-\infty}^{\infty} f(u)e^{-2\pi isu/a}\,du = \frac{1}{a}\mathcal{F}f\left(\frac{u}{a}\right).\end{aligned}$$

For $a < 0$ we get

$$\begin{aligned}\mathcal{F}f^a(s) &= \int_{-\infty}^{\infty} f^a(t)e^{-2\pi ist}\,dt = \frac{1}{a}\int_{\infty}^{-\infty} f(u)e^{-2\pi isu/a}\,du \\ &= -\frac{1}{a}\int_{-\infty}^{\infty} f(u)e^{-2\pi iu/a}\,du = \frac{1}{|a|}\mathcal{F}f\left(\frac{u}{a}\right).\end{aligned}$$

Merging the two cases into one equation, we conclude that for every $a \in \mathbb{R}$

$$\mathcal{F}f^a(s) = \frac{1}{|a|}\mathcal{F}f\left(\frac{u}{a}\right). \tag{2.2.13}$$

Linearity

Another way to view the Fourier Transform is as an operator acting on the space of functions $f : \mathbb{R} \to \mathbb{R}$ for which $\int_{-\infty}^{\infty} |f(t)|\,dt < \infty$. That is, for each f the operator $\mathcal{F}$ maps f to the function $\mathcal{F}f$. An important property of this operator is that is linear, i.e. for every integrable pair $f, g : \mathbb{R} \to \mathbb{R}$

and $a, \beta \in \mathbb{R}$ we have that

$$\mathcal{F}(af + \beta g) = a\mathcal{F}f + \beta\mathcal{F}g. \tag{2.2.14}$$

To verify (2.2.14) we let $s \in \mathbb{R}$ and write

$$\begin{aligned}\mathcal{F}(af + \beta g)(s) &= \int_{-\infty}^{\infty} (af(t) + \beta g(t))\, e^{-2\pi i s t}\, dt \\ &= a\int_{-\infty}^{\infty} f(t)e^{-2\pi i s t}\, dt + \beta\int_{-\infty}^{\infty} g(t)e^{-2\pi i s t}\, dt \\ &= a\mathcal{F}f(s) + \beta\mathcal{F}g(s).\end{aligned}$$

In a similar manner, it easily verified that the Inverse Fourier Transform is also linear, i.e. $\mathcal{F}^{-1}(af + \beta g) = a\mathcal{F}^{-1}f + \beta\mathcal{F}^{-1}g$, for every $f, g : \mathbb{R} \to \mathbb{R}$ and $a, \beta \in \mathbb{R}$. The way that linearity is used in signal processing is when one needs to denoise a signal. A noisy signal $h(t)$ can be represented as the sum of the signals $f(t), g(t)$, where $f(t)$ corresponds to the clean signal and $g(t)$ to the noisy one. The goal is to denoise the signal $h(t) = f(t) + g(t)$ by removing the part $g(t)$ consisting of the noisy signal. Applying the Fourier Transform and using the linearity we get that

$$\mathcal{F}h(s) = \mathcal{F}(f + g)(s) = \mathcal{F}f(s) + \mathcal{F}g(s).$$

Therefore, the problem of reducing the noise of the signal $h(t)$ corresponds to removing the frequencies $\mathcal{F}g(s)$ on the frequency domain.

Convolution and convolution theorem

As we have seen so far, the previous properties allow us to modify a signal by using another signal. For instance, if we want to add some frequencies to a signal $f(t)$ described by the function $g(s)$ we can define $h(s) = \mathcal{F}f(s) + g(s)$, and by taking the Inverse Fourier Transform $\mathcal{F}^{-1}h$ we can end up with the desired modified signal. The convolution operation involves the scaling of the harmonics $\mathcal{F}f(s)$ by a varying function $\mathcal{F}g(s)$. In other words, we are interested in determining the operation between the signals $f(t),\ g(t)$ on the time domain for which the Fourier Transform of the resulting signal is $\mathcal{F}f(s) \cdot \mathcal{F}g(s)$. To this end, we can proceed with reverse engineering by starting from the Fourier Transform $\mathcal{F}f(s) \cdot \mathcal{F}g(s)$ and using Fubini's theorem to change the order of the integrals:

$$\begin{aligned}\mathcal{F}f(s) \cdot \mathcal{F}g(s) &= \left(\int_{-\infty}^{\infty} f(t)e^{-2\pi i s t}\, dt\right) \cdot \left(\int_{-\infty}^{\infty} g(x)e^{-2\pi i x s}\, dx\right) \\ &= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(t)g(x)e^{-2\pi i s(x+t)}\, dt\, dx \\ &= \int_{-\infty}^{\infty}\left(\int_{-\infty}^{\infty} f(t)e^{-2\pi i s(x+t)}\, dt\right) g(x)\, dx\end{aligned}$$

Now, using the substitution $u = x + t$ we can write

$$\begin{aligned}\int_{-\infty}^{\infty}\left(\int_{-\infty}^{\infty} f(t)e^{-2\pi i s(x+t)}\, dt\right) g(x)\, dx &= \int_{-\infty}^{\infty}\left(\int_{-\infty}^{\infty} f(u-x)e^{-2\pi i s u}\, du\right) g(x)\, dx \\ &= \int_{-\infty}^{\infty}\left(\int_{-\infty}^{\infty} f(u-x)g(x)\, dx\right) e^{-2\pi i s u}\, du.\end{aligned}$$

If we define $h(u) = \int_{-\infty}^{\infty} f(u-x)g(x)\,dx$ we observe that the last integral is exactly the Fourier Transform of h at s. Therefore, if we want to somehow combine the two signals $f(t)$, $g(t)$ on the time domain to obtain a signal $h(t)$ for which $\mathcal{F}h(s) = \mathcal{F}f(s) \cdot \mathcal{F}g(s)$ we are bound to define

$$h(t) = \int_{-\infty}^{\infty} f(t-x)g(x)\,dx.$$

The previous operation is the convolution of f with g. Below we state the formal definition.

Definition 2.9 (Convolution)**.** *Let $f, g : \mathbb{R} \to \mathbb{R}$ be two functions for which $\int_{-\infty}^{\infty} |f(t)|\,dt < \infty$, $\int_{-\infty}^{\infty} |g(t)| < dt < \infty$. The convolution of f, g is the function $f * g$ given by*

$$(f * g)(t) = \int_{-\infty}^{\infty} f(t-x)g(x)\,dx. \tag{2.2.15}$$

Based on the upon definition the Convolution theorem is summarized as follows:

Theorem 2.3 (Convolution Theorem)**.** *Let $f, g : \mathbb{R} \to \mathbb{R}$ be two functions such that $\int_{-\infty}^{\infty} |f(t)|\,dt < \infty$, $\int_{-\infty}^{\infty} |g(t)|\,dt < \infty$. Then,*

$$\mathcal{F}(f * g)(s) = \mathcal{F}f(s) \cdot \mathcal{F}g(s),$$

for $s \in \mathbb{R}$.

2.2.5 Filtering

Filtering is a generic term used in signal processing when one needs to apply a transformation to a signal to modify it. For instance, one may want to eliminate some frequencies of the signal to reduce its noisy parts and keep the clean signal. In this paragraph, with the help of the Fourier Transform, we present some of the most essential filters used in signal processing. Following the conventions used in courses on signals and systems, we denote the Fourier Transform of a function f by its corresponding capital letter F.

Lowpass filters

The goal of a lowpass filter is to discard all of the frequencies of a signal that exceed a threshold $v_c > 0$. On the frequency domain, this corresponds to the multiplication of the Fourier Transform F of the original signal f by the function H given by

$$H(s) = \begin{cases} 1, & |s| < v_c \\ 0, & |s| \geq v_c \end{cases}. \tag{2.2.16}$$

In Figure 2.19 you can see the graph of $H(s)$. By using the Inverse Fourier Transform we can determine the signal h that needs to be convolved with the original signal f to get the desired output signal that contains no frequencies higher than v_c. To this end, observe that H is a stretched

rect function by a factor of $1/2\upsilon_c$. To see this, write

$$\Pi(s/2\upsilon_c) = \begin{cases} 1, & \frac{|s|}{2\upsilon_c} < \frac{1}{2} \\ 0, & \frac{|s|}{2\upsilon_c} \geq \frac{1}{2} \end{cases} = \begin{cases} 1, & |s| < \upsilon_c \\ 0, & |s| \geq \upsilon_c \end{cases} = H(s).$$

Now, using the stretching property in (2.2.13) and the linearity of the Fourier Transform (it's Inverse actually) we get that

$$\mathcal{F}^{-1}H = \mathcal{F}^{-1}\Pi(\cdot/2\upsilon_c) = 2\upsilon_c\mathcal{F}^{-1}\Pi(2\upsilon_c t) = 2\upsilon_c \text{sinc}(2\upsilon_c t).$$

Therefore, the desired output signal is given by $y(t) = (f * h)(t)$, where $h(t) = 2\upsilon_c \text{sinc}(2\upsilon_c t)$.

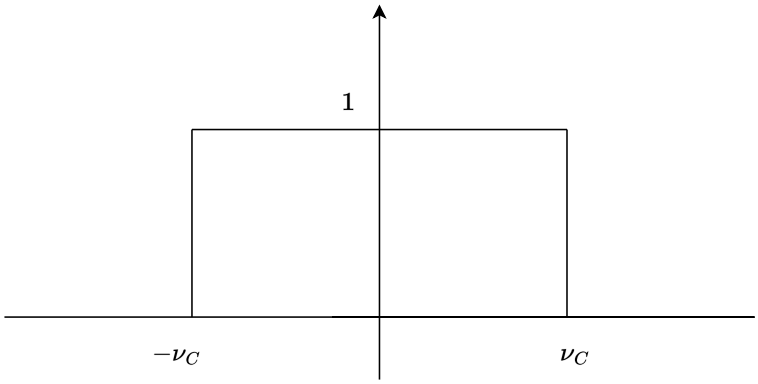

Figure 2.19. *Lowpass Filter.*

Bandpass filters

The Bandpass filters allow a particular band of frequencies to pass without any modification by eliminating all the others. This is described by the multiplication of the Fourier Transform F of the original signal f with the function

$$B(s) = \begin{cases} 1, & \upsilon_0 - \upsilon_c < |s| < \upsilon_0 + \upsilon_c \\ 0, & \text{otherwise} \end{cases}. \tag{2.2.17}$$

The graph of $B(s)$ is shown in 2.20. To find the Inverse Fourier Transform we can express $B(s)$ as a function of the rect function Π. It is easily verified that $B(s) = \Pi\left(\frac{s+\upsilon_0}{2\upsilon_c}\right) + \Pi\left(\frac{s-\upsilon_0}{2\upsilon_c}\right)$. Therefore,

$$\begin{aligned} \mathcal{F}^{-1}B(t) &= \mathcal{F}^{-1}\Pi^{1/2\upsilon_c}_{\upsilon_0}(t) + \mathcal{F}^{-1}\Pi^{1/2\upsilon_c}_{-\upsilon_0}(t) \\ &= 2\upsilon_c \cdot e^{2\pi i\upsilon_0 t} \cdot \mathcal{F}^{-1}\Pi(t) + 2\upsilon_c \cdot e^{-2\pi i\upsilon_0} \cdot \mathcal{F}^{-1}\Pi(t) \\ &= 4\upsilon_c \cos(2\pi\upsilon_0 t)\text{sinc}(2\upsilon_c t). \end{aligned} \tag{2.2.18}$$

Therefore, to apply the bandpass filter to the signal f we need to convolve f with the Inverse Fourier Transform of B in (2.2.18). In Figure 2.21 we can see the graph of the Inverse Fourier Transform of B for $\upsilon_0 = 10$ and $\upsilon_c = 2$.

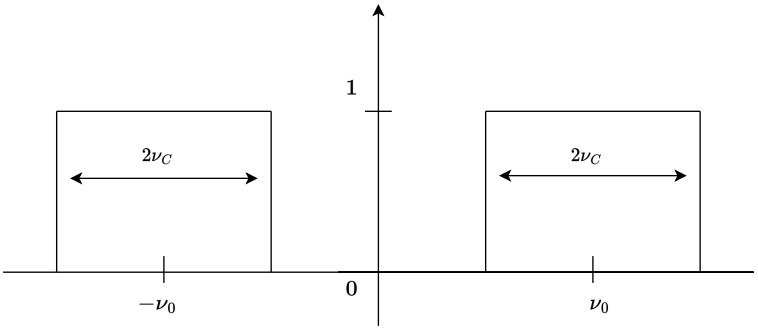

Figure 2.20. *Bandpass Filter.*

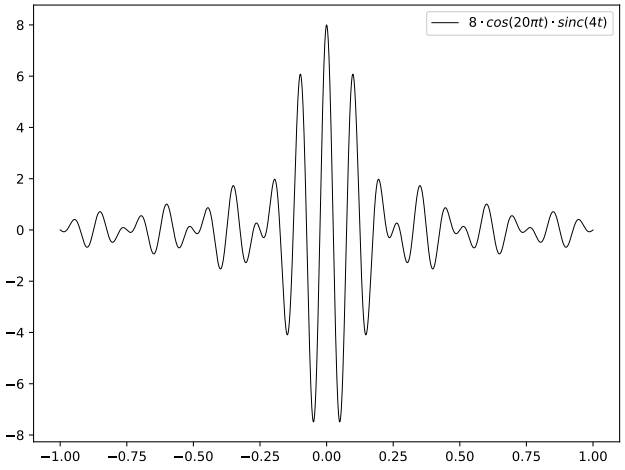

Figure 2.21. *The Inverse Fourier Transform of* $B(s)$ *for* $v_0 = 10,\ v_c = 2$.

Highpass filters

As opposed to the lowpass filters the highpass filters pass only the frequencies above a certain threshold while discarding the others. The function describing this family of filters on the frequency domain is given by

$$\text{High}(s) = 1 - \Pi\left(\frac{s}{2v_c}\right), \tag{2.2.19}$$

where v_c is the cutoff frequency. Figure 2.22 shows the graph of High(s).

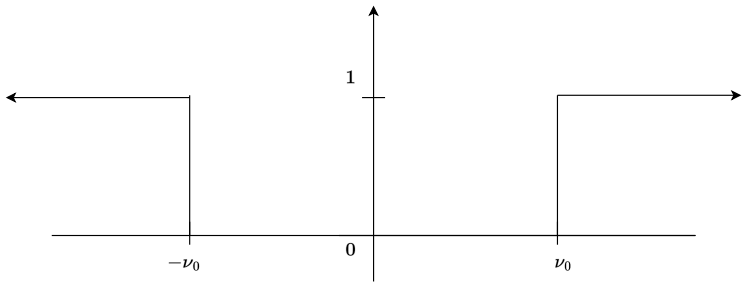

Figure 2.22. *The Highpass filter.*

The calculation of the Inverse Fourier Transform involves the calculation of the Inverse Fourier Transform of the constant function 1. With the theory we have presented so far we can't calculate the Inverse Fourier Transform of this function since it is not integrable. One needs to extend the notion of the Fourier Transform to include such functions. The proper way to achieve this is by defining the Fourier Transform on a proper class of functions. This class is known as the *Schwartz* functions $\mathcal{S}$. Then, the notion of the Fourier Transform is extended to the space of linear functionals $T : \mathcal{S} \to \mathbb{R}$. We will not cover this approach in this thesis as in the upcoming sections of this chapter we will focus on the "discretized" theory of the Fourier Transform which is what is used by computer machines to process a signal. The reader may refer to [4] for a nice exposition of this approach. With this extension of the definition one has

$$\mathcal{F}1 = \delta, \quad \mathcal{F}\delta = 1,$$

where δ is the linear functional $\delta : \mathcal{S} \to \mathbb{R}$ for which $\delta(f) = f(0)$, for $f \in \mathcal{S}$. In engineering courses the linear functional δ is introduced as a function $\delta : \mathbb{R} \to \mathbb{R}$ for which

$$\delta(t) = \begin{cases} +\infty, & t = 0 \\ 0, & t \neq 0 \end{cases},$$

and $\int_{-\infty}^{\infty} \delta(t)\, dt = 1$. In this context, δ is referred to as the *unit impulse* or *Dirac delta* function. One important property of δ is that

$$\int_{-\infty}^{\infty} f(t-x)\delta(x)\, dx = f(t), \tag{2.2.20}$$

for every $f \in \mathcal{S}$ and $t \in \mathbb{R}$. In other words, $(f * \delta)(t) = (\delta * f)(t) = f(t)$. The identity in (2.2.20) is just a conventional notation that results from the fact that the extended Fourier Transform is first defined for the linear functionals $T : \mathcal{S} \to \mathbb{R}$ for which

$$T(f) = \int_{-\infty}^{\infty} f(x)g(x)\, dx, \tag{2.2.21}$$

for some $g \in \mathcal{S}$. It is not true that every linear function can be expressed in the form of (2.2.21), δ is one of these. However, introducing δ as a function allows us to think of δ as the linear functional $\delta : \mathcal{S} \to \mathbb{R}$ for which

$$\delta(f) = \int_{-\infty}^{\infty} f(x)\delta(x)\, dx,$$

for every $f \in \mathcal{S}$. In this case, the relation $\delta f = f(0)$ can be equivalently expressed as $(f * \delta)(0) = f(0)$. By considering translations of δ one also has (2.2.20). With these definitions, we have that

$$\mathcal{F}^{-1}\mathrm{High}(t) = \delta(t) - 2\upsilon_c \mathrm{sinc}\,(2\pi\upsilon_c t)\,. \tag{2.2.22}$$

Therefore, the output signal for which all frequencies in $[-\upsilon_c,\ \upsilon_c]$ have been discarded is given by $f*(\delta - h) = f*\delta - f*h = f - f*h$, where f is the original input signal and $h(t) = 2\upsilon_c \mathrm{sinc}\,(2\pi\upsilon_c t)\,.$

2.3 The Discrete Fourier Transform

In this section we introduce the discrete analog of Fourier's theory which is used by computer machines to process a signal. This is necessary since computers can only store a finite number of values. The first step is to find an appropriate way to represent a function, or equivalently a Continuous-Time Signal (CT) $f : \mathbb{R} \to \mathbb{R}$ in a form that can be processed by a computer machine. To this end, the technique that is used is known as *equidistant sampling* : Given an analog signal $f : \mathbb{R} \to \mathbb{R}$ and a period $T > 0$ measured in seconds, one considers the Discrete Time Signal (DT) $x : \mathbb{Z} \to \mathbb{R}$ given by

$$x(n) = f(nT), \quad n \in \mathbb{Z}. \tag{2.3.1}$$

The reciprocal $F_s = 1/T$ is the *sampling rate* and corresponds to the number of equally-spaced samples obtained from f in a period of 1 second. Larger values of the sampling rate correspond to a higher resolution. Figure 2.23 shows an equidistant sampling for the signal obtained from a piano A4 key. The first figure corresponds to a sampling of 4000 Hz while the second to a sampling of 8000 Hz.

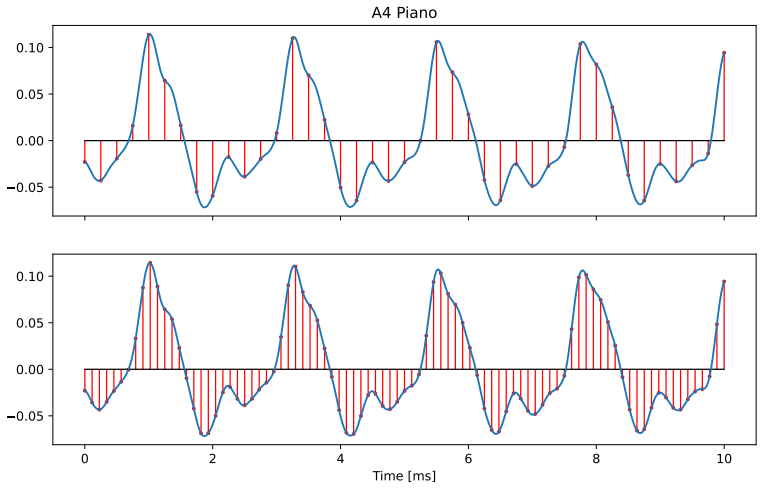

Figure 2.23. *Equidistant sampling of Piano A4 with sampling rates 4KHz (top) and 8KHz (bottom).*

Once we have a discrete analog of the original signal the next step is to define the discrete analog of the Fourier Transform. By looking at the definition in (2.2.6) the problem boils down on how to handle the integration. The approach used in this case is again an approximation via Riemann sums. To this end, since the equidistant sampling partition's the real numbers $\mathbb{R}$ as

$$\mathbb{R} = \bigcup_{n=-\infty}^{\infty} [nT, (n+1)T],$$

we can approximate the integral in (2.2.6) as

$$\int_{-\infty}^{\infty} f(t)e^{-2\pi ist}\, dt \approx T \cdot \sum_{n=-\infty}^{\infty} f(nT)e^{-2\pi isnT}. \tag{2.3.2}$$

Therefore, instead of calculating the Fourier Transform $\mathcal{F}f(s)$ by integration we can use the approximation in (2.3.2). One defines a discrete version of the Fourier Transform for a given Discrete-Time Signal $x : \mathbb{Z} \to \mathbb{R}$ by setting

$$\hat{x}(s) = \sum_{n=-\infty}^{\infty} x(n)e^{-2\pi isn}. \tag{2.3.3}$$

In this definition, where a simple 1-sampling of the exponential function is used, one does not assume that one knows the relation between x and the original signal f. In reality, the computer stores an array consisting of the values $\{x(n) : n \in \mathbb{Z}\}$[7] and then the Fourier Transform is calculated via (2.3.3). If one needs to know the relation between $X(s)$ and $F(s)$, where $F(s)$ is the Fourier Transform of the original analog signal f at s, then one needs to know the sampling period T. By (2.3.2), (2.3.3) we see that

$$\hat{x}(s) \approx \frac{1}{T}F\left(\frac{s}{T}\right). \tag{2.3.4}$$

Therefore, the complex number $X(s)$ is an approximation of the Fourier Transform of the original signal f corresponding to the frequency s/T scaled by a factor of T. At this point, it should be obvious that the approach we have adopted so far is an approximation to the original Fourier Transform. How well is the approximation, is reflected through the sampling period $T > 0$. The smaller the better. However, we have not mentioned anything yet about which T is the suitable one. To this end, an answer is provided by the sampling theorem which says that if the signal does not contain any frequencies higher than $\Omega = F_s/2 = 1/(2T)$[8] Hz, then the signal f can be reconstructed perfectly from its sampled version $x : \mathbb{Z} \to \mathbb{R}$ given by (2.3.1). Formally, we have the following theorem.

Theorem 2.4 (Sampling Theorem)**.** *Let $f : \mathbb{R} \to \mathbb{R}$ be an Ω-bandlimited CT signal. I.e., $F(s) = 0$ for every $|s| > \Omega$, where $F : \mathbb{R} \to \mathbb{C}$ is the Fourier Transform of f. Furthermore, suppose that f is square integrable*

$$\int_{-\infty}^{\infty} |f(t)|^2 \, dt < \infty.$$

Let $x : \mathbb{Z} \to \mathbb{R}$ be the T-sampled version of f with $T = 1/(2\Omega)$, i.e. $x(n) = f(nT)$, $n \in \mathbb{Z}$. Then f can be reconstructed from x by

$$f(t) = \sum_{n=-\infty}^{\infty} x(n) \cdot sinc\left(\frac{t - nT}{T}\right) = \sum_{n=-\infty}^{\infty} f\left(\frac{n}{2\Omega}\right) \cdot sinc\left(2\Omega t - n\right), \tag{2.3.5}$$

for every $t \in \mathbb{R}$. In other words, the CT-signal f can be perfectly reconstructed from the DT-signal obtained by equidistant sampling if the bandlimit is no greater than half the sampling rate.

Proof. First note that we may assume without loss of generality that $\Omega = 1/2$ by considering the scaled function $t \mapsto f(t/2\Omega)$. In this case, the Fourier Transform $F : \mathbb{R} \to \mathbb{C}$ is zero outside the interval $[-1/2, 1/2]$. Therefore, we can extend F to a periodic function $g : \mathbb{R} \to \mathbb{R}$ with fundamental period $1/2$. Since f is square integrable by Parserval's identity in (2.2.10), g would

[7] At this point we let $n \in \mathbb{Z}$ but soon we will restrict to a finite number of samples which what the computer stores.
[8] We also say that f is an Ω-bandlimited signal.

be in $\mathcal{L}_2\left(\left[-\frac{1}{2}, \frac{1}{2}\right]\right)$. Hence, we can express g through its Fourier series by

$$g(s) = \sum_{n=-\infty}^{\infty} \hat{g}(n) e^{2\pi i n s}. \tag{2.3.6}$$

Also, by Fourier inversion theorem we have that

$$f(t) = \int_{-\infty}^{\infty} F(s) e^{2\pi i s t}\, ds = \int_{-1/2}^{1/2} F(s) e^{2\pi i s t}\, ds. \tag{2.3.7}$$

Now, for the Fourier coefficient $\hat{g}(n)$ we have that

$$\hat{g}(n) = \int_{-1/2}^{1/2} g(s) e^{-2\pi i n s}\, ds = \int_{-1/2}^{1/2} F(s) e^{-2\pi i n s}\, ds. \tag{2.3.8}$$

Combining (2.3.7) and (2.3.8) we see that $\hat{g}(n) = f(-n)$ for every $n \in \mathbb{Z}$. Now write

$$\begin{aligned} f(t) &= \int_{-1/2}^{1/2} F(s) e^{2\pi i s t}\, ds = \int_{-1/2}^{1/2} \left(\sum_{n=-\infty}^{\infty} \hat{g}(n) e^{2\pi i n s} \right) \cdot e^{2\pi i s t}\, ds \\ &= \sum_{n=-\infty}^{\infty} \int_{-1/2}^{1/2} f(n) e^{-2\pi i n (s-t)}\, ds = \sum_{n=-\infty}^{\infty} x(n) \int_{-\infty}^{\infty} \Pi(s) e^{-2\pi i s (n-t)}\, ds. \end{aligned}$$

Observing that the last integral is the Fourier Transform of the rect function Π at $(n - t)$ we may write

$$f(t) = \sum_{n=-\infty}^{\infty} x(n) \mathcal{F}\Pi\,(n - t).$$

Now, since $\mathcal{F}\Pi(n - t) = \operatorname{sinc}(n - t) = \operatorname{sinc}(t - n)$ we conclude that

$$f(t) = \sum_{n=-\infty}^{\infty} x(n) \operatorname{sinc}(t - n) = \sum_{n=-\infty}^{\infty} f(n) \operatorname{sinc}(t - n),$$

which exactly the desired relation in (2.3.5) for $T = 1$. □

Assuming that we have some prior knowledge of the original signal's spectrum, the sampling theorem gives us a lower bound on the sampling rate to be used, i.e., if the signal does not contain any frequencies higher than Ω then any sampling rate $F_s \geq 2\Omega$ would suffice to perfectly reconstruct the original signal f by its sampled version $x(n) = f(nT)$. For a sampling rate F_s the lower bound $F_s/2$ is referred to as the *Nyquist frequency*. In case f contains higher frequencies, sampling may cause artifacts referred to as *aliasing*.

Now, there are still two problems concerning the formula in (2.3.3). The first one is the infinite sum, and the second one is that the parameter s corresponding to the frequency values is a continuous parameter. For the first problem, in practice one deals with signals of finite duration. Hence, in a computer machine, a signal is stored as a finite sequence of points $x(0), x(1), \ldots, x(N-1)$. Regarding the second problem, one computes the Fourier Transform for only a finite number of frequencies M. Similarly to the discretization of the x-axis the frequencies s/M are considered for every $s = 0, 1 \ldots, M - 1$. In practice, M is equal to N which leads to the definition of Discrete

Fourier Transform (DFT) which is calculated by digital machines:

$$X(k) = \hat{x}(k/N) = \sum_{n=0}^{N-1} x(n) e^{-2\pi i k n/N}, \tag{2.3.9}$$

for every $k = 0, \dots, N-1$. By (2.3.4) we see that

$$X(k) = \hat{x}(k/N) \approx \frac{1}{T} F\left(\frac{k}{NT}\right). \tag{2.3.10}$$

In other words, the k-th Fourier coefficient of the DT-signal x corresponds to the k/NT-th frequency of the original signal f scaled by a factor of T. Now, in signal processing we are interested in the power spectrum $\{|X(k)|^2 : k = 0, 1 \dots, N-1\}$ of the sampled signal and we omit the information about the phase. Since for real-valued signals we have that $X(k) = \overline{X(N-k)}$ for every $k = 0, \dots, N-1$ we only need to consider the coefficients for $k = 0, \dots, [N/2]$. To see this, write

$$\begin{aligned}
\overline{X(N-k)} &= \overline{\hat{x}\left(\frac{N-k}{N}\right)} = \overline{\sum_{n=0}^{N-1} x(n) e^{-2\pi i n(N-k)/N}} \\
&= \sum_{n=0}^{N-1} \overline{x(n)} \cdot \overline{e^{-2\pi i n} \cdot e^{2\pi i n k/N}} \\
&= \sum_{n=0}^{N-1} x(n) \cdot \cancel{e^{2\pi i n}}^{1} \cdot e^{-2\pi i n k/N}.
\end{aligned}$$

Now, since $X(k) = \sum_{n=0}^{N-1} x(n) e^{-2\pi i n k/N}$ we conclude that $X(k) = \overline{X(n-k)}$ for every $k = 0, \dots, N-1$. Observe that if N is divisible by 2 then $k = N/2$ corresponds to the Nyquist frequency $1/2T$.

2.4 Short-Time Fourier Transform

In the previous section , we introduced the discrete version of the Fourier Transform, which is used by digital machines to process a signal. The DFT yields frequency information that is averaged over the entire time domain of the input signal. However, the information about when these frequencies occur is hidden. In 1946 Dennis Gabor introduced the Short-Time Fourier Transform (STFT) to recover this hidden information. The idea is simple. We apply the DFT in short-time windows and slide these windows until we get every part of the signal. One decides the length N of the window function referred to as the *window size* and the length H of the sliding window referred to as the *hop length*. There are many choices for the window function and its choice has its effects on the signal. One choice is the *rectangular window* $w : [0, N-1] \to \mathbb{R}$ given by

$$w(n) = \begin{cases} 1, & n \in [0, N-1] \\ 0, & n \notin [0, N-1] \end{cases}.$$

Then, given a DT-signal $x : \mathbb{Z} \to \mathbb{R}$ the discrete Short-Time Fourier Transform $\mathcal{X}$ of the signal x is given by

$$\mathcal{X}(m,k) = \sum_{n=0}^{N-1} x(n+mH)w(n)e^{-2\pi ink/N}, \tag{2.4.1}$$

for $m \in \mathbb{Z}$, and $k = 0, \ldots, N-1$. Observe that the STFT has two variables, the m corresponding to the time frame and k to the k-th Fourier coefficient for that time frame. If x is obtained by an equidistant sampling from a CT-signal f with a sampling rate F_s then the m-th frame corresponds to the physical time position mH/F_s given in seconds. Small values of hop length H result in a dense sequence of spectral vectors which corresponds to a huge increase in data volume and redundancy. For these reasons, one usually relates the hop length H to the window size N. Common values are $H = N/2$ or $H = N/4$, depending on the nature of the application. Figure 2.24 illustrates this idea with a concrete example. In the first figure we see the waveform of the signal $f(t) = \mathbb{1}_{0\leq t<5} \cdot \sin(2\pi t) + \mathbb{1}_{5\leq t\leq 10} \cdot \sin(10\pi t)$. The first 5 seconds the signal f is equal to a sinusoid with fundamental frequency of 1 Hz, while in the last 5 seconds it is equal to sinusoid with associated frequency of 5 Hz. A window size corresponding to a duration of 4 seconds is applied, with a hop size of 2 seconds. As we can see in the second figure which focuses on the part of the signal corresponding to the time interval $[1, 5]$, we observe that the frequency exhibiting the highest energy is indeed the one corresponding to 1 Hz. In the third figure where both frequencies are contained in the window segment, we see on the figure of the power spectrum that this is reflected since the highest values occur for the frequencies corresponding to 1 Hz and 5 Hz. Finally, in the last figure, we see that only the frequency corresponding to 5 Hz is present since the segment does not contain the part of the sinusoid with the fundamental frequency of 1 Hz.

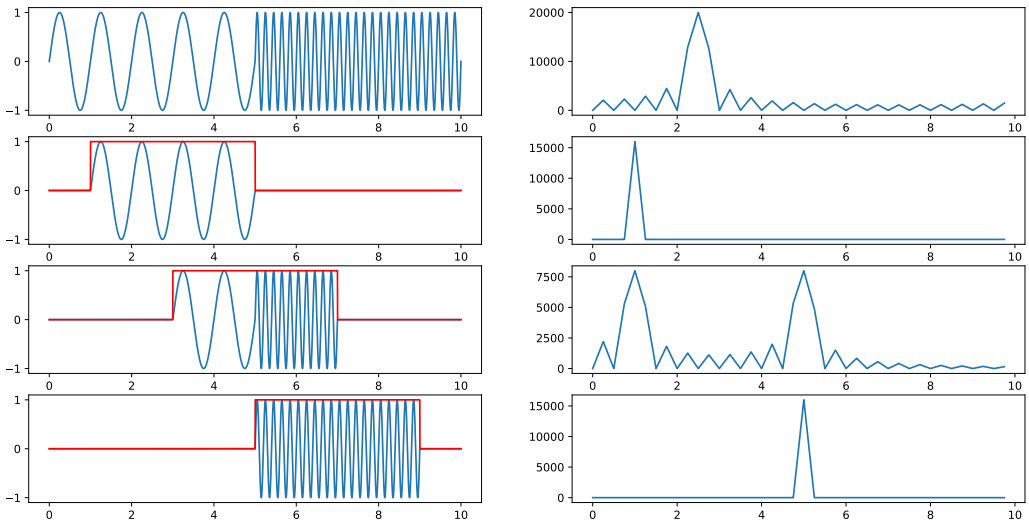

Figure 2.24. *Signal and Fourier Transform consisting of two subsequent sinusoids of frequency* 1 *Hz and* 5 *Hz. The first figure shows the original signal. The second figure shows the windowed signal centered at* $t = 3$*. The third figure shows the windowed signal centered at* $t = 5$*. The last figure shows the windowed signal centered at* $t = 7$*. The example is borrowed from [2].*

One may ask where the rest of the ripples in Figure 2.24 apart from the frequencies corresponding to 1 and 5 Hz come from. These ripples reflect the effect of the rectangular window on the original signal and as such, rather than being part of the original signal f, these frequency components come from the properties of the rectangular window. The design of a proper window function boils down to eliminating these ripples. To this end, a window often used in signal processing is the *Hann window* (also known as the *Hanning window*). The Hann window g is given by

$$g(u) = \begin{cases} \frac{1}{L}\cos^2\left(\frac{\pi u}{L}\right), & -\frac{L}{2} \leq u \leq \frac{L}{2} \\ 0, & |u| > \frac{L}{2} \end{cases}. \tag{2.4.2}$$

In Figure 2.25 we see the effect of the Hann window on the signal shown in Figure 2.24. As is evident from the third figure the ripples are discarded resulting in a smoother representation of the power spectrum.

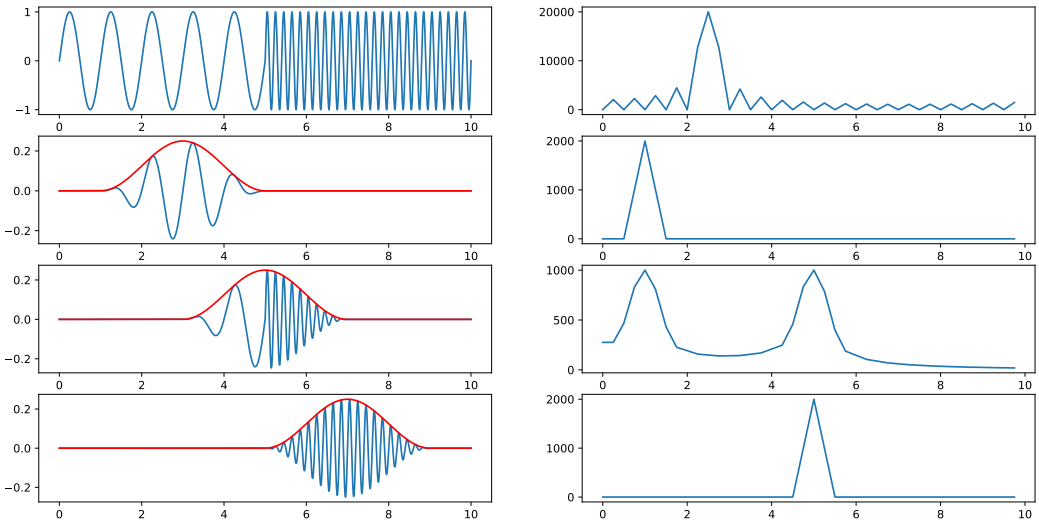

Figure 2.25. *Signal and Fourier Transform consisting of two subsequent sinusoids of frequency* 1 *Hz and* 5 *Hz. The STFT is calculated using the Hann window leading to a smoother representation of the power spectrum.*

At this point, with the theory we have developed so far we can introduce the concept of a *spectrogram* $\mathcal{Y}$, perhaps the most important feature used in audio signal processing. Its richness lies in the fact that it combines the temporal and spectral aspects of a signal at the same time. Furthermore, it can be processed as an image by Convolutional Neural Networks (CNN), a powerful architecture that thrives in image processing nowadays. We will say more about this type of architecture in the subsequent chapters as is the main architecture that we will use for our song identification task. Below we formally define the spectrogram of a DT-signal.

Definition 2.10 (Spectrogram)**.** *Let $x : \mathbb{Z} \to \mathbb{R}$ be a DT-signal and let $\mathcal{X}$ denote the discrete Short-Time Fourier Transform of the signal x for a given window length N and a hop length H. The*

spectrogram $\mathcal{Y}$ of the DT-signal x is the set

$$\mathcal{Y} = \left\{ |X(m,k)|^2 : \ m \in \mathbb{Z}, \ k = 0, \ldots, N-1 \right\}. \tag{2.4.3}$$

We conclude this section with two concrete examples. In Figure 2.4 we see the waveform of an A4 key played on a piano. As we will see in the next chapter, there is a one-to-one correspondence between frequencies and musical pitches. An A4 key corresponds to a pure sinusoid with a fundamental frequency of 440 Hz. This is reflected by the high energy in that frequency in Figure 2.26, where the spectrogram of this waveform is shown. As we can see there is a high energy in multiples of the fundamental frequency 440 Hz corresponding to the harmonics.

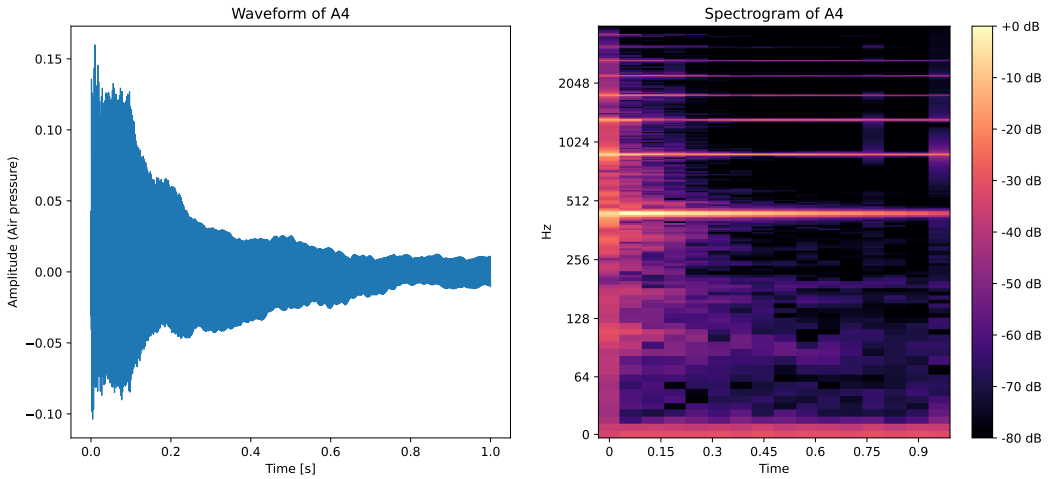

Figure 2.26. *The waveform of A4 key along with its spectrogram representation. The A4 played on piano contains a high degree of the frequency component corresponding to* 440 *Hz. The other contributions visible in the spectrogram correspond to the harmonics of this fundamental frequency, which are integer multiples of* 440 *Hz (i.e.,* 880 *Hz,* 1320 *Hz, etc.).*

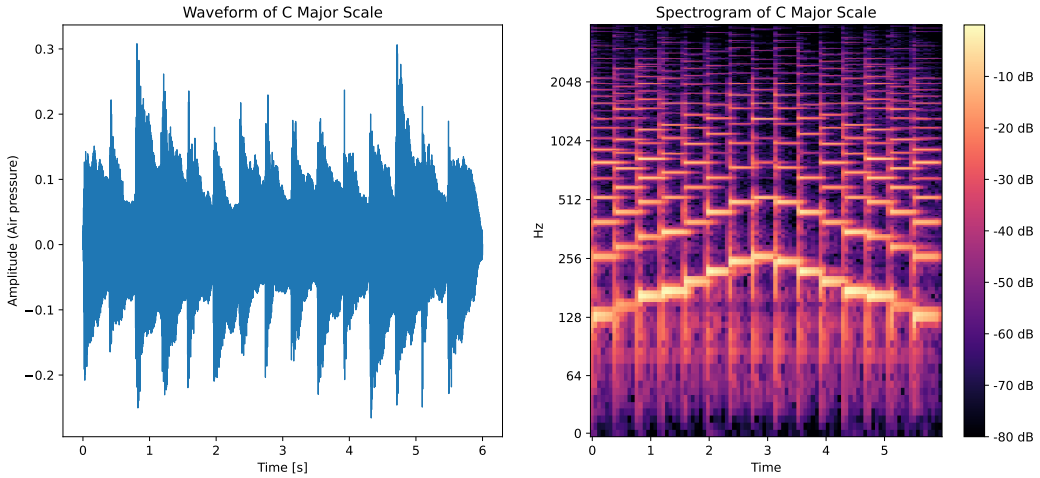

Figure 2.27. *Waveform and spectrogram representation of a C4 major scale played on a Piano. The first note is C3 corresponding to a fundamental frequency of 130 Hz.*

The color bar in Figure 2.26 is shown in a decibel scale with reference point the maximum amplitude present in this spectrogram. This means that 0 dB corresponds to the highest energy. Mathematically this is expressed by the following formula:

$$\mathrm{dB}\left(\mathcal{X}(m,k)\right) = 20 \cdot \log_{10}\left(\frac{|\mathcal{X}(m,k)|}{\mathcal{X}^*}\right),$$

where $\mathcal{X}^* = \max\left\{|\mathcal{X}(m,k)|^2 : m \in \mathbb{Z},\ k = 0,\ldots,N-1\right\}$. Figure 2.27 shows the waveform and the spectrogram of a C4 major scale played on a piano. By looking at the spectrogram we can deduce that exactly 15 notes were played, something which cannot be seen from the waveform representation.

Chapter 3

Audio Signal Processing

In this chapter, we shift from the mathematical theory of Fourier's transform and focus on the most common techniques used in audio signal processing nowadays. We introduce the most important features extracted from the raw audio signals and used in many applications. Much of the content in this chapter is inspired by Müller's well-written book, *Fundamentals of Music Processing: Audio, Analysis, Algorithms, Applications* [2].

3.1 Music Representations

Music can be represented in different ways and formats. In this section, we introduce three different categories of representing a piece of music, or equivalently, a musical sound. These categories are 1) sheet music, 2) symbolic, and 3) audio representations. We start by describing these categories and giving some examples. We mainly focus on the audio representation of music, which allows us to apply mathematical operations and take advantage of the mathematical concepts introduced in the previous chapter.

Sheet Music: A music composition may be written down by a composer in the form of a musical score. In a score, musical symbols are used to visually encode notes and how these notes should be played by a musician. This representation of music is referred to as *sheet music*. The original medium of this representation is paper and constitutes a human-readable representation of music that musicians can share, and by following the symbolic conventions can reproduce a musical composition. In Figure 3.1 you can see an example of a music score corresponding to a part of Beethoven's Moonlight Sonata.

Symbolic music representations: The term *symbolic* refers to any machine-readable data format that explicitly represents musical entities. Some of the most common symbolic representations of music are the Musical Instrument Digital Interface (MIDI) protocol, Piano-Roll representations, and various file formats such as MusicXML. An important and challenging task involving symbolic representations is the *Optical Music Recognition* (OMR) (e.g. [9, 10, 11]), which can be thought of as the equivalent of *Optical Character Recognition* (OCR) in the text domain. As in the case of OCR where the goal is to convert scanned images of printed text into machine-encoded text, the goal of OMR is to convert digital images of sheet music into symbolic music representations such as MIDI or MusicXML.

Audio representations: For our purposes, the most important representation of music is what we call the *audio representation*. This representation corresponds to the interpretation of a musical sound as an acoustic wave, which is transmitted through the air as pressure oscillations. In this way, one can view a musical sound as an audio signal corresponding to the change in air pressure at any given location as a function of time. Since we can view a musical sound as a signal we can take advantage of Fourier's theory and extract valuable information to solve challenging music information retrieval tasks.

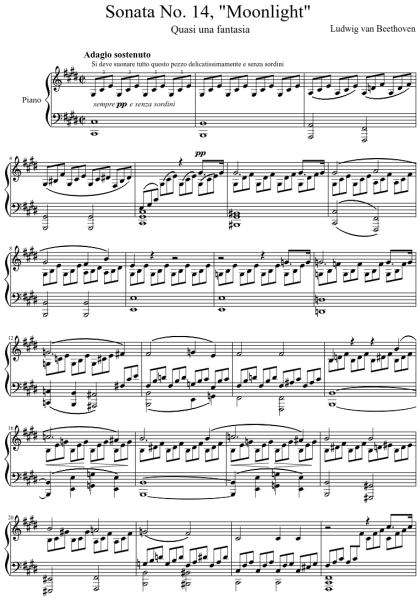

Figure 3.1. *Sheet Score of Beethoven's Moonlight Sonata.*

In definition 2.10 we introduced the concept of a spectrogram. As we will see, variants of this spectral-temporal representation of a musical sound will be used extensively throughout this thesis for the song identification task presented in the upcoming chapters. The most common file formats storing this audio representation of music are Wav and Mp3 files. It is important to stress out at this point that the importance of the audio representation lies in how and when these pressure oscillations occur. In practice, a Wav or an Mp3, when loaded into a computer machine it is stored as an array of finite values $x(0), x(1), \ldots, x(N)$. Typically, these values are represented by a 16-bit integer corresponding to the air pressure at a certain location and time. As discussed in Section 2.3 if one needs to know the time occurrence of $x(n)$ with respect to $x(0)$ one needs to know the sampling rate of the sampling procedure. Typical values of the sampling rate are 8000 Hz, 22050 Hz, and 44100 Hz. In Figures 2.1, 2.4, and 2.27 you can see the waveform of three different audio signals.

3.2 Audio characteristics

3.2.1 Musical Notes, Pitches, and Frequencies

In music, the term note is not strict. It may refer to either a musical symbol (when referring to the score representation of music) or a pitched sound (when talking about audio representations). The

notion of *pitch* is a subjective concept and refers to a perceptual property that allows a listener to order a sound on a frequency-related scale. Playing a note on an instrument more or less results in a periodic sound of a certain fundamental pitch of note. For example, playing the note A4 on a piano results in a sound with a fundamental frequency of 440 Hz. Psychoacoustic experiments have shown that any two notes with fundamental frequencies in a ratio equal to any power of two are perceived as very similar. Because of this perceptual property, one can group the notes into different pitch classes: Given a fundamental frequency of f_0 Hz, its pitch class can be described as the set

$$\mathrm{PC}(f_0) = \left\{ f_0 \cdot 2^k : \ k \in \mathbb{Z} \right\}.$$

This observation leads to the fundamental notion of an *octave*, which is defined to be the interval between one musical note and another with half or double its fundamental frequency. Since in practice one deals with only a finite number of musical symbols, one needs to discretize the set of all possible pitches. This leads to the notion of a *musical scale* which can be thought of as a finite set of representative pitches. In other words, a musical scale consists of a finite partition of an octave's interval. Many different scales have been suggested and used, however, the predominant tuning system of Western music is the *twelve-tone equal-tempered scale* , where an octave is partitioned into twelve scale steps. The twelve points in an octave are all equally spaced on a logarithmic scale, with a ratio equal to the 12th root of 2 ($2^{1/12} \approx 1.05946$). The smallest possible interval in this scale is called a *semitone*, and it is the difference between the fundamental frequencies of two subsequent scale steps.

In the twelve-tone equal-tempered scale, there are twelve pitch classes. In Western music notation, these pitch classes are denoted by combining a letter name and accidentals. Seven of the pitch classes are denoted by the letters C, D, E, F, G, A, and B. These pitch classes correspond to the white keys of a piano keyboard as shown in Figure 3.2.

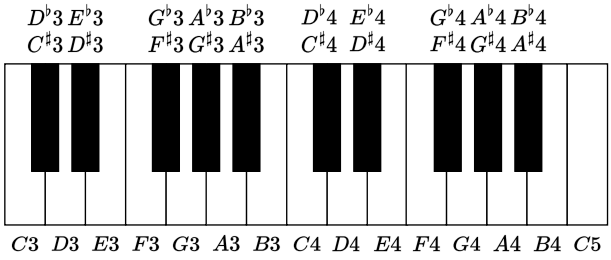

Figure 3.2. *Section of a piano keyboard with keys ranging from C3 to C5.*

The remaining five pitch classes correspond to the black keys of a piano keyboard and are denoted by a combination of a letter and an *accidental* (♯, ♭). A sharp (♯) raises a note by a semitone, i.e. multiplying the given frequency by $2^{1/12}$, and a flat (♭) lowers it by a semitone, i.e. dividing the frequency by $2^{1/12}$. The accidentals are written after the note name. For example, $D^\sharp$ represents D-sharp and $D^\flat$ represents D-flat. In the equal-tempered scale, the remaining five pitches can be either denoted by $C^\sharp$, $D^\sharp$, $F^\sharp$, $G^\sharp$, $A^\sharp$ or by $D^\flat$, $E^\flat$, $G^\flat$, $A^\flat$, $B^\flat$. This means, that $C^\sharp$ and $D^\flat$ represent the same pitch class. The number next to the letters corresponds to the octave that each note belongs to. For example, the A4 means that the musical note A belongs to the 4th octave. The A4 is determined to have a fundamental frequency of 440 Hz. Since subsequent octaves

correspond to a multiplication or division by a factor of two we see that A1 has a fundamental frequency of $440 \cdot \frac{1}{2^3} = 55$ Hz, and A0 a fundamental frequency of 27.5 Hz. The lowest note C0 in this notation has a fundamental frequency in the region of 16 Hz, which is already below what a human can acoustically perceive. Ordering all notes of the equal-tempered scale according to their pitches, one obtains an equal-tempered *chromatic scale* , where all notes of the scale are equally spaced. In the music context, the term "chroma" is equivalent to the notion of pitch class. This means, for example, that both $D3$ and $D4$ have the same chroma since they belong to the same pitch class. As opposed to the piano keyboard, where more or less each piano key can be associated with a periodic wave of fixed frequency, equivalently a pure tone, in real-world situations, sounds are far from being a simple pure tone with a well-defined frequency. Playing a single note on an instrument may result in a complex sound that contains a mixture of different frequencies changing over time. To analyze such sounds, one uses the Fourier theory presented in Chapter 2. In this way, one expresses the complex sound as a superposition of pure tones or sinusoids, each with its own frequency of vibration, amplitude, and phase.

3.2.2 Intensity and Loudness

By now it should be clear that pitch is a subjective notion corresponding to an objective measure, i.e. the notion of frequency. The subjectiveness comes from the fact that each person may perceive a musical sound differently. However, the pure tones used to produce the sound are well-defined since they have a fixed fundamental frequency, amplitude, and phase. Similarly, *loudness* is another subjective measure that correlates to the objective measures of *sound intensity* and *sound power* . Loudness tries to order the musical sounds on a scale extending from quiet to loud. Formally, sound power is the rate at which sound energy is emitted, reflected, transmitted, or received, per unit of time and it is measured in watts (W). Then, *sound intensity* represents the sound power per unit area, and it is measured in watts per square meter (W/m^2). Human beings are capable of realizing sounds with extremely low values of power and intensity. For example, the *threshold of hearing* (TOH), which is the minimum sound intensity of a pure tone a human can hear, is roughly equal to

$$I_{TOH} = 10^{-12} W/m^2. \tag{3.2.1}$$

On the other hand, the upper bound on the intensities a human can perceive is equal to $I_{TOP} = 10 W/m^2$ referred to as the *threhsold of pain* (TOP). It is customary to switch to a logarithmic scale to express power and intensity. More precisely, one uses a *decibel* (dB) scale , which is a logarithmic unit expressing the ratio between two values. Then, the intensity measured in dB is defined as

$$dB(I) = 10 \cdot \log_{10}\left(\frac{I}{I_{TOH}}\right). \tag{3.2.2}$$

From (3.2.2) we see that $db(I_{TOH}) = 0$, and doubling the intensity results in an increase of roughly 3 dB since

$$dB(2 \cdot I) = 10 \cdot \log_{10}(2) + dB(I) \approx 3 + dB(I).$$

Table 3.1 shows the intensity values of some typical sounds.

Table 3.1. *Typical intensity values given in W/m^2. Table borrowed from [2]*

Source	Intensity	Intensity level	$\times TOH$
Threshold of hearing (TOH)	10^{-12}	0 dB	1
Whisper	10^{-10}	20 dB	10^2
Pianissimo	10^{-8}	40 dB	10^4
Normal conversation	10^{-6}	60 dB	10^{10}
Fortissimo	10^{-2}	100 dB	10^{10}
Threshold of pain	10	130 dB	10^{13}
Jet take-off	10^2	140 dB	10^{14}
Instant perforation of eardrum	10^4	160 dB	10^{16}

3.3 Audio features

3.3.1 Spectrograms

In Definition 2.10 we introduced the concept of a spectrogram, which constitutes, perhaps, the most important audio feature used in audio signal processing. The reason for this is that it serves as the building block for extracting the log-power mel-spectrograms and chromagrams. These audio features are used extensively nowadays in many music information retrieval tasks. In our case, the log-power mel-spectrograms will be the primary audio features that we will use for the song identification task.

At this point, let us remind you how one can extract the spectrogram from a digital signal $x : [0, M] \to \mathbb{R}$, resulting from an analog signal $f : \mathbb{R} \to \mathbb{R}$ with equidistant sampling with a sample rate equal to $F_s \in \mathbb{N}$. With this reminder we also make a comment on the number of frequency bins that are selected in practice when someone performs the STFT on any audio signal processing library, such as in librosa [12][1] for instance. First, one assumes that the analog signal $f : \mathbb{R} \to \mathbb{R}$ is of finite duration. Mathematically, this means that there exists an interval $I_a = [0, a]$, $a > 0$ such that $f(t) = 0$ for every $t \notin I_a$. The equidistant sampling with a sample rate equal to F_s yields a DT signal $x : [0, M] \to \mathbb{R}$

$$x(n) = f(n \cdot T), \quad T = 1/F_s. \tag{3.3.1}$$

Then, one determines the length N of the window size and the hop length H to use before applying the STFT. The Fourier Transform of the m^{th} frame is then given by

$$\mathcal{X}(m, k) = \sum_{n=0}^{N-1} x(n + mH)w(n)e^{-2\pi ikn/N}. \tag{3.3.2}$$

The crucial part at this point is to determine the bounds for the time frames m and the number of frequency bins k. To determine the number of time frames m one needs to find how many sections of N points with a shift of H points can fit into the section with M points. To be more

[1] https://librosa.org/doc/latest/index.html

concrete, we illustrate this with an example. Assume that an audio signal $f : \mathbb{R} \to \mathbb{R}$ has a duration corresponding to 5 sec. Furthermore, assume that the equidistant sampling has a sample rate equal to 8 kHz. Then, the digital signal will have 40000 points in total. Some typical values for the window size N and the hop length H are $N = 1024$ and $H = N/2 = 512$. In this case, one needs to determine the lower and upper bound for m such that

$$0 \leq n + mH \leq 40000, \quad 0 \leq n \leq 1023. \tag{3.3.3}$$

The lower bound is obviously $m = 0$. For the upper bound, solving for m in (3.3.3) one obtains

$$\begin{aligned} n + mH \leq 40000 &\iff m \leq \frac{40000 - n}{H} \\ &\iff m \leq \frac{40000 - n}{N/2} \end{aligned}$$

where $0 \leq n \leq 1023 = N - 1$. Since the minimum on the left-hand side is achieved when $n = N - 1$ we get that

$$\begin{aligned} m &\leq \frac{80000}{N} - 2 + \frac{2}{N} \\ &= \frac{80000}{1024} - 2 + \frac{1}{512} \approx 76.12. \end{aligned}$$

Therefore, we conclude that the upper bound is equal to 76, and hence we have 77 time frames in total.[2] To determine the ideal number of frequency bins one refers to the Sampling Theorem in 2.4. First, observe that for fixed m the point mH corresponds to the physical time of mH/F_s seconds of the original audio signal. By (2.3.10) the k^{th} Fourier coefficient in (3.3.2) corresponds to the frequency kF_s/N. As shown at the end of paragraph 2.3, for every k we have that

$$\mathcal{X}(m, k) = \overline{\mathcal{X}(m, n - k)},$$

for every $k = 0, \ldots N-1$. Since we are only interested in the magnitudes of the Fourier coefficients we only need to consider the frequency bins corresponding to $k = 0, \ldots N/2$. Observe that the $N/2^{\text{th}}$ frequency bin corresponds to the physical frequency $F_s/2$, which is the Nyquist frequency. Therefore, we have $N/2 + 1$ frequency bins in total. In our example, the application of the STFT yields a complex matrix $\mathbb{C}^{513\times 77}$. Then, the spectrogram $\mathcal{Y}$ of the audio signal is defined as

$$\mathcal{Y} = \left\{ |\mathcal{X}(m, k)|^2 : 0 \leq m \leq \left[\frac{M - N}{H} \right], 0 \leq k \leq N/2 \right\}, \tag{3.3.4}$$

where M is the number of points that make up the sampled version of the original signal, N is the window size, and H is the hop length of the STFT. In Figures 2.26 & 2.27 shows the (log-amplitude) spectrograms of the A4 piano key and the C major scale. Below you can see the spectrogram obtained from the notes C2, C4, G6, and A6, played in that order. The fundamental frequencies of these notes are 65, 262, 1568, and 1760 Hz, respectively.

Observe that indeed we see a high energy content in the frequency of 65 Hz for the 1st second, in

[2] Results may differ from library to library depending on how they treat the remaining part of the signal. For example, librosa calculates 79 time frames in total.

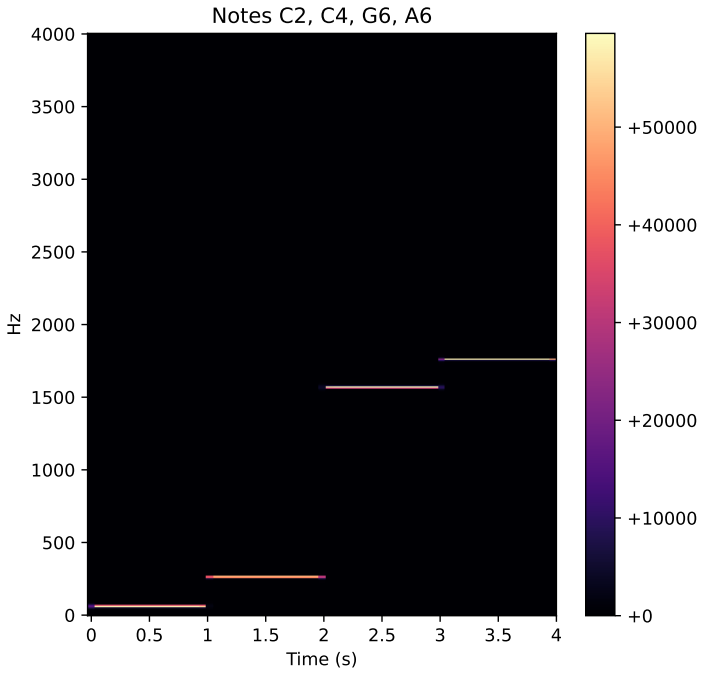

Figure 3.3. *Spectrogram of the notes C2, C4, G6, and A6.*

the frequency of 262 Hz for the 2nd second, and in the frequencies of 1568 Hz and 1760 Hz for the 3rd and 4th seconds, respectively, indicating that indeed the signal corresponds to the notes C2, C4, G6, and A6.

3.3.2 Log Amplitude Spectrograms

Now, a common practice is to convert the square magnitudes $|\mathcal{X}(m,k)|^2$ of the Fourier coefficients obtained from the STFT in a scale which is more perceptual-relevant to the human's ears. In 3.2.2 we introduced the decibel (dB) scale, and in Table 3.1 we presented some typical sounds and their respective intensity decibel levels *with respect to the Threshold of Hearing (TOH).* In the case of spectrograms, it is customary to use as a reference the maximum or minimum value among all values of the square amplitudes contained in the given spectrogram. Mathematically, this can be expressed as follows: Suppose that we have a spectrogram $\mathcal{Y} \in \mathbb{R}^{K\times M}$[3], where K is the number of frequency bins and M is the number of time frames. Then, the *log-amplitude spectrogram* $\mathrm{dB}(\mathcal{Y})$ *with respect to the maximum value* $\mathcal{Y}^*$ is given by

$$\mathrm{dB}\left(\mathcal{Y}\right)(k,m) = 10 \cdot \log_{10}\left(\frac{|\mathcal{Y}(k,m)|}{Y^*}\right), \tag{3.3.5}$$

for every $k = 0,\ldots,K-1$, $m = 0,\ldots M-1$. Observe that the maximum values of $\mathrm{dB}\left(\mathcal{Y}\right)$ in (3.3.5) is 0. In Figure 3.4 you can see the log-amplitude spectrogram obtained from the signal corresponding to the notes C2, C4, G6, and A6, played sequentially.

[3]Here we consider the transpose Y^T of Y defined in (3.3.4). Throughout we will represent the spectrogram by $\mathcal{Y}$, and $\mathcal{Y}^T$ interchangeably, without distinguishing these two representations.

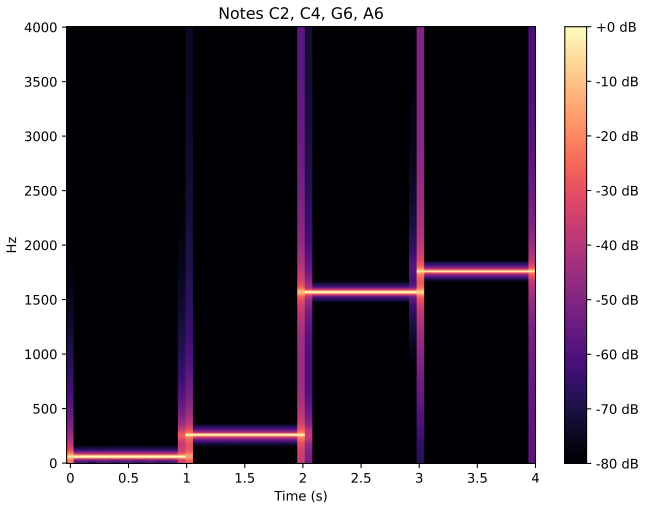

Figure 3.4. *Log-amplitude spectrogram of the notes C2, C4, G6, and A6.*

3.3.3 Mel Spectrograms

Mel-spectrograms is another audio feature that is extracted from spectrograms. As in the case of log-amplitude spectrograms where the idea was to convert the magnitudes of the Fourier coefficients to a more perceptual-relevant scale, i.e. the decibel scale, in the case of mel-spectrograms, the idea is to represent the y-axis corresponding to the spectral dimension in a more perceptual-relevant scale to the human ears. The need to convert the spectral dimension arises from psychoacoustic experiments indicating that the human perception of pitch is not linear but logarithmic. A quick way to see this is by hearing the pitches of the notes C2, C4, G6, and A6 in Figure 3.4. These notes have fundamental frequencies equal to 65, 262, 1568, and 1760, respectively. Observe that the distances between the pitches with frequencies 262 - 65, and 1760 - 1568 are almost the same (around 200 Hz). However, if we were to hear these pitches played sequentially we would observe that the pitch distance (difference) between the pair of notes corresponding to G6 and A6 sounds as if were the same as opposed to the pair of notes corresponding to C2 and C4. This indicates that the human perception of pitch varies in a non-linear fashion. From psychoacoustic experiments, it has been observed that the relation that best describes the human perception of pitch is given by the following formula

$$m = 2595 \cdot \log_{10}\left(1 + \frac{f}{700}\right). \tag{3.3.6}$$

One speaks about mel frequencies and the mel scale when referring to the conversion from the standard Hz scale to the scale described by (3.3.6). For example, the Mel frequencies of C2, C4, G6, and A6 are 100, 358, 1324, and 1416 Mel Hz, respectively. Observe now that the difference between C4 and C2 is 258 Mel Hz, and the difference between A6 and G6 is 90 Mel Hz. Figure 3.5 shows the graph of the function defined by (3.3.6).

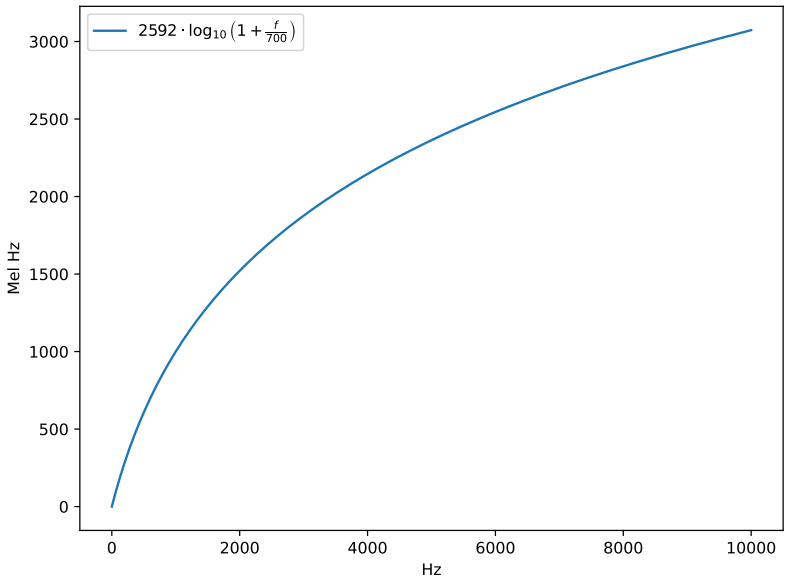

Figure 3.5. *Relation between Hz and Mel Hz.*

The transition from spectrograms to mel-spectrograms requires a few more steps than just converting the frequencies from the standard Hz scale to the mel scale in one-to-one correspondence and then writing down the amplitudes of these frequencies. In practice, one determines the number of frequencies bins to keep after the transformation of a log-amplitude spectrogram to a mel-spectrogram in the mel scale. Then, the amplitudes of these selected frequencies are calculated by applying triangular filters in the amplitudes of the frequencies in the original log-amplitude spectrogram. This process can be divided into the following three steps:

(i) Choose the number of mel bands (the frequency bins).

(ii) Construct the mel filter banks (The matrix obtained from the triangular filters).

(iii) Apply mel filter banks to the log-amplitude spectrogram to obtain the mel-spectrogram (matrix multiplication).

We explain these three steps both by presenting the general methodology and by illustrating it with the conversion of the log-amplitude spectrogram, corresponding to the notes C2, C4, G6, and A6 in Figure 3.4, to their respective mel-spectrogram representation. In the general case, suppose we have a log-amplitude spectrogram $\mathcal{Y} \in \mathbb{R}^{F \times T}$, where F denotes the number of frequency bins in the original Hz scale, and T to the number of time frames obtained after the application of the STFT to the original raw audio signal. Having chosen a number of frequency bins M, the construction of the *mel filter banks* can be summarized in the following steps:

(i) Convert lowest (L') and highest (H') standard frequency to Mel by (3.3.6) to obtain two ending points $L < H$.

(ii) Create M number of equally spaced points in the interval $[L, H]$.

(iii) Convert the equally spaced points from the mel scale back to the standard Hz scale.

(iv) Create M triangular filters to obtain a matrix $\mathcal{M} \in \mathbb{R}^{M \times F}$ (mel filter banks).

Then, the mel-spectrogram $\mathcal{S} \in \mathbb{R}^{M \times T}$ is calculated via the multiplication $\mathcal{S} = \mathcal{M} \cdot \mathcal{Y}$. In detail, the process of constructing the mel filter banks $\mathcal{M}$ is the following: The M equally-spaced number of points $\{m_1, \ldots, m_M\}$ in the interval $[L, H]$ are given by

$$m_j = L + j \cdot \frac{H - L}{M + 1}, \quad j = 1, \ldots, M. \tag{3.3.7}$$

Then, we convert these m_j equally-spaced points back to standard Hz scale using the inverse of (3.3.6) given by

$$f_j = 700 \cdot \left(10^{m_j/2595} - 1\right), \quad j = 1, \ldots, M - 1. \tag{3.3.8}$$

Create M triangular filters T_j with base determined by the points f_{j-1}, f_{j+1} and the third point being the m_j, where $f_0 = L'$, $f_{M+1} = H'$. The triangles are scaled to have a height of 1. These triangular filters are given by

$$T_j(x) = \left(1 - 2A_j|x - a_j|\right) \cdot \mathbb{1}_{|x| \leq \frac{1}{2A_j}}(x), \quad j = 1, \ldots, M, \tag{3.3.9}$$

where $A_j = f_{j+1} - f_{j-1}$, $a_j = f_j$, and

$$\mathbb{1}_{|x| \leq \frac{1}{2A_j}}(x) = \begin{cases} 1, & |x| \leq \frac{1}{2A_j} \\ 0, & |x| > \frac{1}{2A_j} \end{cases}.$$

To obtain the mel filter banks $\mathcal{M} \in \mathbb{R}^{M \times F}$ one applies the triangular filters T_j to the frequencies $s_1, \ldots, s_F$ corresponding to the F frequency bins of the spectrogram $\mathcal{Y} \in \mathbb{R}^{F \times T}$. For example, in the case where the window of the STFT is equal to $N = 1024$ and the sampling rate of the signal is $F_s = 8000$, then we have $F = N/2 + 1 = 513$ frequency bins in total, where the lowest frequency is 0 and the highest is $F_s/2 = 4000$ Hz. Formally, the matrix $\mathcal{M}$ can be calculated by

$$\mathcal{M} = \begin{pmatrix} T_1(s_1) & \ldots & T_1(s_F) \\ \vdots & \ddots & \vdots \\ T_M(s_1) & \ldots & T_M(s_F) \end{pmatrix}. \tag{3.3.10}$$

The mel-spectrogram representation of $\mathcal{Y}$ is $\mathcal{S} = M \cdot \mathcal{Y}$. To demonstrate this transition consider the log-amplitude spectrogram in 3.4. In this case, the sampling rate is equal to $F_s = 8000$, the window length is equal to $N = 1024$, and the hop length is equal to $H = 512$. The total duration of the signal is 4 seconds. Therefore, the size of the log-amplitude spectrogram $\mathcal{Y}$ in 3.4 has $F = 513$ and $T = 63$. Choosing $M = 20$ Mel frequency bins we obtain the following triangular filters shown in Figure 3.6. In Figures 3.7, 3.8 you can see the corresponding mel-spectrograms of the notes C2, C4, G6, and A6, with 20, and 256 frequency bins, respectively. The number of frequency bins is a tuning parameter and depends on the task at hand. Common choices are 40, 60, 90, 128, and 256. The Mel-spectrograms, are perhaps the most widely used audio features nowadays. Their importance lies in the fact that they can reduce the dimensions of the initial spectrogram while at the same time keeping all the interesting information. They have been extensively used in many

song identification tasks [13, 14, 15] and will also serve as the main audio feature in our song identification task.

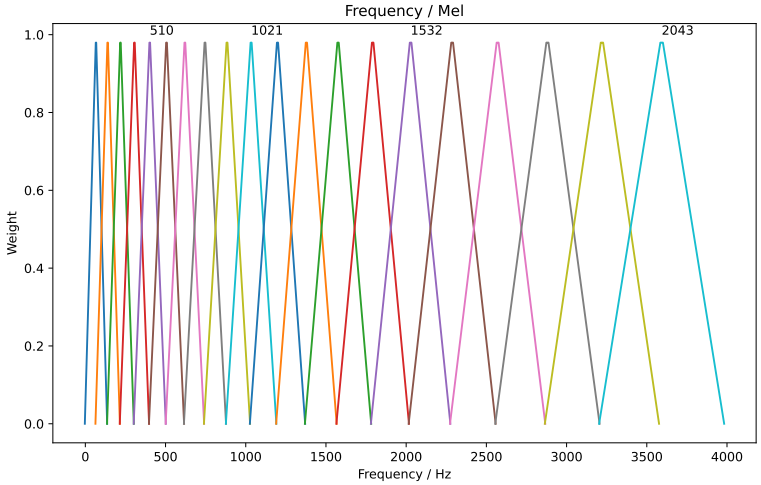

Figure 3.6. *Visualization of the triangular filters.*

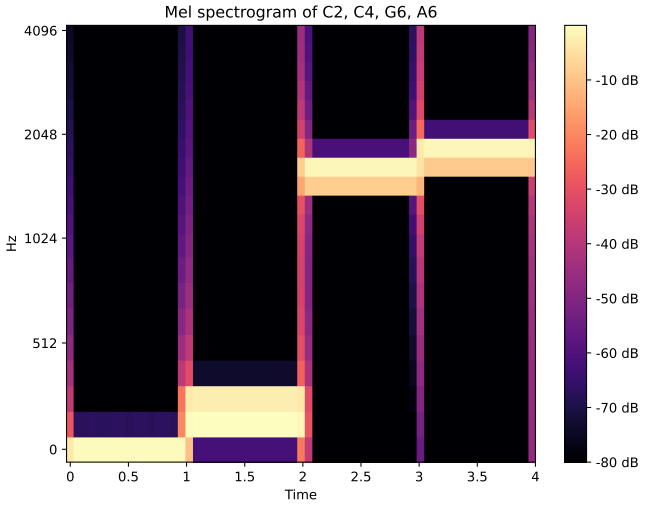

Figure 3.7. *Mel spectrogram of the notes C2, C4, G6, and A6 with 20 frequency bins.*

3.3.4 Continuous Q Transform (CQT)

As we mentioned in 3.2.1, in Western music, every two neighboring points in the twelve-tone equal-tempered scale have a pitch ratio equal to the 12th root of 2, i.e. $2^{1/12} \approx 1.05946$. The application of the STFT on a raw DT signal yields a linear frequency resolution. To see this, assume that the DT signal x has a sampling period equal to $T = 1/F_s$ and the window length of the STFT is N. Then, by (2.3.10) the k^{th} frequency bin obtained by the STFT corresponds to the physical frequency k/NT of the original CT signal. This shows that the obtained frequencies from the STFT are linearly spaced with each other as shown in Figure 3.9. However, when we

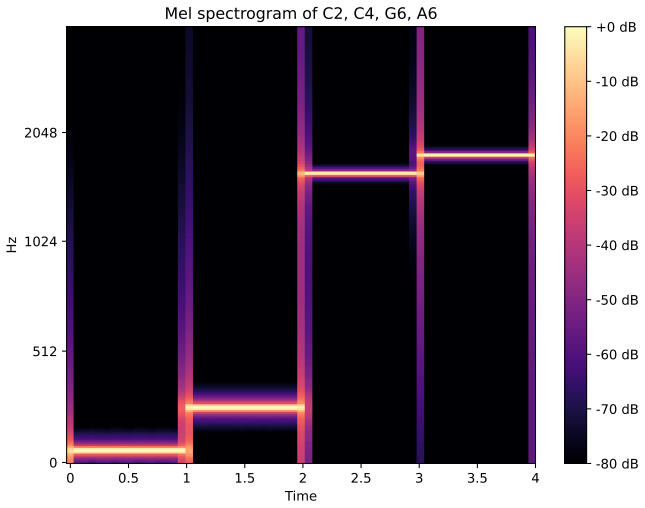

Figure 3.8. *Mel spectrogram of the notes C2, C4, G6, and A6 with 256 frequency bins.*

deal with audio signals corresponding to some piece of music it would be beneficial to have a frequency resolution that each two neighboring points will have a pitch ratio equal to $2^{1/12}$. When we introduced the mel-spectrograms we mapped the frequency scale obtained from the STFT to the mel scale, following a log scale. But, also in this case the frequency bins that we obtain at the end, may not have a one-to-one correspondence to the notes used in Western music.

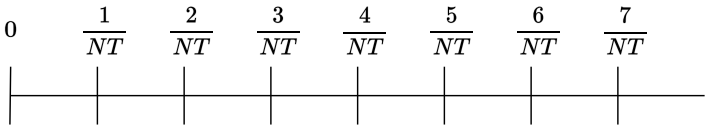

Figure 3.9. *The frequency bins obtained from the STFT.*

For that reason, a variant of the STFT known as Continuous Q Transform (CQT) has been developed [16]. In this section, we describe the basic principles underlying this transformation and we visualize the resulting spectrograms. The resulting features from the CQT are called *CQT-features* and have been used as input features to deep learning models in Cover Song Identification tasks (CSI) [17, 18]. A comparison of the performance of the various audio features in an environmental classification task using Convolutional Neural Networks can be found in [19].

In the Continuous Q Transform one fixes a minimum frequency f_{min} and requires the rest of the frequencies f_k to satisfy

$$\frac{f_{k+1}}{f_k} = 2^{1/b}. \tag{3.3.11}$$

In practice f_{min} is equal to the pitch of C1 which is 32.70 Hz, and $b = 12$, corresponding to the semi-tone resolution. With this setting, we require

$$Q = f_k/\delta f_k = \frac{f_k}{0.059 f_k} = 17, \tag{3.3.12}$$

where

$$\delta f_k = f_{k+1} - f_k = 2^{1/12} f_k - f_k = f_k(1 - 2^{1/12}), \tag{3.3.13}$$

is the resolution bandwidth and Q is known as the quality factor Q. Now, in order for the ratio of frequency to bandwidth to be a constant (constant Q) the window size must vary inversely with frequency. To this end, assuming a DT signal x with sampling rate F_s, the length of the window in samples at frequency f_k should be equal to

$$N(k) = F_s/\delta f_k = \frac{F_s}{f_k} Q. \tag{3.3.14}$$

To see why this should happen, observe that the resolution bandwidth in the STFT is

$$\delta f_k = f_{k+1} - f_k = (k+1)\frac{F_s}{N} - k\frac{F_s}{N} = F_s/N, \tag{3.3.15}$$

where N is the window length. For the neighboring frequencies to satisfy (3.3.11), one should also have that $\delta f_k = f_k/Q$. Therefore, we obtain

$$N = F_s/\delta f_k = \frac{F_s}{f_k} Q. \tag{3.3.16}$$

The expression for the k^{th} spectral component in the case of the DFT is

$$X(k) = \sum_{n=0}^{N-1} x(n) w(n) e^{-2\pi j k n/N}, \tag{3.3.17}$$

where $w(n)$ is the Hann window. In this expression, the digital frequency is $2\pi k/N$. In the case of the constant Q transform, in order to take account of the constraints of Eqs. (3.3.11),(3.3.12), and (3.3.14), the digital frequency of the k^{th} component should be $2\pi Q/N(k)$. This is because the ratio for two neighboring points will satisfy

$$\frac{2\pi Q/N(k+1)}{2\pi Q/N(k)} = \frac{N(k)}{N(k+1)} = \frac{f_{k+1}}{f_k} = 2^{1/12},$$

as desired. Furthermore, the window function should have the same shape for each component, but since its length is determined by $N(k)$, it should be a function of k as well as n. Therefore, taking into account the fact that the number of terms varies with k, by normalizing with $N(k)$, the constant Q transform is given by

$$X(k) = \frac{1}{N(k)} \sum_{n=0}^{N(k)-1} W(k,n) x(n) e^{-2\pi j Q n/N(k)}, \tag{3.3.18}$$

where the maximum k is chosen such that the digital frequency f_k not exceeds the Nyquist frequency $F_s/2$. Formally, by (3.3.17) we get

$$\begin{aligned} f_k < \frac{F_s}{2} &\iff \frac{F_s}{N(k)} Q < \frac{F_s}{2} \\ &\iff N(k) > 2Q. \end{aligned}$$

Therefore, the maximum k is determined by

$$k^* = \max\{k \in \mathbb{N} : N(k) > 2Q\}. \tag{3.3.19}$$

Finally, the Hann window in (3.3.18) is given by

$$W(k,n) = a + (1-a)\cos(2\pi n/N(k)),$$

where $a = 25/46$ and $0 \le n \le N(k) - 1$. The CQT representation of the DT signal x is then given by

$$\mathcal{Y}_{cqt} = \left\{|X(k,m)|^2 : 0 \le k \le k^*,\ m \in \mathbb{Z}\right\}, \tag{3.3.20}$$

where $X(k,m)$ is derived by application of the constant Q transform in (3.3.18) in short-time segments of the original signal. For example, in the case of the audio signal of the notes C2, C4, G6, and A6 with a sampling rate equal to $F_s = 8000$, a hop length $H = 512$, and a frequency resolution corresponding to the semi-tone resolution we get a 83×63 2D representation. The maximum number (83) of frequency bins is determined by

$$k^* = \max\left\{k \in \mathbb{N} : f_{\min}^{k/12} = (32.7)^{k/12} < F_s/2 = 4000\right\},$$

assuming that the lowest pitch note is C1. Figure 3.10 shows the corresponding representation obtained from the continuous Q transform. In this representation, it is far from evident when and which notes were played.

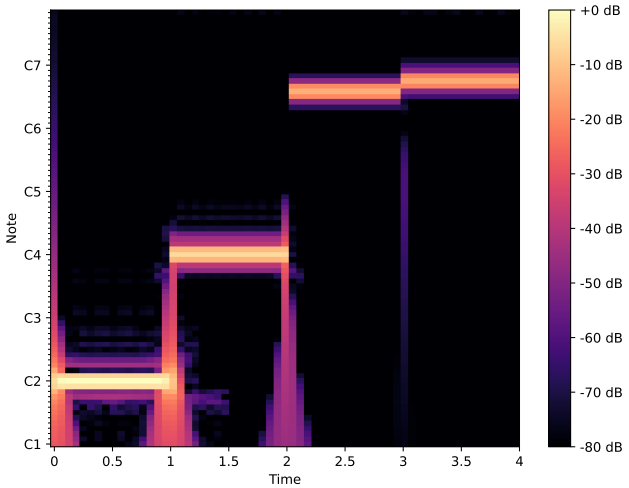

Figure 3.10. *CQT representation of the audio signal corresponding to C2, C4, G6, and A6.*

3.3.5 Chroma Features

Chroma features or chromagrams are yet another important audio feature that is extensively used in several MIR (Music Information Retrieval) tasks. In short, they are constructed by first grouping all pitches that belong to the same pitch class (i.e. their ratio is a power of 2) and then summing

their amplitudes. In the case of the C2, C4, G6, and A6 pattern in 3.10 where a semitone resolution is used, the spectral dimension of the chroma representation is 12. In Figure 3.11 we can see the chroma representation of the audio signal corresponding to the four aforementioned notes.

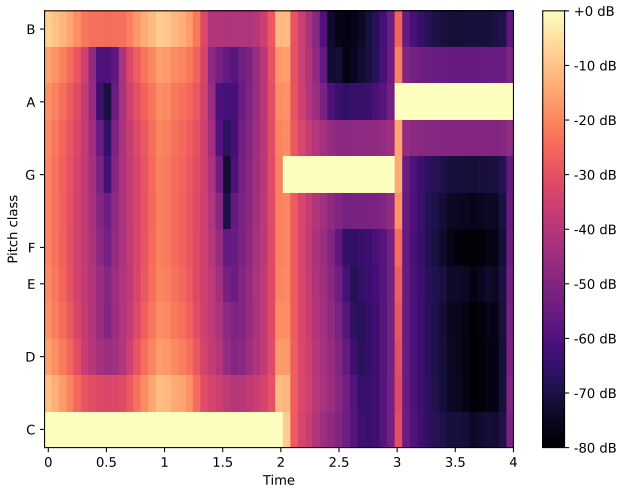

Figure 3.11. *Chroma representation of the audio signal corresponding to C2, C4, G6, and A6.*

3.4 Audio Augmentations

As we shall shortly see, one of the main challenges in designing music information retrieval applications that are used by the users in real-time, is that the audio fragments are captured in noisy environments, and as such, the audio signals arriving on the server are distorted. To this end, one needs to understand these types of distortions that may arise before designing such systems. In the upcoming sections, we will apply on purpose such distortions to the original signals in order to train a deep-learning model to suppress these distortions and output a feature vector that would be as close as possible (with respect to a distance measure) to the feature representation corresponding to the original signal.

3.4.1 Background Noise

One of the most common distortions encountered when recording audio fragments from mobile devices is the presence of background noise, such as people talking, car engines, and other ambient sounds. In the upcoming section, where our focus is on the song identification task, we will explore how this particular type of distortion has driven established applications like Shazam to develop sophisticated algorithms and robust feature representations, aiming to enhance their performance. In this paragraph, we define the term *Signal to Noise Ratio* which determines the "amount" of noise to be added to a clean audio signal, and explain mathematically how one can output an audio signal that is the result of a clean audio with added background noise. We first start with some basic definitions from signal processing:

Definition 3.11 (Energy and Power of a DT signal)**.** *Given a DT signal* $x : \mathbb{Z} \to \mathbb{R}$*, the* energy *of the signal is given by*

$$E_x = \sum_{n=-\infty}^{\infty} |x(n)|^2, \tag{3.4.1}$$

, and its power *by*

$$P_x = \lim_{N\to\infty} \frac{1}{2N+1} \sum_{n=-N}^{N} |x(n)|^2. \tag{3.4.2}$$

In signal processing, Signal-to-noise ratio (SNR) is a measure that compares the level of a desired signal to the level of background noise. SNR is defined as the ratio of signal power to noise power. A ratio higher than 1 indicates more signal than noise while a ratio lower than 1 indicates more signal than noise. More formally, suppose that y is the original signal, and b the signal corresponding to some ambient noise. Then, the SNR is given by

$$SNR = \frac{P_y}{P_b} = \frac{E_y}{E_b}. \tag{3.4.3}$$

It is customary to express the above ratio in decibels by using the logarithm:

$$SNR_{dB} = 10 \log_{10}\left(\frac{P_y}{P_b}\right). \tag{3.4.4}$$

This is equivalent to

$$P_{y,dB} = SNR_{dB} + P_{b,dB}, \tag{3.4.5}$$

where $P_{y,dB} = 10\log_{10}(P_y)$, $P_{b,dB} = 10\log_{10}(P_b)$, the power of y, and b, in decibel scale. Below we provide an algorithm that outputs a distorted signal z, corresponding to the addition of some ambient noise b to a clean signal y with fixed SNR.

Algorithm 3.1: *Adding Background Noise*

Input: Clean audio y, Noise signal b, SNR in dB
1: $SNR \longleftarrow 10^{SNR/10}$
2: $E_y \longleftarrow \sum_{n=-\infty}^{\infty} |y(n)|^2$.
3: $E_b \longleftarrow \sum_{n=-\infty}^{\infty} |b(n)|^2$.
4: $z \longleftarrow \sqrt{\frac{E_b}{E_y} SNR} \cdot y + b$.
Output: z

3.4.2 Reverberation - Impulse Response

Reverberation, often referred to as reverb, is a phenomenon that occurs in acoustic environments when sound waves reflect off surfaces and create a persistence of sound after the original sound source has stopped. It is the collection of reflections that continue to bounce around in a room or space before fading away. When sound waves encounter surfaces such as walls, floors, and objects, they bounce off these surfaces and travel back to our ears. This creates a series of reflections that blend with the direct sound from the source. These reflections give a sense of spaciousness, depth, and ambiance to the sound. To add the reverberation effect from a physical environment, such as

a theater, a train station, a street, etc., to a clean audio signal to make it sound as if were played in this environment, one needs an impulse response produced in the environment and the discrete convolution

$$(y * x)(n) = \sum_{m=-\infty}^{\infty} y(m)x(n-m), \quad n \in \mathbb{Z}$$

of the DT signals y, x.

3.4.3 High and Low Pass Filters with Decaying Energy

The energy decay of low and high frequencies is a common distortion when the audio stream is captured through a microphone in noisy environments. Therefore, any efficient music recognition system must exhibit robust behavior against such distortions. To develop a system capable of disregarding these distortions, we first need to understand how to introduce these distortions to a clean audio signal. In paragraph 2.2.5, we showed how one can completely remove frequencies below/above certain thresholds using the properties of the Fourier Transform. However, these transformations are extreme cases that rarely arise in realistic scenarios. In practice, it is more common for some frequencies below/above a certain threshold to experience a decay in terms of their energy. We refer to these transformations as *low/high pass filters with decaying energy*. For example, in the case of the low pass filter, octaves above a certain threshold usually experience a decay with constant decreasing factors. Similarly, in the high pass filter, octaves below a certain threshold experience a similar decay. In this paragraph, we address the case where the energy of frequencies above a certain threshold is reduced by a fixed number of decibels. It is not hard to extend this idea to the case where the frequencies of each octave are scaled down by different factors. Furthermore, the case of the high pass filter is handled in a similar manner.

Suppose now that we have a clean audio signal $f : \mathbb{R} \to \mathbb{R}$ and we want to reduce all the frequencies of f above a threshold v_C by a decibels. The goal is to determine the function $g : \mathbb{R} \to \mathbb{R}$ that when convolved with f yields the desired result. Using the Fourier Transform in 2.2.6 and taking the square magnitudes we obtain the power spectrum of f

$$\left\{|\mathcal{F}f(s)|^2 : \ s \in \mathbb{R}\right\}. \tag{3.4.6}$$

The next step is to find the factor β such that the product $\beta \cdot |\mathcal{F}f(s)|^2$ corresponds to a reduction of a decibels. To this end, we convert the square magnitudes to the decibel scale by

$$20 \cdot \log_{10}\left(\frac{|\mathcal{F}f(s)|}{F^*}\right),$$

where $F^* = \max_s |\mathcal{F}f(s)|^2$. Then, we need to find β such that

$$20 \cdot \log_{10}\left(\frac{\beta \cdot |\mathcal{F}f(s)|}{F^*}\right) = 20 \cdot \log_{10}\left(\frac{|\mathcal{F}f(s)|}{F^*}\right) - a. \tag{3.4.7}$$

Solving 3.4.7 for β, we obtain $\beta = 10^{-a/20}$. Therefore, the Fourier Transform of g must be equal

to

$$\mathcal{F}g(s) = \begin{cases} 1, & |s| < v_C \\ 10^{-a/20}, & |s| \geq v_C. \end{cases}$$

Or equivalently, using the high pass and low pass filters in 2.2.5,

$$\mathcal{F}g(s) = 10^{-a/20} \cdot \left(1 - \Pi\left(\frac{s}{2v_C}\right)\right) + \Pi(s/2v_C). \tag{3.4.8}$$

Hence, by the inverse Fourier Transform we conclude that

$$g(t) = 10^{-a/20} \cdot \delta(t) - 2 \cdot 10^{-a/20} v_C \text{sinc}(2\pi v_C t) + 2v_C \text{sinc}(2v_C t).$$

In practice, these operations are handled by their discrete analog using the theory in section 2.3. This means that, instead of the continuous Fourier Transform the discrete version is used to obtain the discrete power spectrum

$$\{|\hat{x}(s)|^2 : s = 0, \ldots, N/2\},$$

where N is the length of the sampled version of f. Then, the frequencies satisfying $s \cdot F_s/N > v_c$ are multiplied by $10^{-a/10}$. The inverse discrete Fourier Transform is used to yield the transformed audio signal.

3.5 Song Identification

At this point, using the theoretical tools and the audio features that we introduced in the previous sections, we can design sophisticated music information retrieval systems. In this section, we focus on one of the most popular tasks: song identification, which will serve as our main task for applying our theory. In short, in the audio identification task, one is interested in retrieving content information, such as the artist's name, album name, release date, etc., about a short segment of a song that is played and recorded through a device.

Throughout this thesis, we will present two different approaches for the song identification task: the classic method corresponding to the audio fingerprinting technique that many well-known commercial systems utilize, such as Shazam, and the modern approach via deep learning and contrastive learning. In this section, we focus on the classic method, where we introduce the concept of an audio fingerprint and present the basic principles underlying the algorithms that these systems use to index and query large databases consisting of audio fingerprints from millions of songs. The exposition with regard to the audio identification presented in this section is again based on Müller's book *Fundamentals of Music Processing* [2]. Here, we only provide a quick overview of the classic approach as we mainly focus on the deep learning approach in the upcoming sections.

3.5.1 Problem and Challenges

The goal of an audio identification system is simple: assume you hear a song in a restaurant, shopping mall, or car, and you want to learn more about it. Then, you record a short audio fragment with a mobile device, which is then converted into a compact and descriptive fingerprint representation. This fingerprint representation is transmitted through the network to a server and

compared to the audio fingerprints in the server's database using a similarity measure. When this similarity measure exceeds a certain threshold with the best matching part of the database, the user receives all the metadata linked to this part of the database (e.g., song name, artist name, release date, album, genre, etc.). This high-level description of an audio identification system following the client-server approach is depicted in Figure 3.12.

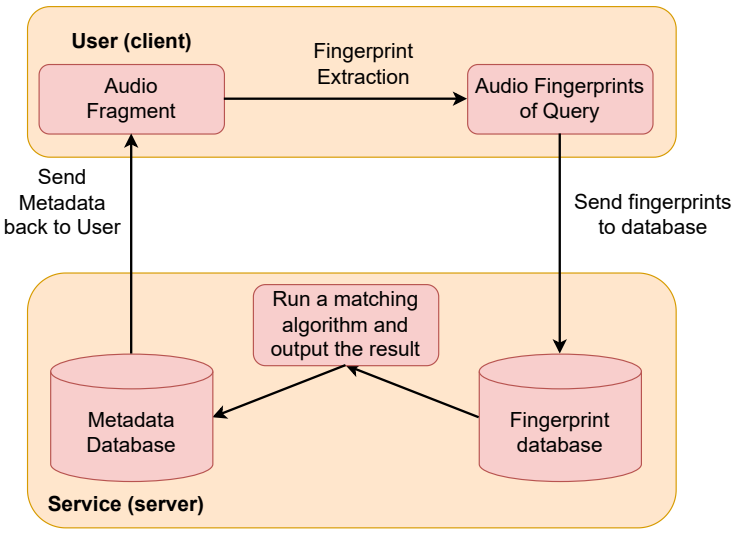

Figure 3.12. *A high-level representation of an audio identification system.*

These music recognition systems need to be robust and computationally efficient, which leads to a number of technical challenges that need to be solved. To begin with, the fingerprint representation of the audio fragments must be as descriptive as possible in order to be correctly identified and matched among millions or even billions of other fingerprints, while also being in a compressed form to reduce the storage footprint. The main challenge in these systems is that the discrimination requirement is undermined by the fact that the recorded audio fragments are exposed to many different distortions. For instance, the user may capture the audio fragment in a noisy and reverberant environment where people talking in the background or other ambient noise is present. Therefore, the audio fingerprint representation must also be robust to such degradations. Additionally, when the query arrives at the database, the searching algorithm should be computationally efficient in order to retrieve the result within a reasonable amount of time for real-time applications. The exhaustive approach in a database with millions or billions of entries is prohibitive, and one needs to design or choose an algorithm that utilizes suboptimal searches to reduce the response time. It is worth noting at this point that the audio identification system described above falls into the broader category of recommendation and similarity search systems. For example, in another scenario, one may be interested in retrieving all relevant documents in a database with a query document. The pattern, in this case, is the same: one extracts a feature representation (in vector form) of the documents and utilizes similarity search techniques to query the database. To illustrate the aforementioned challenges in designing an effective audio identification system we first adopt a naive solution that reveals the difficulties one has to face when designing such systems. The first step is to extract a feature representation of the songs to be stored in the database. An ideal feature that captures both the spectral information and the time presence of this information is the spectrogram

representation of the audio introduced in 3.3.1. Assuming we choose this feature representation for our audio tracks, and use a 32-bit floating point precision, we can easily see that the size of the database is roughly

$$O\left(4K \cdot \left(\frac{M-N}{H}\right) \cdot \frac{N}{2}\right), \tag{3.5.1}$$

bytes, where K is the number of total songs in the database, N is the window length of the STFT, H is the hop size, and $M = T/F$ is the average number of samples for every song, with T being the average duration in seconds, and F the sampling rate. Table 3.2 shows how the size of the database varies with respect to the parameters in (3.5.1). Observe that in the case of a database consisting of 1 million songs the size is more than 15 TB, which is already a huge amount of space.

Table 3.2. *Database size with spectrograms as feature representations.*

K	T (sec)	F	N	H	**Total size**
100000	180	8000	1024	512	575 GB
1000000	180	8000	1024	512	5,7 TB
100000	180	22050	1024	512	1,5 TB
100000	180	22050	2048	1024	1,5 TB
1000000	180	22050	1024	512	15,8 TB
1000000	180	22050	2048	512	31,7 TB

The size requirements in Table 3.2 indicate that the feature representation must be as compact as possible while at the same time retain all the relative information of the audio segment to be correctly matched. Another issue is how to index the database to allow for fast retrieval operations. To see this, in the approach with the spectrogram representation, assuming a query spectrogram S_Q of size $T_Q \times F$, and $T_B \times F$ ($T_B >> T_Q$), a naive solution would be to slide over the query spectrogram S_Q over the spectrograms in the database and perform a time-wise cosine similarity to output the part of the database that best matches the query. However, this approach has a computational complexity that is linearly dependent on the size of the database K, and as such, it would be impractical for real-time scenarios. In the following sections, we address these issues and present the main ideas adopted by Wang [1] that are used in Shazam's audio identification system.

3.5.2 Audio Fingerprints

The prevalent feature representations that were used (and still used), before the rapid expansion of deep learning, are the audio fingerprints. These features can be thought of as a compressed form of spectrograms while at the same time keeping all the spectral information that is more likely to survive if the audio is captured in a noisy environment. The idea is simple: instead of keeping all the points in the spectrogram, one only chooses to keep the points that have higher magnitude than all their neighbors within a region around the respective points. In detail, given an audio signal in the form of sampled waveform, the first step in deriving the fingerprints consists of computing an STFT $\mathcal{X}$ as in (2.4.1). Then, one obtains a spectral-temporal representation $\mathcal{X}(n, k)$ of the signal, where $k \in [0, K]$ is referred to as the frequency stamp, and $n \in \mathbb{Z}$ as the time stamp.

In the second step, one determines the neighborhood to extract the local maximums in the representation $\mathcal{X}$. This neighborhood is determined by two parameters, n_0, and k_0, corresponding to the time domain, and the frequency domain, respectively. For a point (n, k), these parameters define a small rectangular described by

$$\mathcal{R}_{n_0,k_0}(n,k) = [n - n_0, n + n_0] \times [k - k_0, k + k_0] \subseteq \mathbb{Z} \times K. \tag{3.5.2}$$

Then, the point (n, k) is a local maximum on the rectangular $\mathcal{R}_{n_0,k_0}(n, k)$ if and only if

$$|\mathcal{X}(n,k)| > \left|\mathcal{X}(n',k')\right|, \tag{3.5.3}$$

for every $(n', k') \in \mathcal{R}_{n_0,k_0}(n, k)$. Figure 3.13 shows the local maximum points in the spectrogram corresponding to 30 seconds of "The Four Seasons - Spring" by Vivaldi[4], as well as the local maximum points in the resulting spectrogram when background noise with people talking has been added with an SNR of 0 dB.

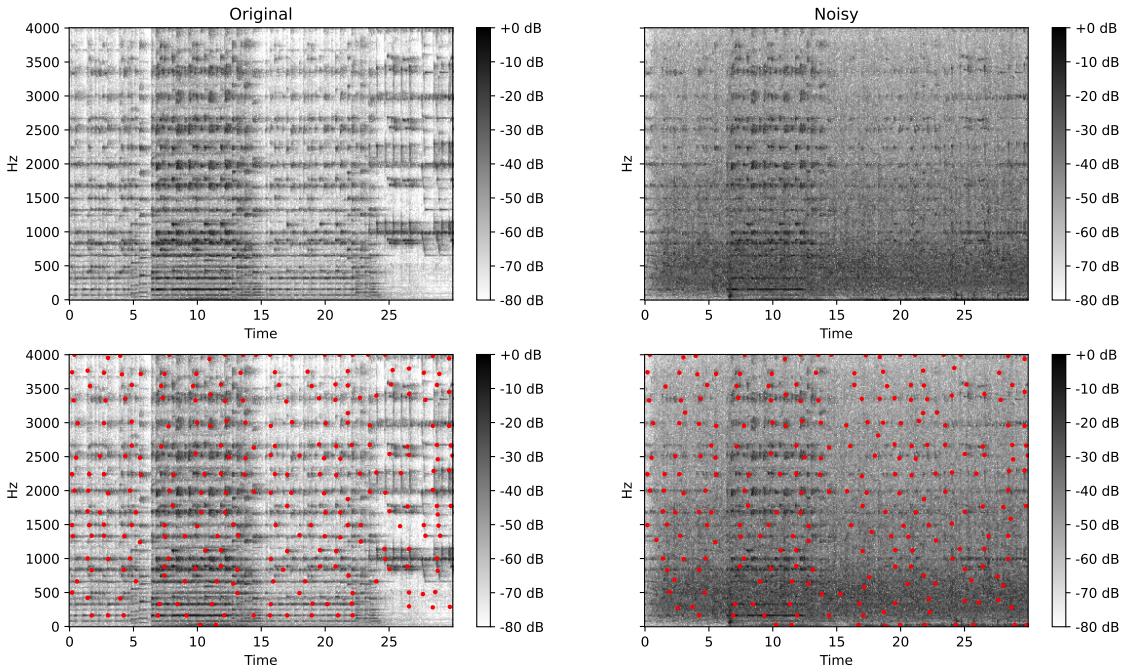

Figure 3.13. *Spectrograms corresponding to 30-sec audio fragment from The Four Seasons - Spring of Vivaldi. On the left is the clean audio, and on the left is the distorted signal with background noise added (SNR = 0 in dB).*

As you can see, the spectrograms in the first row are quite different, but they agree on most of their local maximums, as indicated by the second row. At this point, we can define a similarity measure to evaluate the local maximum preservation between pairs of spectrograms. As before, suppose that the query spectrogram S_Q has size $T_Q \times F$, and the concatenation of the spectrograms in the database yields a large spectrogram of size $T_B \times F$, with $T_B >> T_Q$. Denote by $C(S_Q)$ the points

[4]Hear it in: https://www.youtube.com/watch?v=t2cIUu-sS7w

(n, k) where the local maximums occur on S_Q for fixed n_0, k_0. Formally, $C(S_Q)$ is described as

$$C(S_Q) = \{(n, k) \in T_Q \times F : |S_Q(n, k)| > |S_Q(n', k')|, (n', k') \in R_{n_0,k_0}(n, k)\}. \tag{3.5.4}$$

Similarly, $C(S_B)$ represents the points where the local maximums occur in the concatenated spectrogram of the database. We define the shifted local maximums $m + C(S_Q)$ of $C(S_Q)$ to be the set

$$m + C(S_Q) = \{(n + m, k) \in T_Q \times F : (n, k) \in S_Q\}, \tag{3.5.5}$$

for $m \in \mathbb{N}$. Now, for every m we define the confidence measure $\Delta : \mathbb{N} \to \mathbb{R}$ by

$$\Delta(m) = \frac{\left|(m + C(S_Q)) \cap C(S_B)\right|}{\left|C(S_Q)\right|}. \tag{3.5.6}$$

Then, the part of the database with the highest confidence measure Δ can be determined by the optimal integer m^*, where

$$m^* = \arg\max_{m \in \mathbb{N}} \Delta(m).$$

In Figure 3.14 you can see how the confidence $\Delta(m^*)$ varies with respect to the amount (SNR) of noise added to the query of the 30-sec audio fragment of Vivaldi's Four Seasons - Spring. You can see that even in very low SNRs (0 dB) we have that $\Delta(m^*) > 0.5$.

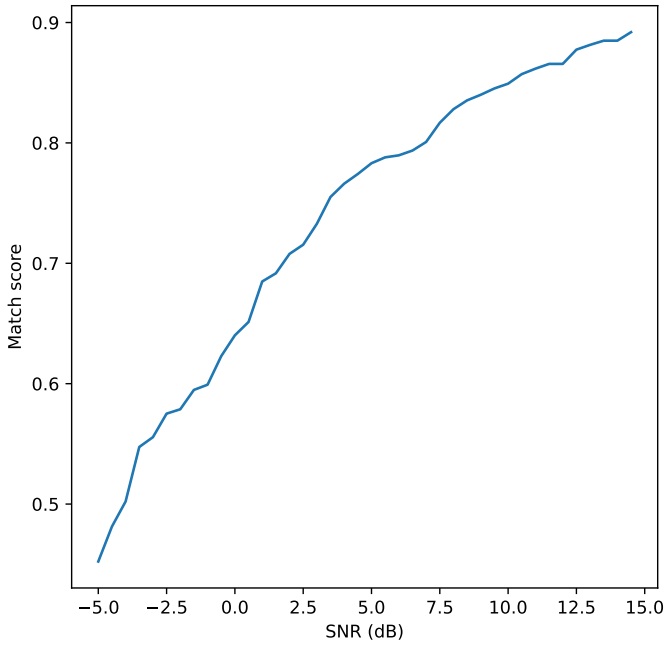

Figure 3.14. *SNR vs Match Score*

3.5.3 Indexing with Inverted Lists

As we have already mentioned, the computational complexity of the previous approach where we slide the feature representation $C(Q)$ of the query Q over the entire database to find the optimal m^*, is linearly dependent on the size of the database. This is prohibitive for real-time applications where the client expects the answer in a few seconds. To this end, efficient search strategies typically use indexing techniques, which optimize the speed performance by cutting down the search space through suitable look-up operations. In this section, we introduce a general indexing and retrieval framework based on inverted lists. It should be noted that this framework is not restricted only to the song identification case, and can be applied to any system that stores data items (e.g. documents, audio, text, etc) in the form of a specific feature representation on a database, and requires efficient and fast retrievals. To formalize this framework, we assume that the data items to be indexed consist of a time stamp $n \in \mathbb{Z}$, and a hash $h \in \mathcal{H}$, where $\mathcal{H}$ denotes the set of all possible hash values. In information retrieval, the notion of a hash is used to refer to a fixed length identifier (e.g. a binary string consisting of a fixed number of bits). Furthermore, we assume that the data items $\mathcal{D}$ are stored on the database in the form $\mathcal{F}(\mathcal{D}) \subseteq \mathbb{Z} \times \mathcal{H}$, referred to as the feature representation of $\mathcal{D}$. Now, for every $h \in \mathcal{H}$, we consider the inverted list $L(h)$, consisting of all time stamps $n \in \mathbb{Z}$, in increasing order, for which $(n, h) \in \mathcal{F}(\mathcal{D})$. I.e. $L(h) = \{n \in \mathbb{Z} : (n, h) \in \mathcal{F}(\mathcal{D})\}$.

Now, we show how the inverted lists $L(h)$ can be used to accelerate the retrieval process. Denote by Q a query data item and $C(Q) \subseteq \mathbb{Z} \times \mathcal{H}$ it's feature representation. Let $\Delta : \mathbb{Z} \to \mathbb{R}$ be the similarity measure in (3.5.6). Our goal is to find $m^* = \arg\max_{m \in \mathbb{Z}} \Delta(m)$. Observe that a point $(n, h) \in \mathcal{F}(Q)$ contributes to $\Delta(m)$ if and only if

$$\begin{aligned} m + (n, h) \in \mathcal{F}(D) &\iff (m + n, h) \in \mathcal{F}(\mathcal{D}) \\ &\iff m + n \in L(h) \\ &\iff m \in L(h) - n. \end{aligned}$$

This equivalence shows that in order to find m^*, one needs to consider the lists $L(h) - n$, and find the m that most frequently occurs in the sets $L(h) - n$, for all $(n, h) \in \mathcal{F}(Q)$. This also gives an alternative expression of Δ via the indicator functions:

$$\Delta_Q(m) = \frac{\sum_{(n,h) \in C(Q)} \mathbb{1}_{L(h)-n}(m)}{|C(Q)|}. \tag{3.5.7}$$

Now let us analyze the complexity of processing a query Q, using the inverted list method. Denote by $N = |\mathcal{F}(\mathcal{D})|$ the number of database items, and by $M = |\mathcal{F}(Q)|$, the number of query items. In practice, M is negligible compared to N. Let $L = |\mathcal{H}|$ the number of inverted lists. Assuming that the hash function uniformly distributes the database items to the inverted lists, each of the L lists contains roughly N/L elements. Since on the query time only M of the inverted lists need to be processed, we conclude that the complexity in processing a query is linear in

$$\frac{M \cdot N}{L}. \tag{3.5.8}$$

Observe that this is still linear in the database size N, but it is reduced by a factor of L when

compared to the exhaustive approach which requires $M \cdot N$ operations. In our case, the set of frequency bins $[0 : K]$, obtained from the STFT corresponds to the set of possible hashes $\mathcal{H}$. Therefore, if for example the window length of the Fourier Transform is 2048 samples, then $L = 1024$. This means that the retrieval process will be reduced by a factor of 1024 when compared to the brute-force approach. However, for large databases containing millions of recordings, much larger speed-up factors are needed to capture the necessities of a real-time scenario. One idea to further reduce the query time would be to increase the window of the STFT in order to increase L as well. This, however, would drastically reduce the overall performance of the system, since higher frequency resolution would increase the probability of a peak point (n, h) to move to a neighboring point (n, h'), when the query audio fragment is distorted with ambient noise.

An alternative approach adopted by Wang [1] in Shazam's audio identification system is to enlarge the set of hashes $\mathcal{H}$ by considering pairs of peaks instead of individual peaks. To this end, one fixes a point $(n_0, k_0) \in C(\mathcal{D})$, called the *anchor point*, and a rectangular region $T_{(n_0,k_0)} \subseteq \mathbb{Z} \times [0 : K]$ around that point. Then, instead of taking the single point corresponding to the frequency stamp, one considers all triplets of the form

$$(k_0, k_1, n_1 - n_0), \tag{3.5.9}$$

for every $(n_1, k_1) \in T_{(n_0,k_0)} \cap C(\mathcal{D})$ and hashes these three values. Assuming that the peak coordinates are uniformly distributed on the spectrogram of an audio clip, the number of points in every region $T_{(n_0,k_0)}$ will be roughly the same. This number is called the *fan-out* and it is denoted by F. To compare this triplet-hash approach with the single-hash approach we suppose that each of the three numbers $k_0, k_1, n_1 - n_0$ is represented by B bits. Then, the single-hash approach has 2^B hashes, whereas the triplet-hash approach has $2^B \cdot 2^B \cdot 2^B = 2^{3B}$ hashes in total. As before, let $N = |\mathcal{F}(\mathcal{D})|$ be the number of database items, $M = |\mathcal{F}(\mathcal{Q}|$ the number of peak coordinates of a query $\mathcal{Q}$, and $L = 2^B$ the number of hashes of the single-value approach. If we assume that the database items are evenly distributed on the inverted lists, in the triplet-hash approach, each list will roughly contain N/L^3 items. On the other hand, the number of triplets on the query $\mathcal{Q}$ are $F \cdot M$. Therefore, we deduce that complexity of the retrieval time is linear in

$$\frac{F}{L^2} \cdot \frac{M \cdot N}{L}. \tag{3.5.10}$$

This shows, that by considering three values for hashing instead of one, reduces the search time on the database to find the best match by a factor of L^2/F. For example, if $F = 10$ and we use 10 bits to represent each value, then the response time is reduced by a factor $2^{20}/10 \approx 10000$ when compared to the single-hash approach. In Chapter 6, we present an experimental use-case of an audio fingerprinting system that utilizes these ideas by taking advantage of the abundance of free software on the Web. Furthermore, we set up two different evaluation protocols to measure the performance of the system, and compare it with the approach that leverages the ideas of deep learning, as presented in the upcoming chapters.

Chapter 4

Deep Learning

Deep learning has emerged as a revolutionary approach to artificial intelligence (AI) and has gained immense popularity over the last decade. It has revolutionized numerous fields, including computer vision, natural language processing, music recognition, and many others. A concrete example is ChatGPT (Chat Generative Pre-Trained Transformer), an artificial intelligence chat bot released by OpenAI on November 30, 2022 that can engage in insightful and meaningful conversations in an unprecedented manner.

One of the key reasons for the popularity of deep learning is its ability to learn and extract complex patterns and representations from vast amounts of data. Traditional machine learning algorithms often rely on handcrafted features, which can be time-consuming and challenging to design. Deep learning, on the other hand, uses neural networks with multiple layers, known as deep neural networks, to automatically learn these features directly from the raw data. In other words, deep learning models relieve humans from the burden of having to come up with meaningful data representations in order to train and develop a model for a given task. The core theme of deep learning is to make the model to discover not only the mapping from representation to output, but also to discover the representation itself. We can loosely say that deep learning is the hack to many problems that require subjective and intuitive reasoning in order to make a machine to behave in an intelligent way. Before the advent of deep learning, these problems required an immense intellectual effort from humans to coin sophisticated algorithms and procedures for efficient solutions. In addition, these solutions were difficult, if not impossible, to transfer to different areas and domains. Deep learning has surpassed by a huge margin these approaches in many cases with only a few simple rules and by leveraging the advances in computational processing power, which is usually taken for granted nowadays.

The purpose of this chapter is to provide a quick overview of deep learning and motivate the methodology of deep audio fingerprinting adopted in the next chapter for the song identification task. We present the basic principles underlying the theory of deep learning, and introduce the type of architectures that we will use in the next sections. For a more comprehensive exposition on the subject, the reader may refer to [20].

4.1 Deep Feedforward Networks

4.1.1 The Architecture

Deep feedfoward networks, also called feedfoward neural networks, or multilayer perceptrons (MLPs), form the most fundamental class of deep learning models. The goal of these models is to approximate a target function f^*, by employing linear mappings with activation functions. Formally, a feedforward neural network can be represented by a function $f : \mathbb{R}^K \to \mathbb{R}^N$, mapping points from the Euclidean space $\mathbb{R}^K$ to $\mathbb{R}^N$. The function f can be expressed as the composition

$$f = f^{(n)} \circ f^{(n-1)} \circ \cdots \circ f^{(1)}, \tag{4.1.1}$$

where each $f^{(i)} : \mathbb{R}^{K_i} \to \mathbb{R}^{K_{i+1}}$ is a linear mapping $T_i : \mathbb{R}^{K_i} \to \mathbb{R}^{K_{i+1}}$ followed by the elementwise composition with a non-linear mapping $\sigma_i : \mathbb{R} \to \mathbb{R}$. We denote this operation by $\sigma_i \otimes T_i$. Since every linear mapping $T_i : \mathbb{R}^{K_i} \to \mathbb{R}^{K_{i+1}}$ can by expressed by as $T(x) = W_i \cdot x$, where $W_i \in \mathbb{R}^{K_{i+1} \times K_i}$, we can write $(\sigma_i \otimes T_i)(x) = \sigma(W_i \cdot x)$, where for a vector $W_i \cdot x = (v_1, \ldots, v_{K_{i+1}}) \in \mathbb{R}^{K_{i+1}}$ we have that

$$\sigma(W_i \cdot x) = \left(\sigma(v_1), \ldots, \sigma(v_{K_{i+1}})\right).$$

In (4.1.1), f_1 is called the first layer, or input layer, f_2, the second layer, and so on. The last layer is called the output layer. The overall length of the chain defines the depth of the model. The matrices $W_1, \ldots, W_n$ are the weights of the model and their entries constitute the learnable parameters of the model. These parameters are collectively denoted by $\boldsymbol{\theta}$, and the value of f at a point x by $f(x; \boldsymbol{\theta})$. In practice, one adds a bias weight b_i to the linear mapping T_i to allow to capture regions in R^N that not necessarily centered at the origin. Therefore, the mappings T_i correspond to affine transformations of the form $T_i(x) = W_i \cdot x + b_i$. The main goal is to determine the parameters $\boldsymbol{\theta}$ such that for every point x, $f(x; \boldsymbol{\theta})$ would be close to the target function f^*. In practice, a finite set of observations $\{x_1, \ldots, x_n\}$ are given, accompanied by the labels $\{y_1, \ldots, y_n\}$, serving as approximate examples of $y_1 \approx f^*(x_1), \ldots, y_n \approx f^*(x_n)$. The set $\{(x_1, y_1), \ldots, (x_n, y_n)\}$ is called the training set, and specifies what the output layer must do at each point x_i; it must produce a value close to y_i. To this end, the learning algorithm must decide how to adjust the parameters $\boldsymbol{\theta}$ of the model to produce the desired output. Since the behavior of the other layers is not directly specified by the training data, but rather depending on the learning algorithm, these layers are called hidden layers. The dimensionality of these hidden layers determines the width of the model. Inspired by neuroscience, hence the name neural, these networks are graphically depicted as a direct acyclic graph as in Figure 4.1. The vertices are called units or neurons, and represent the elements of the vectors in each of the linear mappings comprising the network. The edges correspond to the weights of each linear mapping. For instance, in Figure 4.1 the first two layers correspond to a linear mapping $T : \mathbb{R}^3 \to \mathbb{R}^4$. The edges are the entries of the matrix $W \in \mathbb{R}^4 \times \mathbb{R}^3$ of the linear mapping T for which $T(x) = W \cdot x$, for every $x \in \mathbb{R}^3$.

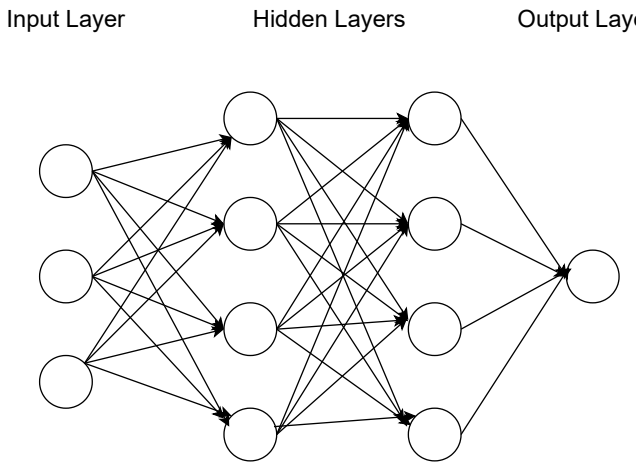

Figure 4.1. *Graphical Representation of a FeedForward Neural Network.*

The parallelism with neuroscience comes from the fact that the neurons in the network loosely resemble the organization of neurons in the brain. In the brain, information typically flows in a forward direction from sensory receptors to high-level processing areas. Each neuron receives inputs from other neurons, performs a computation, and transmits the result to other neurons. In our case, we will treat neural networks as pure mathematical entities, i.e., as functions that map low-level features extracted from raw data (audio data in our case) to discriminative high-level features that capture the essential parts of the raw information in a compressed form.

4.1.2 Activation functions

The key feature of neural networks that discriminates them from the linear models are the activation functions, employed after each linear transformation. These activation functions are non-linear and are important for several reasons:

1) **Non-linearity:** Non-linear activation functions introduce non-linearity into the neural network. Without non-linear activation functions, a neural network would simply be a linear combination of its inputs, and it would not be able to learn and represent complex, non-linear relationships in the data. Non-linear activation functions enable neural networks to model and approximate complex functions, making them capable of capturing intricate patterns and relationships in the data.

2) **Representation power:** Non-linear activation functions significantly enhance the representation power of neural networks. By introducing non-linear transformations, neural networks can model complex mappings between inputs and outputs. This allows them to learn and represent highly non-linear and abstract concepts, enabling more powerful and flexible modeling of real-world phenomena.

3) **Gradient propagation:** During the training process, neural networks rely on gradient-based optimization algorithms, such as backpropagation, to update their weights and biases. Non-linear activation functions ensure that the gradients can flow backward through the network and propagate the error signal effectively. If only linear activation functions were used, the

gradients would simply be scaled versions of the input, leading to limited learning capability and inefficient training.

4) **Decision boundaries:** Non-linear activation functions enable neural networks to learn decision boundaries that are not constrained to linear separations. By introducing non-linearities, neural networks can learn complex decision boundaries that can separate data points of different classes more effectively. This is particularly important in tasks such as image recognition, natural language processing, and other tasks where the data exhibits complex relationships.

In this paragraph we introduce some of the most well-known activation functions that are used in many deep learning applications. The choice of activation function depends on the specific task at hand. Changing the activation function of a neural network can drastically affect the overall performance. Many surveys have been conducted [21, 22] comparing the most frequently used activation functions in various applications and different types of networks.

Sigmoid Function

The sigmoid activation function, referred to as the logistic function or squashing function, is given by

$$S(x) = \frac{1}{1+e^{-x}} = \frac{e^{x}}{1+e^{x}}. \tag{4.1.2}$$

It is easily seen that S is everywhere differentiable with derivative

$$S'(x) = \frac{e^{-x}}{(1+e^{-x})^2} = S(x)\cdot(1-S(x)).$$

Since $\lim_{x\to\infty} S(x) = 1$, $\lim_{x\to-\infty} S(x) = 0$, and $S(x) > 0$, it follows that the range of S is the open interval $(0, 1)$. Figure 4.2 shows the graph of the sigmoid function and its derivative. The sigmoid function appears in the output layers of the DL architectures, and it is used for predicting probability based output. A one-layer feedforward neural network, followed by a sigmoid activation function defines the logistic regression

$$f(x) = \frac{1}{1+e^{-(b_0+b_1x+\dots b_nx^n)}}, \tag{4.1.3}$$

one of the most widely used machine learning models in medical literature [23]. However, while the sigmoid function might be a good choice for shallow networks, it falls short when the depth of the network grows larger. This is because all of the learning algorithms update the model's parameters $\boldsymbol{\theta}$ by taking steps towards the negative of the gradient $\nabla_{\boldsymbol{\theta}}J$, for a cost function J, in order to minimize it. Since the derivative of the sigmoid function vanishes for large values, this may stop the training procedure from finding better parameters $\boldsymbol{\theta}$. This problem is known as the vanishing gradients problem in DL.

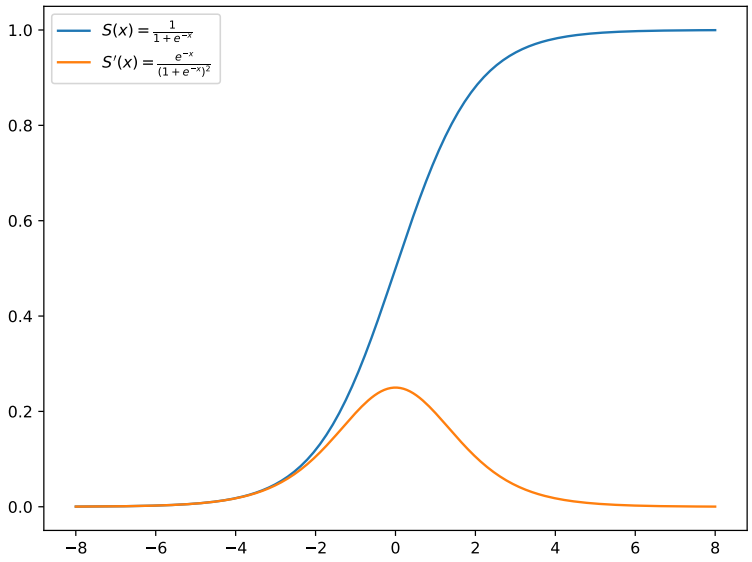

Figure 4.2. *The sigmoid function and its derivative.*

Hyperborlic Tangent Function

The hyperbolic tangent function is another type of activation function used in DL. The hyperbolic function, denoted by tanh, is a smooth, zero-centered function with range in $(-1, 1)$. It is given by

$$\tanh x = \frac{e^x - e^{-x}}{e^x + e^{-x}}. \tag{4.1.4}$$

Figure 4.3 shows the graph of tanh. Compared to the sigmoid function, tanh has steeper gradients, especially around the origin. This can lead to stronger gradients and faster learning, particularly in the first stages of training. However, similar to the sigmoid function, tanh is prone to saturation, especially for input values far from zero. Apart from that, tanh maps input values to the range $(-1, 1)$. While this range may be suitable for certain tasks, it might not be ideal for others. Finally, tanh is computationally expensive as it involves more complex operations compared to other activations. The exponentiations in 4.1.4 can be computationally expensive, particularly when dealing with large-scale neural networks or real-time applications.

Softmax Function

Softmax is the multi-dimensional analog of the sigmoid function. It is commonly used in the output layer of neural networks to map a vector $x = (x_1, \ldots, x_n)$ to a probability vector. It is given by

$$\text{Softmax}(x) = \left(\frac{e^{x_1}}{\sum_{i=1}^n e^{x_i}}, \ldots, \frac{e^{x_n}}{\sum_{i=1}^n e^{x_i}} \right), \tag{4.1.5}$$

for every $x = (x_1, \ldots, x_n) \in \mathbb{R}^n$. The softmax function is well-suited for multiclass classification problems where there are more than two classes. It assigns probabilities to each class, providing

a way to rank and compare multiple classes simultaneously. Moreover, the softmax function is differentiable, which is crucial for backpropagation and gradient-based optimization algorithms. The gradients of the softmax function can be efficiently calculated, allowing for efficient training of the neural network.

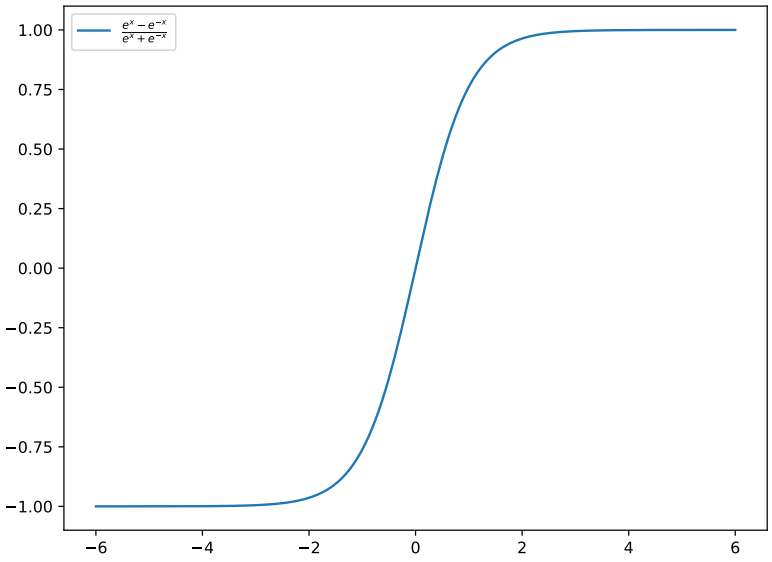

Figure 4.3. *The graph of* tanh.

Rectified Linear Unit (ReLU) Function

The rectified linear unit (ReLU) activation function was proposed by Nair and Hinton in 2010 [24], and ever since, has been the most widely used activation function for deep learning applications. It is given by

$$\mathrm{ReLU}(x) = \begin{cases} x, & x \geq 0 \\ 0, & x < 0 \end{cases} . \tag{4.1.6}$$

ReLU is simple and efficient. It involves only a comparison and a linear function, making it computationally efficient compared to other activation functions that involve more complex operations. As opposed to the sigmoid and tanh, ReLU helps alleviate the vanishing gradient problem, which can occur when training deep neural networks. The derivative of ReLU is either 0, or 1, which means it doesn't suffer from saturation issues like sigmoid or tanh. This allows the gradient to flow more effectively, facilitating better gradient-based optimization and avoiding the problem of vanishing gradients. Furthermore, ReLU encourages sparsity in the activations of the neural network. Since ReLU outputs 0 for negative values, it results in a more sparse representation where only a subset of neurons is activated. This sparsity can help in reducing the computational load and overfitting, as fewer neurons need to be computed and updated during training. On the contrary, the zero part of ReLU can lead to "dead" neurons that are non-responsive and never activate. If a ReLU neuron gets stuck in the negative region and consistently outputs 0, it effectively becomes a

dead neuron and does not contribute to the learning process. Dead neurons can hinder the learning capacity of the network. Another drawback, is that ReLU is not bounded above, which means it can produce very large activations. In some cases, unbounded activations can lead to numerical instability or exploding gradients during training. Despite these disadvantages, the simplicity, non-linearity, and effectiveness of ReLU have made it popular choice in many deep learning applications. Below we see some variations of ReLU that researchers have developed to address some of its drawbacks.

Leaky ReLU (LReLU)

The Leaky Rectified Linear function (LReLU), introduced in [25], was applied to an acoustic model, exhibiting improved results compared to ReLU. It is given by

$$\text{LReLU}(x) = \begin{cases} x, & x > 0 \\ \frac{x}{a}, & x \leq 0 \end{cases}, \tag{4.1.7}$$

where a is a fixed parameter in the range $(1, +\infty)$. In [25] the authors suggest to set a to a large number like 100. The parameter a was introduced to address the problem of "dead" neurons in ReLU, and can lead to faster convergence during training compared to ReLU. Other variants of rectified activations exists, such as Parametric Rectified Linear Unit (PReLU) and Randomized Leaky Rectified Linear Unit (RReLU). In [26] you can see an experimental comparison of these activations in an image classification task. Figure 4.4 shows the graph of ReLU and Leaky ReLU for different values of the parameter a.

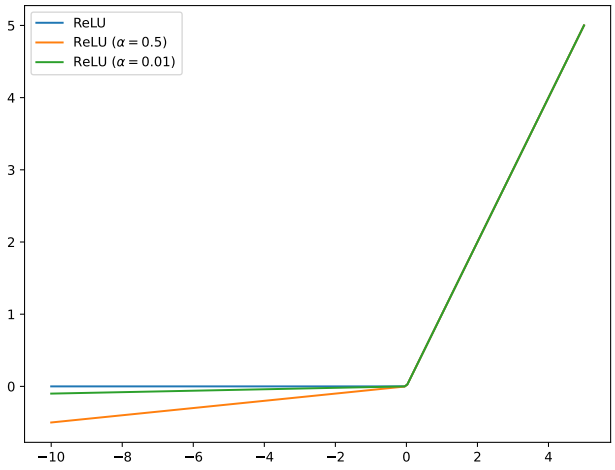

Figure 4.4. *ReLU and Leaky ReLU activation functions.*

Exponential Linear Unit (ELU)

The exponential linear units (ELUs) is another type of activation function introduced in [27], and they are used to speed up the training of deep neural networks. ELU also introduces a small slope

for negative input values but goes a step further by using an exponential function for negative values. The main advantage of ELUs is that they can alleviate the vanishing gradient. It is given by

$$\text{ELU}(x) = \begin{cases} x, & x > 0 \\ a \cdot \exp(x) - 1, & x \leq 0 \end{cases} . \tag{4.1.8}$$

Figure 4.5 shows the graph of ELU for $a = 1$, along with the graph of ReLU.

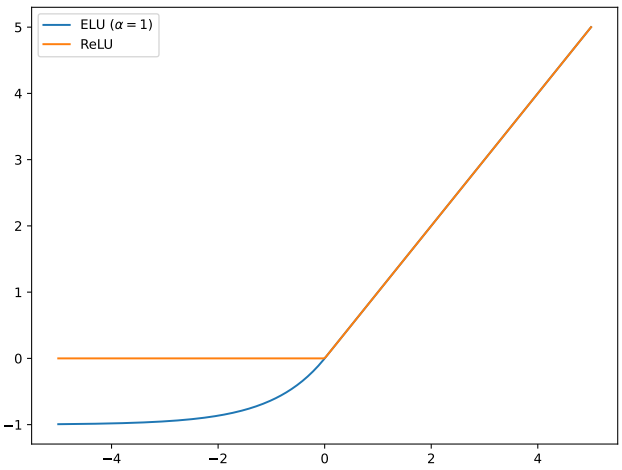

Figure 4.5. *The graph of ELU and ReLU.*

4.2 The Learning Paradigm

4.2.1 Supervised, Unsupervised, and Self-Supervised Learning

As we have already stated, the core theme of DL is to approximate a target function f^* through a parametric class of functions (neural networks) $\{f(x;\boldsymbol{\theta}) : \boldsymbol{\theta} \in \mathbb{R}^N\}$ defines the parametric space of the model. To this end, one needs to adopt an appropriate loss function $J(x;\boldsymbol{\theta})$, and a learning algorithm in order to find the optimal parameters $\boldsymbol{\theta}^*$ such that $f(x;\boldsymbol{\theta}) \approx f^*(x)$ for all x. All learning algorithms are variations of the gradient descent algorithm, and they are called gradient based algorithms. However, the choice of loss function is dependent at the specific task at hand. In DL, and in Machine Learning in general, there are three fundamental types of problems:

1) Supervised learning is the most well-known and widely used paradigm, it involves training a model on labeled data, where each example is associated with a known target or a class label. Supervised learning encompasses tasks such as classification, where the goal is to assign predefined labels to new data, and regression, which involves predicting continuous values. Formally, in a classification task, we are given a set of observations $\{(x_1, y_1), \ldots, (x_n, y_n)\}$, where x_i is a feature vector in a high-dimensional Euclidean space, extracted from the raw data, and $y_i \in [0 : K-1]$ is the class label indicating in which class the observation x_i belongs to. A standard assumption in this context is that the features are generated by a distribution $\mathcal{D}$. Given a class of neural networks

$f(\cdot;\boldsymbol{\theta}) : \mathbb{R}^M \to \mathbb{R}^K$, and a loss function $J(x, y)$, the goal is to find the parameters $\boldsymbol{\theta} \in \mathbb{R}^N$ that minimize the expected generalization loss

$$\arg\min_{\boldsymbol{\theta}} \mathbb{E}_{x\sim\mathcal{D}} \left[J\left(f(x;\boldsymbol{\theta}), y\right)\right]. \tag{4.2.1}$$

Given the estimated parameters $\boldsymbol{\theta}_*$ obtained from the learning algorithm, the model $f(\cdot\,;\boldsymbol{\theta}_*)$ can be used to make predictions to unseen data $x \sim \mathcal{D}$ and categorize the sample x to one of the K classes. In the multiclass case ($K > 2$), the most widely used loss function is the *Cross Entropy Loss* function . The model is designed in such a way that the last output layer corresponds to a linear mapping to K dimensions. Therefore, $f(x)$[1] is a vector $(z_1, \ldots, z_K)$ in $\mathbb{R}^K$. If $y_x \in [1, K]$ is the label of x, the cross entropy loss function is given by

$$J(f(x), y_x) = -\sum_{i=1}^{K} \log \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \cdot \mathbb{1}_{i=y_x}. \tag{4.2.2}$$

Observe that the cross entropy loss applies a softmax activation function to map the output of the last layer $(z_1, \ldots, z_K)$ to a probability vector. In this way, during the prediction phase, for a sampled data point $x \sim \mathcal{D}$, the model assigns the samples to the class determined by

$$\arg\max_{1\le i\le K} \frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}}, \tag{4.2.3}$$

where $f(x) = (z_1, \ldots, z_n)$ is the output of the model's last layer. On the other hand, in a regression task, the target function f^* takes real values and the goal to find the best possible parameters $\boldsymbol{\theta}$ such that $f(x;\boldsymbol{\theta}) \approx f^*(x)$, for all $x \in \mathbb{R}^M$. In this case, the set of observations $\{(x_1, y_1), \ldots, (x_n, y_n)\}$ contains tuples (x_i, y_i), where x_i is the feature representation obtained from the raw data, and $y_i \in \mathbb{R}$ is the target value associated with x_i, i.e. $f^*(x_i) = y_i$. A standard loss function in this setup is the *Mean Squared Error* (MSE)

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n} (\hat{y}_i - y_i)^2\,, \tag{4.2.4}$$

where $\hat{y}_i = f(x_i;\boldsymbol{\theta})$. This also referred to as the L_2-loss. Another common choice is the *Mean Absolute Error* (MAE) or L_1-loss

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n} (\hat{y}_i - y_i)\,. \tag{4.2.5}$$

In general, one defines the L_p-loss for every $p > 1$ by

$$L_p\text{-loss} = \frac{1}{n}\sum_{i=1}^{n} (\hat{y}_i - y_i)^p\,. \tag{4.2.6}$$

2) Unsupervised learning on the contrary deals with observations $\{x_1, \ldots, x_n\}$ without having any information about the class labels of the feature vectors x_i. In this type of problem, the goal

[1] For simplicity we omit the parameters $\boldsymbol{\theta}$.

is to unravel the underlying similarities and cluster (group) "similar" vectors together [28]. These clusters must be as compact as possible and the instances in different clusters must be different as much as possible. There is an extensive research literature around clustering. Some of the main challenges are to define appropriate similarity or dissimilarity measures, and efficient algorithms of low complexity. In this thesis, we focus on the supervised and self-supervised aspects of deep learning. The reader may refer to [29] for a quick introduction to clustering and its challenges.

3) Self-supervised learning, a prominent deep learning technique, is poised to overcome the challenges associated with the heavy reliance on labeled data in supervised settings. Training a neural network effectively demands a vast amount of training data. However, in the supervised setting, this data needs to be preprocessed and labeled, which requires laborious work. This process is not always feasible, and it can also introduce errors, thereby deteriorating the data quality due to human fallibility. The goal of supervised learning is to alleviate these problems by utilizing the raw data, and extract feature representations that can be used to downstream tasks. In the upcoming chapter, we will introduce the *contrastive learning* framework, a prominent self-supervised technique that has gained a lot of attention recently, achieving state-of-the-art performance when applied to the downstream task of image classification [30]. In our case, we will make use of this framework to extract robust feature representations of the audio fragments in order to set up a completely different song identification system from the one we presented in Section 3.5.

4.2.2 Gradient Based Algorithms

In the previous sections, we introduced a specific class of neural networks, the *feedforward neural networks*. As we have already stated, in any deep learning task, whether is a supervised or self-supervised task, the goal is to find the parameters $\boldsymbol{\theta}$ that minimize the generalization error in (4.2.1), for a loss function J. As opposed to convex optimization where we aim for the global minimum of J, in deep learning we are happy with local minimums as well, provided that the empirical loss

$$\frac{1}{n}\sum_{i=1}^{n} J\left(f(x^{(i)};\boldsymbol{\theta}), y^{(i)}\right), \tag{4.2.7}$$

is sufficiently low, on a training set $\{(x^{(1)}, y_1), \ldots, (x^{(n)}, y_n)\}$. In this section, we introduce the *stochastic gradient descent* and some of its most widely used variants utilized nowadays by the community to train deep neural networks.

Stochastic Gradient Descent

Stochastic gradient descent (SGD) is a fundamental machine learning algorithm and one of the most important ones. It is widely used due to its simplicity and effectiveness. Many learning algorithms used nowadays are variations built upon the SGD algorithm. The idea of SGD is simple: Given a class of neural networks $f(x;\boldsymbol{\theta})$, a training set of examples $\{(x_1, y_1), \ldots, (x_n, y_n)\}$, and a loss function $J\left(f(x;\boldsymbol{\theta}), y\right)$, one examines one-by-one the samples (x_i, y_i) and updates the parameters of the neural network by taking small steps towards to the opposite direction of the gradient:

$$\boldsymbol{\theta}^{(i+1)} = \boldsymbol{\theta}^{(i)} - \eta \cdot \nabla_{\boldsymbol{\theta}} J\left(f(x^{(i)};\boldsymbol{\theta}^{(i)}), y^{(i)}\right). \tag{4.2.8}$$

The parameter η is called the *learning rate* and determines the size of each step. The gradient $\nabla_{\boldsymbol{\theta}}$ shows the direction where the function J experiences the most rapid increase in an infinitesimal region. By analogy, the negative of the gradient shows the direction of the most rapid decrease. This version of SGD is also referred to as online SGD, since it processes the examples one-by-one. In practice, when training a deep neural network one iterates over all the examples of the training set $\{(x_1, y_1), \ldots, (x_n, y_n)\}$[2], and updates the parameters at each example. Once the training set is exhausted, the examples are shuffled and the process is repeated until a predetermined number of epochs or a termination criterion is met. A complete pass over the training set defines an *epoch*. To define the termination criterion and to evaluate the generalization performance of the obtained model at the end of the SGD, one partitions the set of examples $\{(x_1, y_1), \ldots, (x_n, y_n)\}$ into three subsets; the *training set* S_{train}, the *validation set* S_{val}, and the *test set* S_{test}. The validation set determines the termination criterion; at the end of each epoch this set is used to evaluate the empirical validation loss

$$\text{Val Loss} = \frac{1}{|S_{\text{val}}|} \sum_{(x,y)\in S_{\text{val}}} J\left(f(x;\boldsymbol{\theta}), y\right). \tag{4.2.9}$$

A criterion commonly used in this case is called *early stopping*, which is a form of regularization. In early stopping, one defines a maximum number of epochs N, and a number of epochs P, called the *patience*. At the end of each epoch $1 \leq K < N$, the minimum value of (4.2.9) across the epochs $1, \ldots, K$ is tracked. If this value has not changed for $j = P$ consecutive epochs the training stops. Otherwise, if this value changes at epoch K, then j is set to 0. If the validation loss decreases before reaching P consecutive epochs without a change the training stops at the end of the N-th epoch. At the end, the model scoring the lowest validation loss during the training stage is the final model. Then, to measure the generalization performance the test set S_{test} is used; the loss

$$\text{Test Loss} = \frac{1}{|S_{\text{test}}|} \sum_{(x,y)\in S_{\text{val}}} J\left(f(x;\boldsymbol{\theta}), y\right), \tag{4.2.10}$$

is an indicator on how well the final model generalizes to new unseen data. While the update rule in (4.2.8) is the most fundamental one; it is rarely used in practice. Processing the examples one-by-one makes the training procedure very sensitive to unrepresentative data points that might not reflect intrinsic characteristics of the data, leading to unstable convergence. Another issue is that this approach limits the potential for parallelization resulting to long training times. A common approach to eliminate this problem is to process the training example in mini-batches of fixed size M. In this setting the training set is split into $|S_{\text{train}}|/M$-groups of size M and the network process each batch one-by-one instead of the individual examples. The update rule in this case is

$$\boldsymbol{\theta}^{(k+1)} = \boldsymbol{\theta}^{(k)} - \eta \cdot \frac{1}{M} \sum_{j=1}^{M} \nabla_{\boldsymbol{\theta}^{(k)}} J\left(f(x^{(kj)};\boldsymbol{\theta}^{(k)}), y^{(kj)}\right), \tag{4.2.11}$$

where k corresponds to the k^{th} batch $\{x_{k1}, \ldots, x_{kM}\} \subseteq S_{\text{train}}$. The update rule in (4.2.11) defines the *mini-batch stochastic gradient descent* which is the predominate approach to train deep neu-

[2]For the moment, we only discuss the supervised setting. Later on, we will also focus on self-supervised tasks. The theory doesn't change that much since these problems reduce to the supervised case.

ral networks nowadays. In 4.2.11 we described the mini-batch SGD using a fixed learning rate. However, a common practice is to gradually decrease the learning rate at the end of each epoch by adopting a specific learning rate schedule. From now on, we denote the learning rate at epoch k as η_k. Algorithm 4.1 describes the training procedure using the mini-batch approach.

ALGORITHM 4.1: *Mini Batch Stochastic Gradient Descent*

Input: Number of epochs N, batch size M.
Input: Initial parameters $\boldsymbol{\theta}^{(0)}$.
Input: Starting learning rate η_0 with a learning rate schedule η_n.
Input: Training set $S_{\text{train}} = \{(x^{(1)}, y^{(1)}), \ldots, (x^{(n)}, y^{(n)})\}$.
for all $n = 0, \ldots, N-1$ **do**
  Shuffle and group the training set $S_{\text{train}} = \bigcup_{k=0}^{K-1} \{x_{k1}, \ldots, x_{kM}\}$, where $K = |S_{\text{train}}|/M$.
  for all $k = 0, \ldots, K-1$ **do**
    $\boldsymbol{\theta}^{(\text{old})} \leftarrow \boldsymbol{\theta}^{(n+k)}$
    Compute gradient estimate: $\hat{J} \leftarrow \frac{1}{M} \sum_{j=1}^{M} \nabla_{\boldsymbol{\theta}^{(\text{old})}} J\left(f(x^{(kj)}; \boldsymbol{\theta}^{(k)}), y^{(kj)}\right)$
    Apply update rule: $\boldsymbol{\theta}^{(\text{new})} \leftarrow \boldsymbol{\theta}^{(\text{old})} - \eta_n \hat{J}$

4.3 Optimizers

In this section we present variants of the SGD algorithms that are used to overcome some the drawbacks posed by the vanilla SGD. These variants are known as *optimizers* and play a vital role in deep learning as they contribute to the efficiency and effectiveness of the training process. A study comparing the performance of some of these optimizers in deep learning can be found in [31].

4.3.1 SGD with momentum

While stochastic gradient descent remains a popular optimization strategy, learning with it can sometimes be slow. The method of momentum [32] is designed to accelerate learning. The momentum algorithm accumulates an exponentially decaying moving average of past gradients and continues to move in their direction [20]. The momentum algorithm introduces a variable $\boldsymbol{v}$, containing the information of the direction of all past gradients. A hyperparameter $a \in [0, 1)$ controls the amount of contribution of $\boldsymbol{v}$ on the calculation of the new direction. Formally, assuming $\boldsymbol{\theta}^{(0)}$ are the starting parameters of the model, then $\boldsymbol{v}$ is calculated by the update rule:

$$\boldsymbol{v}^{(\text{new})} \leftarrow a\boldsymbol{v}^{(\text{old})} - \eta \frac{1}{M} \sum_{j=1}^{M} \nabla_{\boldsymbol{\theta}^{(\text{old})}} J\left(f(x^{(kj)}; \boldsymbol{\theta}^{(k)}), y^{(kj)}\right) \tag{4.3.1}$$

The new parameters $\boldsymbol{\theta}^{(\text{new})}$ are given by $\boldsymbol{\theta}^{(\text{new})} \leftarrow \boldsymbol{\theta}^{(\text{old})} + \boldsymbol{v}^{(\text{new})}$. The goal of momentum is to mitigate the problem of variance that might occur in vanilla SGD. Some updates in SGD might lead the parameters away from low values of the loss function J due to the high variance in the data. For example, if a batch sample consists of unrepresentative samples, then the SGD might completely change the direction. On the other hand, the parameter $\boldsymbol{v}$ controls this behavior by

always adding a component towards the average of the past directions. Algorithm 4.2 summarizes the SGD with momentum.

Algorithm 4.2: *Stochastic Gradient Descent with Momentum*

Input: Number of epochs N, batch size M.
Input: Initial parameters $\boldsymbol{\theta}^{(0)}$.
Input: Starting learning rate η_0 with a learning rate schedule η_n.
Input: Momentum parameter $a \in [0, 1)$.
Input: Training set $S_{\text{train}} = \{(x^{(1)}, y^{(1)}), \ldots, (x^{(n)}, y^{(n)})\}$.
$\boldsymbol{v}^{(\text{old})} \leftarrow 0$.
for all $n = 0, \ldots, N-1$ **do**
  Shuffle and group the training set $S_{\text{train}} = \bigcup_{k=0}^{K-1} \{x_{k1}, \ldots, x_{kM}\}$, where $K = |S_{\text{train}}|/M$.
  for all $k = 0, \ldots, K-1$ **do**
    Update the velocity $\boldsymbol{v}^{(\text{new})} \leftarrow a\boldsymbol{v}^{(\text{old})} - \eta_n \frac{1}{M} \sum_{j=1}^{M} \nabla_{\boldsymbol{\theta}^{(\text{old})}} J\left(f(x^{(kj)}; \boldsymbol{\theta}^{(k)}), y^{(kj)}\right)$
    Apply update rule: $\boldsymbol{\theta}^{(\text{new})} \leftarrow \boldsymbol{\theta}^{(\text{old})} + \boldsymbol{v}^{(\text{new})}$.

4.3.2 AdaGrad

Experiments have shown that the learning rate is one of the most crucial parameters for training deep neural networks. Apart from various learning rate schedules that have been developed to gradually decrease the learning rate at the end of each training epoch, several optimization algorithms have been introduced that adapt the learning rates of model parameters. The first algorithm that we introduce and falls into this category is AdaGrad. The AdaGrad algorithm individually adapts the learning rates of all model parameters by scaling them inversely proportional to the square root of the sum of squared values of the past gradients. In Algorithm 4.3 you can see the update rule of AdaGrad.

Algorithm 4.3: *AdaGrad Algorithm*

Input: Number of epochs N, batch size M.
Input: Initial parameters $\boldsymbol{\theta}^{(0)}$.
Input: Starting learning rate η_0 with a learning rate schedule η_n.
Input: Small constant $\delta \approx 10^{-7}$, for numerical stability.
Input: Training set $S_{\text{train}} = \{(x^{(1)}, y^{(1)}), \ldots, (x^{(n)}, y^{(n)})\}$.
Initialize gradient accumulation variable $\boldsymbol{r}^{(\text{old})} = 0$.
for all $n = 0, \ldots, N-1$ **do**
  Shuffle and group the training set $S_{\text{train}} = \bigcup_{k=0}^{K-1} \{x_{k1}, \ldots, x_{kM}\}$, where $K = |S_{\text{train}}|/M$.
  for all $k = 0, \ldots, K-1$ **do**
    Compute gradient estimate: $\hat{J} \leftarrow \frac{1}{M} \sum_{j=1}^{M} \nabla_{\boldsymbol{\theta}^{(\text{old})}} J\left(f(x^{(kj)}; \boldsymbol{\theta}^{(k)}), y^{(kj)}\right)$.
    Accumulate squared gradient: $\boldsymbol{r}^{(\text{new})} \leftarrow \boldsymbol{r}^{(\text{old})} + \langle \hat{J}, \hat{J} \rangle$.
    Compute update: $\Delta\boldsymbol{\theta} \leftarrow -\frac{\eta_n}{\delta + \sqrt{\boldsymbol{r}^{(\text{new})}}} \cdot \hat{J}$.
    Apply update: $\boldsymbol{\theta}^{(\text{new})} \leftarrow \boldsymbol{\theta}^{(\text{new})} + \Delta\boldsymbol{\theta}$.

4.3.3 RMSProp

The RMSProp modifies the AdaGrad optimization algorithm by changing the accumulation into an exponentially weighted moving average. A parameter $p \in (0, 1)$ is initialized at the beginning of the algorithm, determining the rate of the decay. In this case, the update of the accumulation variable is given by the convex combination

$$p\boldsymbol{r}^{(\text{old})} + (1 - p)\langle \hat{J}, \hat{J} \rangle,$$

of the past accumulation variable $\boldsymbol{r}^{(\text{old})}$, and the new accumulation gradients $\langle \hat{J}, \hat{J} \rangle$. In 4.4 we summarize the RMSProp algorithm.

ALGORITHM 4.4: *RMSProp Algorithm*

Input: Number of epochs N, batch size M.
Input: Initial parameters $\boldsymbol{\theta}^{(0)}$.
Input: Starting learning rate η_0 with a learning rate schedule η_n.
Input: Small constant $\delta \approx 10^{-7}$, for numerical stability.
Input: Training set $S_{\text{train}} = \{(x^{(1)}, y^{(1)}), \ldots, (x^{(n)}, y^{(n)})\}$.
Initialize gradient accumulation variable $\boldsymbol{r}^{(\text{old})} = 0$.
for all $n = 0, \ldots, N - 1$ **do**
Shuffle and group the training set $S_{\text{train}} = \bigcup_{k=0}^{K-1} \{x_{k1}, \ldots, x_{kM}\}$, where $K = |S_{\text{train}}|/M$.
for all $k = 0, \ldots, K - 1$ **do**
Compute gradient estimate: $\hat{J} \leftarrow \frac{1}{M} \sum_{j=1}^{M} \nabla_{\boldsymbol{\theta}^{(\text{old})}} J\left(f(x^{(kj)}; \boldsymbol{\theta}^{(k)}), y^{(kj)}\right)$.
Accumulate squared gradient: $\boldsymbol{r}^{(\text{new})} \leftarrow \boldsymbol{r}^{(\text{old})} + \langle \hat{J}, \hat{J} \rangle$.
Compute update: $\Delta\boldsymbol{\theta} \leftarrow -\frac{\eta_n}{\delta + \sqrt{\boldsymbol{r}^{(\text{new})}}} \cdot \hat{J}$.
Apply update: $\boldsymbol{\theta}^{(\text{new})} \leftarrow \boldsymbol{\theta}^{(\text{new})} + \Delta\boldsymbol{\theta}$.

4.3.4 Adam

Adam [33] is one of the most popular adaptive learning rate optimization algorithms and is presented in 4.5. The method computes individual adaptive learning rates for different parameters from estimates of first and second moments of the gradients; the name Adam is derived from adaptive moment estimation. It combines the advantages of Adagrad 4.3, which works well with sparse gradients, and RMSProp 4.4, which works well in on-line and non-stationary settings.

4.3.5 LAMB

LAMB [34] is one the latest layerwise adaptive optimization algorithms designed to efficiently train deep neural networks by employing large batch sizes without degrading the performance. With the advent of large scale datasets, training deep neural networks, even using computationally efficient optimization algorithms like the variants of SGD presented in the previous paragraphs, has become particularly challenging. For instance, as reported in [34], training state-of-the-art deep learning models like BERT and ResNet-50 takes 3 days on 16 TPUv3 chips and 29 hours on 8 Tesla P100 GPUs respectively. These training times have lead to the development of optimization algorithms that scale to large batch sizes, by significantly reducing the training time of deep neural

networks. The authors in [34] managed to train BERT in 76 minutes by maintaining the same generalization performance. The term layerwise is derived from the fact that the weights of the deep neural network are updated for each layer separately. To formulate this procedure mathematically, suppose that the deep neural network $f : \mathbb{R}^K \to \mathbb{R}^N$ has h-layers. Let J be a loss function as in (4.2.8). Let $\mathbb{I}$ be the $d \times d$ identity matrix, and let $\mathbb{I} = [\mathbb{I}_1, \mathbb{I}_2, \ldots, \mathbb{I}_h]$ be its decomposition into column submatrices. For $x \in \mathbb{R}^d$, let $x^{(i)}$ be the block of variables corresponding to the columns of I_i, i.e., $x^{(i)} = I_i^T x \in \mathbb{R}^{d_i}$, for $i \in \{1, 2, \ldots, h\}$. Furthermore, let $\phi(z) = \min\{\max\{z, \gamma_l\}, \gamma_u\}$ for some constants $\gamma_l, \gamma_u \in \mathbb{R}$. The update rule of LAMB is the same with ADAM but in its layerwise version; i.e.,

$$\boldsymbol{\vartheta}^{(i)}_{(\text{new})} \leftarrow \boldsymbol{\vartheta}^{(i)}_{(\text{old})} - \eta \frac{\phi\left(\left\|\boldsymbol{\vartheta}^{(i)}_{(\text{old})}\right\|_2\right)}{\left\|r^{(i)}_{(\text{new})}\right\|_2} \cdot \underbrace{\frac{\boldsymbol{\sigma}^{(i)}_{(\text{new})}}{\sqrt{\boldsymbol{\rho}_{(\text{new})}} + \delta}}_{r^{(i)}_{(\text{new})}}, \tag{4.3.2}$$

where $\boldsymbol{\sigma}_{(\text{new})}, \boldsymbol{\rho}_{(\text{new})}$ are calculated as in 4.5, and $\boldsymbol{\vartheta}^{(i)}_{(\text{new})}$ refers to the block of variables corresponding to the i-th layer of neural network for $i \in \{1, \ldots, h\}$.

Algorithm 4.5: *Adam Algorithm*

Input: Number of epochs N, batch size M.
Input: Exponential decay rates for moment estimates, ρ_1 and ρ_2 in $[0, 1)$.
Input: Small constant δ used for numerical stabilization (Suggested value $\sim 10^{-8}$).
Input: Initial parameters $\boldsymbol{\vartheta}^{(0)}$.
Input: Starting learning rate η_0 with a learning rate schedule η_n.
Input: Training set $S_{\text{train}} = \{(x^{(1)}, y^{(1)}), \ldots, (x^{(n)}, y^{(n)})\}$.
Initialize 1st and 2nd moment variables $\boldsymbol{\sigma}^{(\text{old})} = 0, \boldsymbol{\rho}^{(\text{old})} = 0$.
for all $n = 0, \ldots, N-1$ **do**
Shuffle and group the training set $S_{\text{train}} = \bigcup_{k=0}^{K-1} \{x_{k1}, \ldots, x_{kM}\}$, where $K = |S_{\text{train}}|/M$.
for all $k = 0, \ldots, K-1$ **do**
Compute gradient estimate: $\hat{J} \leftarrow \frac{1}{M} \sum_{j=1}^{M} \nabla_{\boldsymbol{\vartheta}^{(\text{old})}} J\left(f(x^{(kj)}; \boldsymbol{\vartheta}^{(k)}), y^{(kj)}\right)$.
Update biased first moment estimate: $\boldsymbol{\sigma}^{(\text{new})} \leftarrow \rho_1 \boldsymbol{\sigma}^{(\text{old})} + (1 - \rho_1)\hat{J}$.
Update biased second moment estimate: $\boldsymbol{\rho} \leftarrow \rho_2 \boldsymbol{\rho}^{(\text{old})} + (1 - \rho_2) \cdot \hat{J} \odot \hat{J}$.
Correct bias in first moment: $\hat{\boldsymbol{\sigma}}^{(\text{new})} \leftarrow \frac{\boldsymbol{\sigma}^{(\text{new})}}{1 - \rho_1^{k+n+1}}$.
Correct bias in second moment: $\boldsymbol{\rho}^{(\text{new})} \leftarrow \frac{\boldsymbol{\rho}^{(\text{new})}}{1 - \rho_2^{k+n+1}}$.
Compute update: $\Delta\boldsymbol{\vartheta} = -\eta_n \cdot \frac{\boldsymbol{\sigma}^{(\text{new})}}{\sqrt{\boldsymbol{\rho}^{(\text{new})}} + \delta}$.
Apply update: $\boldsymbol{\vartheta}^{(\text{new})} \leftarrow \boldsymbol{\vartheta}^{(\text{old})} + \Delta\boldsymbol{\vartheta}$.

4.4 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are a class of deep learning models specifically designed for processing structured grid-like data, such as images, audio spectrograms, and 1D sequences. They are widely used for various tasks in computer vision including image classification [35], object detection [36], image segmentation [37], and more. Initially introduced by LeCun et al. [38], CNNs have revolutionized the field of image analysis achieving unprecedented success. The key idea behind CNNs is to leverage the local spatial correlations present in the input data by using

convolutional layers. In this section we present the architecture of CNNs and describe mathematically the basic operations employed by these networks. Much of the content presented in this section is inspired by a nice paper by Jianxin Wu [39]. The core operation of CNNs is the convolutional operation which acts usually on 2D objects. In our case, we will leverage the power of CNNs for the song identification task in order to map the 2D spectrogram representations corresponding to 1 sec audio fragments to a compact 1D representation that can be further indexed and processed efficiently, as opposed to the 2D spectrogram representation.

4.4.1 The Architecture in a Nutshell

As we have already mentioned, CNNs operate on images. In the simplest case, a CNN maps an image into a 1D feature vector. Mathematically, a CNN can be represented by a function $f : \mathbb{R}^{H\times W\times C} \to \mathbb{R}^d$, where H is the height, W is the width, and C the number of channels of the input feature. In the case of images, $C = 3$, where each channel corresponds to red, green, and blue colors. In the case of spectrograms, $C = 1$. Therefore, the input is a 3D matrix $x \in \mathbb{R}^{H\times W\times C}$. The input sequentially goes through a series of processing until it gets to its final vector form in $\mathbb{R}^d$. One processing step is usually called a layer, which could be a convolution layer, a pooling layer, a normalization layer or a fully connected layer. We will introduce these concepts in more detail in the next subsections . For the moment, let us give an abstract description of how the forward pass works in the case of CNNs. We can think of the structure of a CNN with the following equation

$$x^1 \to \omega^1 \to x^2 \to \cdots \to x^{L-1} \to \omega^{L-1} \to x^L \to \omega^L \to z \in \mathbb{R}^d. \tag{4.4.1}$$

Equation (4.4.1) illustrates how a CNN runs layer by layer in a forward pass. The input is x^1, which is usually a 3D matrix of size $H \times W \times C$. It goes through the processing in the first layer, which is denoted by ω^1. The processing operation usually translates to a matrix operation between the input and another matrix with parameters denoted by ω^1. The output of the first layer is x^2, which also acts as the input to the second processing layer. This processing proceeds until all layers in the CNN have been exhausted, which outputs x^L. One additional layer, however, is added for backward error propagation, i.e., the calculation of the gradients of the model's parameters. For instance, in an image classification task, a common strategy is to output x^L as a K-dimensional vector, whose i^{th} entry encodes the prediction (posterior probability of x^1 comes from the i^{th} class, which could be a certain object depicted in the image). As in the case of a classification task with MLPs, the standard way is to use the softmax function in (4.1.5). In detail, if we denote by $x_i^{L-1}, i \in \{1, \ldots, K-1\}$ the output of the $L-1$ layer; then

$$x_i^L = \text{softmax}\left(x_i^{L-1}\right) = \frac{\exp(x_i^{L-1})}{\sum_{j=1}^{K-1} \exp\left(x_j^{L-1}\right)}. \tag{4.4.2}$$

The calculation of the loss and the update of the parameters follow the same principles with MLPs; i.e., an optimization algorithm is chosen, like the ones presented in section 4.3, and the calculation of the gradients is done via the back propagation algorithm [40].

4.4.2 The Convolution Layer

Convolution is perhaps the most notable operation in CNNs. It usually acts on 2D or 3D objects, like RGB or grayscale images. Its goal is to extract relevant information from local positions on the image by gradually reducing the size of the image until it becomes 1D. You can think of the convolution layer as the linear layer in MLPs but instead of acting on 1D vectors it acts on objects of higher dimensions. To introduce the operation we first consider the simplest case where the input on the convolution layer is a 2D matrix $x \in \mathbb{R}^{H\times W\times 1}$. Then, one decides the size of convolution kernel, which is usually a square matrix $K \in \mathbb{R}^{K\times K\times 1}$, with $K < \min\{H, W\}$. In the case where the number of channels on the input are C, the dimensions of the kernel are $K\times K\times C$. Then, we overlap the convolution kernel K on top of the input image and we compute the product between the numbers at the same location in the kernel and the input. The output is a single number resulting by summing these products together. Figure 4.6 illustrates this operation with a concrete example where a 2×2 kernel acts on a 3×4 matrix.

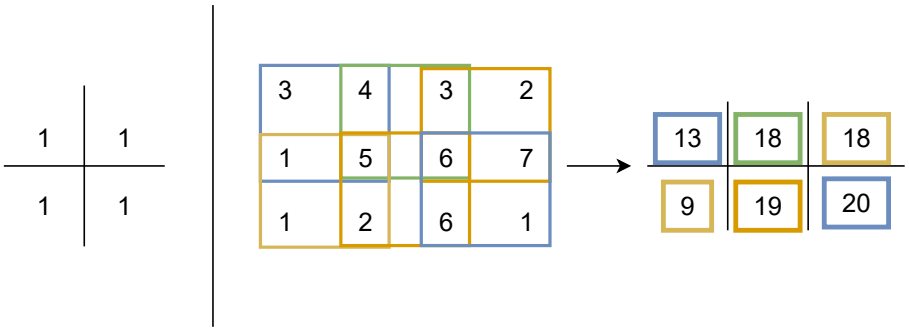

Figure 4.6. *The convolution operation in CNNs. A 2×2 kernel acts on a 3×4 2D matrix. The output is a 2×3 matrix.*

In Figure 4.6, if we overlap the kernel with the top left region in the input, the convolution result at that spatial location is: $1\times 3+1\times 4+1\times 1+1\times 5=13$. We then slide the kernel one pixel on the right and get the next convolution result as $1\times 4+1\times 3+1\times 5+1\times 6=18$. This procedure repeats until all possible locations for the kernel's starting position have been exhausted.

Now the case where the input is a 3D object of size $H\times W\times C$ the convolution operation is defined similarly. To this end, suppose that the input in the l-th layer is an order 3 matrix with size $H^l\times W^l\times C^l$. In this case, the convolution kernel is also an order 3 matrix with size $H\times W\times D^l$. When we overlap the kernel on top of the input tensor at the spatial location $(0,0,0)$, we compute the products of the corresponding elements in all the D^l channels and sum the HWD^l products to get the convolution result at this spatial location. Then, we move the kernel from left to right and from top to bottom to complete the convolution. In a convolution layer, multiple convolution kernels are usually used. Assuming D kernels are used and each of the kernel is of spatial span $H\times W$, then each of this kernels act on the input in the same manner to give the output result which is of size $(H^l-H+1)\times(W^l-W+1)\times D$. As we can see, the output of this operation has been reduced by $H-1$ pixels on height, and by $W-1$ pixels on width, respectively. To retain the spatial dimensions, a technique called *padding* is used. With padding if one inserts $\left\lfloor\frac{H-1}{2}\right\rfloor$ rows above the first row and $\left\lfloor\frac{H}{2}\right\rfloor$ rows below the last row, and pad $\left\lfloor\frac{W-1}{2}\right\rfloor$ columns to the left of the first

column and $\left\lfloor \frac{W}{2} \right\rfloor$ columns to the right of the last column of the input, the convolution output will be $H^l \times W^l \times D$ in size, which is the same as the input dimensions. The elements of the padded rows and columns are usually set to 0, but other values can be used.

Stride is another important concept in convolution. In Figure 4.6 we convolve the kernel input at every possible spatial location, which corresponds to the stride $s = 1$. If we use $s > 1$, then, every movement of the kernel skip $s - 1$ pixel locations (i.e., the convolution is performed once every s pixels both horizontally and vertically. Assuming that the parameters p_H, p_W control the amount padding on the vertical and horizontal side, respectively (i.e., $p_H = 1$ adds one column on the left and one on the right), and the parameters s_H, s_W the vertical and horizontal stride, respectively. For an input object of size $H^l \times \times W^l \times D^l$ and D kernels of size $H \times W \times D^l$, the output of the $l-$th layer is a 3D object with size $H^{l+1} \times W^{l+1} \times D$, where

$$H^{l+1} = \left\lfloor \frac{H^l + 2p_H - H}{s_H} + 1 \right\rfloor, \tag{4.4.3}$$

and

$$W^{l+1} = \left\lfloor \frac{W^l + 2p_W - W}{s_W} + 1 \right\rfloor. \tag{4.4.4}$$

4.4.3 Normalization Layers

A common practice when training deep learning models is to use normalization techniques along with the usual operations (linear operation in MLPs, convolution in CNNs). In this paragraph we introduce two of the most common normalization techniques; Batch Normalization [41] and Layer Normalization [42]. Normalization was introduced to address the challenges associated with training deep neural networks effectively. Some of the advantages of normalization are summarized below.

1. Adressing Internal Covariate Shift: As neural networks become deeper, the distribution of inputs to each layer (referred to as internal covariate shift) can change significantly during training, which can slow down training and make convergence difficult. Batch Normalization and Layer Normalization help stabilize and normalize the activations, mitigating this issue and leading to faster convergence.

2. Accelerating Training: By normalizing the activations in each layer, Batch Normalization and Layer Normalization can lead to faster training times. Normalized activations allow for more stable gradient propagation through the network, reducing the vanishing and exploding gradient problems. This results in quicker convergence and allows for the use of higher learning rates.

3. Reducing Sensitivity to Initialization: Traditional weight initialization methods might not work optimally for very deep networks. Batch Normalization and Layer Normalization help make networks less sensitive to the choice of initial weights, making training more robust and less dependent on careful initialization.

4. Regularization Effect: Batch Normalization and Layer Normalization introduce a form of regularization by adding noise to the activations during training. This has a similar effect to dropout [43], which can help prevent overfitting and improve the generalization ability of the network.

5. Improved Gradient Flow: Normalizing activations helps maintain a more consistent range of values throughout the network, which improves the flow of gradients during backpropagation. This, in turn, leads to more stable and efficient training.

6. Enabling Higher Learning Rates: The normalized activations reduce the likelihood of extreme values, which allows for the use of higher learning rates during optimization. Higher learning rates can help escape local minima and converge to better solutions.

As their name suggests, batch normalization involves a normalization operation along the batch dimension whereas layer normalization involves a normalization operation for each layer input, independently of the batch. In this paragraph we present the 2D versions of these normalization operations, the reader may refer to the original papers [41, 42] for the 1D case. Formally, suppose that we train a CNN with batch size M. Then, the input to the l-th layer will be a 4D object $x \in \mathbb{R}^{B\times H\times W\times C}$. Denote by $x(b,h,w,c)$ the entries of x, where $0 \leq b < B, 0 \leq h < H, 0 \leq w < W$, and $0 \leq c < C$. Then, in this case, batch normalization proceeds as follows: We first compute the mean and standard deviation across the batch dimension for each channel separately:

$$\mu_c = \frac{1}{BHW}\sum_{b=0}^{B-1}\sum_{h=0}^{H-1}\sum_{w=0}^{W-1} x(b,h,w,c), \tag{4.4.5}$$

and

$$\sigma_c^2 = \frac{1}{BHW}\sum_{b=0}^{B-1}\sum_{h=0}^{H-1}\sum_{w=0}^{W-1} (x(b,h,w,c)-\mu_c)^2\,, \tag{4.4.6}$$

$c \in \{0,\dots,C-1\}$. Then, we normalize each channel dimension by considering

$$y(b,h,w,c) = \frac{x(b,h,w,c)-\mu_c}{\sqrt{\sigma_c^2+\epsilon}}, \tag{4.4.7}$$

where $\epsilon > 0$ is a small constant used for numerical stability. A set of learnable parameters $(\gamma_c)_{c=0}^{C-1}$, $(\beta_c)_{c=0}^{C-1}$ are used to allow representations that are not centered around 0 with unit standard deviation. Therefore, the output of a batch normalization layer in the case of CNNs is a 4D object $\hat{y} \in \mathbb{R}^{B\times H\times W\times C}$, where

$$\hat{y}(b,h,w,c) = \gamma_c \cdot \frac{(x(b,h,w,c)-\mu_c)}{\sqrt{\sigma_c^2+\epsilon}} + \beta_c. \tag{4.4.8}$$

Now, in the case of CNNs, layer normalization calculates the same statistics (mean and standard deviation) for each feature map independently. In detail, for every fixed $0 \leq b < B-1$ we calculate μ_b, σ_b by

$$\mu_b = \frac{1}{HWC}\sum_{h=0}^{H-1}\sum_{w=0}^{W-1}\sum_{c=0}^{C-1} x(b,h,w,c), \tag{4.4.9}$$

$$\sigma_b^2 = \frac{1}{HWC}\sum_{h=0}^{H-1}\sum_{w=0}^{W-1}\sum_{c=0}^{C-1} (x(b,h,w,c)-\mu_b)^2\,. \tag{4.4.10}$$

As in the case of batch normalization, two learnable matrices $\gamma, \beta \in \mathbb{R} \in \mathbb{R}^{H\times W\times C}$ are considered.

The output of the layer normalization is a 4D object $\hat{y} \in \mathbb{R}^{B\times H\times W\times C}$ given by

$$\hat{y}(b,h,w,c) = \gamma(h,w,c) \cdot \frac{x(b,h,w,c) - \mu_b}{\sqrt{\sigma_b^2 + \epsilon}} + \beta(h,w,c). \tag{4.4.11}$$

4.4.4 The Pooling Layer

The purpose of pooling layers in CNNs is to reduce the dimensions of the input. In a typical scenario, the height and width of the input feature map are halved while the channel dimension remains the same. Two are the most common pooling operations; max pooling and average pooling. As in the case of the convolution layer, one decides the size of the kernel to be used in the pooling operations. These layers do not contain any learnable parameters and they are used to extract a compact representation from local regions in the input feature map by reducing its size at the same time. In detail, suppose that the input feature map is a 3D object $x \in \mathbb{R}^{H\times W\times C}$ and the kernel has size $K_H \times K_W \times C$. Then, both max and average pooling are performed by sliding the kernel on the input feature map in a non-overlapping fashion and either computing the maximum or average value, respectively, from the entries on the input feature map where the kernel is placed. Note that the aggregations operations are performed for each channel independently. Therefore, the output of a pooling layer is a 3D object of size $H' \times W' \times C$ where

$$H' = \left\lfloor \frac{H + 2p_H - K_H}{K_H} + 1 \right\rfloor,$$

and

$$W' = \left\lfloor \frac{W + 2p_W - K_W}{K_W} + 1 \right\rfloor,$$

where p_H, p_W corresponds to the number of columns/rows (padding) added on the sides of the feature map. To halve the dimensions of the input feature map one chooses a kernel with size $2 \times 2 \times C$ with no padding. Figure 4.7 demonstrates the max pooling operation with a kernel size of 2×2.

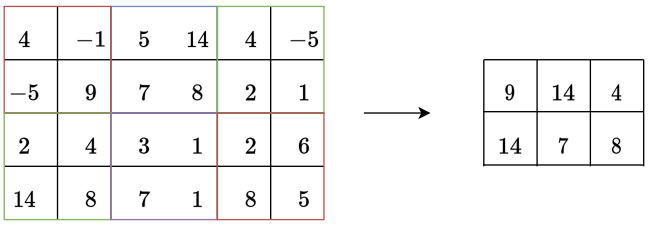

Figure 4.7. *Max pooling operation. A kernel of size* 2×2 *acts on a* 4×6 *matrix. The input is halved, resulting to an output of size* 2×3.

Chapter 5

Methodology: Deep Audio Fingerprinting

In this chapter, we adopt an alternative approach to the song identification problem presented in Section 3.5. We leverage the theory we have developed so far, spanning from the classic theory of the Fourier Transform to the pioneering ideas of deep learning to establish an efficient music recognition system that is entirely distinct from the one we introduced in 3.5. At an abstract level, both of these systems exhibit similar functionality; they both extract compact representations from the raw audio signals at the level of seconds, store these representations in a database, and during query time, employ non-exhaustive searching techniques to retrieve the most similar segment from the database relative to the query audio fragment. The key difference between these two systems is that in the deep audio approach, we develop a deep neural network and train it in order to learn by itself the latent representations of the audio fragments. This approach relieve us from the burden of crafting sophisticated ideas to extract relevant information from the raw audio signals, instead, we let the model to discover this representation. These ideas were first introduced by Google's research team in [13] where they launched an app called Now Playing, which is available on their Google Pixel smartphone devices. Later on, these ideas were refined in [30] where the authors used an alternative technique based on contrastive learning to train the deep neural network.

In the upcoming paragraphs we explain how one can set up a music recognition system based on these ideas. Furthermore, we introduce the contrastive learning framework and train a CNN neural network with the contrastive loss in order to learn compact feature representations from the raw audio signals. We create a database with the learned representations and use techniques such as product quantization to compress the feature representations. As in the case of dejavu, on query time, non-exhaustive searching algorithms are employed for fast and efficient retrieval. At the end, we compare these two music recognition systems with respect to the accuracy, time retrieval, the size of the database, and in view of scalability.

5.1 An Overview of the Music Recognition System

In this section we provide a high level overview of the music recognition system based on the deep audio fingerprinting approach. This section will serve as the building block for the upcoming sections where we delve into the design details of the system. The whole process of designing

such a system can be divided into three parts: (a) The development and training of the model, (b) the extraction of the feature representations from the raw audio signals, and c) the creation of the database. In Section 5.2 we explain in detail how to approach (a) and (b), and in Section 5.3 we create the database consisting of the latent representations and provide an efficient and fast retrieval pipeline.

On an abstract level, the model is a simple CNN that maps 2D mel-spectrograms 3.3.3, extracted from the raw audio signals to 1D vectors. Mathematically, this is represented by a function $f : \mathbb{R}^{F\times T} \to \mathbb{R}^D$, where F corresponds to the number of mel-frequency bins, and T to the number of time frames on the mel-spectrogram representation. As we will see, in our case we pick $F = 256$, and $T = 32$, resulting from the use of the STFT on audio fragments corresponding to 1 sec of total duration. Therefore, the time resolution of our seconds is of the order of 1 sec. In addition, a normalization layer $g : \mathbb{R}^D \to \mathcal{S}^{D-1}$ is added on top of f that maps the vectors in $\mathbb{R}^D$ to the unit sphere

$$\mathcal{S}^{D-1} = \left\{ z \in \mathbb{R}^D : \|z\|_2 = 1 \right\}.$$

Therefore, the final model is $g \circ f : \mathbb{R}^{F\times T} \to \mathcal{S}^{D-1}$. Figure 5.1 illustrates the sequence of transformations from the raw audio signal to the final feature vector. The model is trained in such a way that the feature vectors in $\mathbb{R}^D$ exhibit robustness to high distortions. This means that the feature representation of a distorted signal is close to the original clean audio signal with respect to cosine similarity. This is achieved by training the model using the contrastive learning framework presented in 5.2.2. Once the model has been trained it is used to extract the feature representations from the songs that will be added to the database. Suppose that the database consists of N songs in total. Then, for every song $i \in \{1, \ldots, N\}$ we extract a feature representation $x_{i,j} \in \mathbb{R}^D$ for each second with an overlap corresponding to 500ms. Therefore, if the i^{th} song has D_i seconds duration, we extract $d_i = 2D_i - 1$ feature vectors in total. Hence, the database $\mathcal{B}$ consists of the feature representations

$$\mathcal{B} = \bigcup_{i=1}^{N} \{x_{i1}, \ldots, x_{id_i}\}.$$

Also, a second structure tracks the indices where each song's feature vectors start and end on the database. This structure is utilized on retrieval time to return all the metadata related to the matched song. In Figure 5.2 you can see a visual representation of the database after the feature extraction.

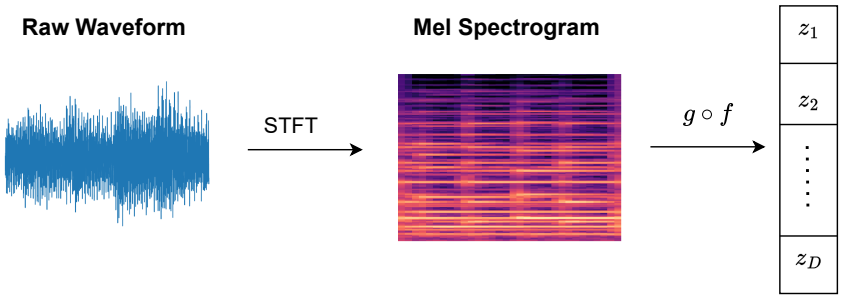

Figure 5.1. *The sequence of processing steps from the raw audio signal corresponding to 1 sec duration to the final feature vector in* $\mathcal{S}^{D-1}$.

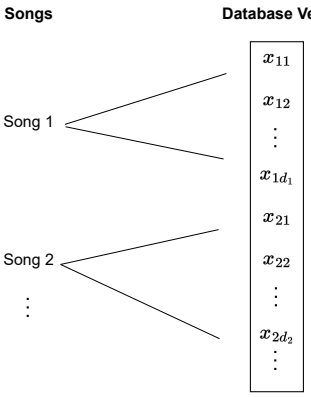

Figure 5.2. *A representation of the database consisting of the feature representations from the audio segments of 1 second. A second structure (on the left) keeps track of first and last index of each song's representations.*

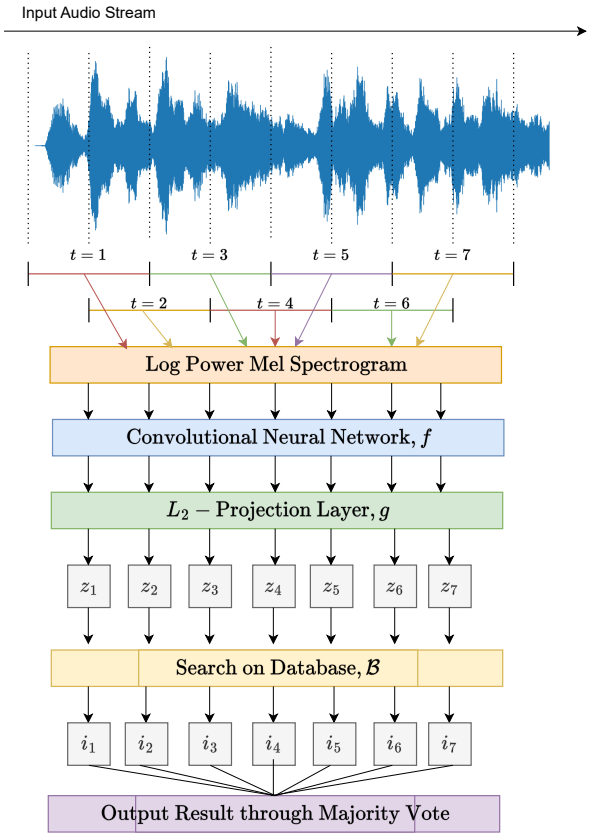

Figure 5.3. *A high level overview of the deep audio fingerprinting system during the retrieval process.*

Once we have established the database consisting of the feature representations and all the relevant metadata for the songs we can use the vast theory of searching algorithms to obtain efficient and fast retrieval capabilities. A naive approach for the retrieval process would work as follows: Suppose we capture from the microphone a query audio signal of D seconds in duration. Then, we use the STFT to extract the corresponding mel spectrogram representations $\{s_1, \ldots, s_{2D-1}\} \subseteq \mathbb{R}^{F \times T}$. We use the trained model $g \circ f$ to obtain the query representations $\{y_1, \ldots, y_{2D-1}\} \subseteq \mathbb{R}^D$. In the naive case, we can run an exhaustive search for each of the query segments y_i, $1 \leq i \leq 2D-1$ and

extract the best match according to the highest cosine similarity value. Therefore, for each query segment y_i we obtain the best matching index on the database by

$$j_i = \arg\max_{x \in \mathcal{B}} \langle y_i, x \rangle. \tag{5.1.1}$$

Utilizing the structural correspondence between the song names and the database indexes, we convert the set of best-matching indexes $\{j_1, \ldots, j_{2D-1}\}$ to the corresponding song names. The final output of the system is the song with the most occurrences. In Figure 5.3, you can see a high-level overview of the retrieval process. While this approach is simple and effective for small databases, it becomes prohibitive for databases containing thousands or even millions of songs. This is due to the fact that the exhaustive search slows down the retrieval process. We encountered a similar problem in paragraph 3.5.1, where we adopted alternative searching techniques. In this case, we will leverage the vast theory known as *Approximate Nearest Neighbor (ANN) Search*. We introduce *Product Quantization* [44], a technique that compresses the feature vectors significantly, reducing the overall memory footprint of the database. The non-exhaustive search through the inverted file index allows us to achieve database look-ups in the order of milliseconds.

5.2 Training the Model

5.2.1 The Architecture

In this paragraph we describe the architecture of the CNN that we will train for the purposes of the music recognition system. The operations handled by the model are similar to the ones presented in 4.4 with slight modifications; i.e., we use separable convolutions instead of the full convolutional operation to reduce the number of model's parameters and gain some inference speed. The architecture is essentially the same as in [14, 13] where it has been shown to perform relatively well on the song identification task.

Following the notation in [14] we denote by $C^{o \leftarrow i}_{k*s}$ convolution operation with input channel i, output channel o, kernel size $1 \times k$, and stride $1 \times s$. The k' and s' denote rotation as $k \times 1$ and $s \times 1$. In any of the operations $C^{o \leftarrow i}_{k*s}$ or $C^{o \leftarrow i}_{k'*s'}$ we assume a padding equal to 0×1, 1×0 is used, respectively. Therefore, $C^{o \leftarrow i}_{k*s}$ acts on 3D objects of shape $H \times W \times i$ by mapping them to 3D objects of shape $H' \times W' \times o$, where $H' = H$ and

$$W' = \left\lfloor \frac{W + 2 - k}{s} + 1 \right\rfloor. \tag{5.2.1}$$

Similarly, $C^{o \leftarrow i}_{k'*s'}$ acts on 3D objects of shape $H \times W \times i$ by mapping them to 3D objects of shape $H' \times W' \times o$, where $W' = W$ and

$$H' = \left\lfloor \frac{H + 2 - k}{s} + 1 \right\rfloor. \tag{5.2.2}$$

Now, if we denote by ReLU the layer that applies the ReLU activation elementwise on the input, and by LN the layer normalization as presented in 4.4.3, the separable convolution is the mapping given by

$$SC^{o \leftarrow i}_{k*s}(\cdot) = \text{ReLU} \circ \text{LN} \circ C^{o \leftarrow i}_{k'*s'} \circ \text{ReLU} \circ \text{LN} \circ C^{o \leftarrow i}_{k*s}(\cdot) \tag{5.2.3}$$

With these notations, we can express the encoder part f of the model as

$$f(\cdot) = SC_{3*2}^{h\leftarrow h} \circ SC_{3*2}^{h\leftarrow 4d} \circ SC_{3*2}^{4d\leftarrow 4d} \circ SC_{3*2}^{4d\leftarrow 2d} \circ SC_{3*2}^{2d\leftarrow 2d} \circ SC_{3*2}^{2d\leftarrow d} \circ SC_{3*2}^{d\leftarrow d} \circ SC_{3*2}^{d\leftarrow 1}(\cdot), \quad (5.2.4)$$

where h is the encoder's embedding dimension, and d the final output dimension. Hence, the encoder part f of the final model $g \circ f$ takes as input a log power mel spectrogram $\mathcal{S} \in \mathbb{R}^{F\times T}$ and outputs an h-dimension vector. In practice, we choose h to be a multiple of d. Now, the projection head g is a mapping $g : \mathbb{R}^h \to \mathcal{S}^{d-1}$. To map the h-dimensional vector onto the unit sphere $\mathcal{S}^{d-1}$ we first split the vector into $m = h/d$ subsequent subvectors $v_1, \ldots, v_m \in \mathbb{R}^d$. Then, these vectors are inserted as columns to form a 2D matrix of size $1 \times 128 \times m$. We denote this transformation by $\text{Split}^{h/d}$. With this notation, g can be expressed as

$$g(\cdot) = L_2 \circ C_{1*1}^{1\leftarrow 32} \circ \text{ELU} \circ C_{1*1}^{32\leftarrow m} \circ \text{Split}^{h/d}(\cdot), \quad (5.2.5)$$

where ELU is the ELU activation function 4.1.2, and L_2 denotes the L_2-normalization given by $z \mapsto z/\|z\|_2$, for $z \in \mathbb{R}^d$. The input spectrogram is derived by 1 sec audio segments. We use a samping rate equal to $F_s = 8000$ Hz. The STFT window N is equal to 1024 with a hop length H corresponding to 256 samples. These parameters yield a spectrogram $\mathcal{S}$ of size 512×32 (frequency bins $\times$ time stamps). 256 mel frequency bins have been adopted resulting to the log power mel spectrogram of size 256×32. The embedding dimension for the encoder is equal to $h = 1048$, and the final output dimension of g is 128. Table 5.1 summarizes the forward pass through the network $g \circ f$.

Table 5.1. *The Forward Pass through the Network $g \circ f$.*

Operation	Input	Output
$SC_{3*2}^{d\leftarrow 1}$	$256 \times 32 \times 1$	$128 \times 16 \times 128$
$SC_{3*2}^{d\leftarrow d}$	$128 \times 16 \times 128$	$64 \times 8 \times 128$
$SC_{3*2}^{2d\leftarrow d}$	$64 \times 8 \times 128$	$32 \times 4 \times 256$
$SC_{3*2}^{2d\leftarrow 2d}$	$32 \times 4 \times 256$	$16 \times 2 \times 256$
$SC_{3*2}^{4d\leftarrow 2d}$	$16 \times 2 \times 256$	$8 \times 1 \times 512$
$SC_{3*2}^{4d\leftarrow 4d}$	$8 \times 1 \times 512$	$4 \times 1 \times 1024$
$SC_{3*2}^{4d\leftarrow 4d}$	$4 \times 1 \times 1024$	$2 \times 1 \times 1024$
$SC_{3*2}^{h\leftarrow 4d}$	$2 \times 1 \times 1024$	$1 \times 1 \times 1024$
$SC_{3*2}^{h\leftarrow h}$	$2 \times 1 \times 1024$	$1 \times 1 \times 1024$
$\text{Split}^{h/d}$	$1 \times 1 \times 1024$	$1 \times 128 \times 8$
$C_{1*1}^{32\leftarrow m}$	$1 \times 128 \times 8$	$1 \times 128 \times 32$
ELU	$1 \times 128 \times 32$	$1 \times 128 \times 32$
$C_{1*1}^{1\leftarrow 32}$	$1 \times 128 \times 32$	$1 \times 128 \times 1$

5.2.2 The Contrastive Learning Framework

In this paragraph, we introduce the concept of *Contrastive Learning*, which serves as the framework for training the CNN presented in the previous paragraph. One of the most well-known papers about contrastive loss is [30], in which the authors introduce the concept of contrastive

learning – a promising self-supervised technique that facilitates the learning of discriminative visual representations. By leveraging this framework they manage to achieve state-of-the-art performance in image classification. One of the most significant advantages of the contrastive learning framework is that it does not require labeled data; rather, it only necessitates raw data inputs to develop a model capable of mapping high-dimensional data points (e.g., images or spectrograms) to low-dimensional and discriminative feature vectors in the latent space. The main idea is simple: Assuming a training dataset, whether it be images or audio fragments, we apply a series of augmentations to the clean samples and train the model in such a way that it maps the distorted samples as closely as possible to the clean samples in the latent space. This is achieved via the *contrastive loss*. The crucial factor for learning effective representations is the choice of a series of data augmentations. In our case, we aim for our CNN to model all the audio distortions that may arise in a realistic scenario, where audio fragments are captured through a microphone in reverberant and noisy environments. To achieve this, we introduce a novel augmentation pipeline that effectively models all kinds of distortions that can arise in various environments where music is playing in the background. Compared to the existing literature [13, 14], our approach improves the overall performance of the system by a huge margin when music is captured through the microphone, making it possible to develop promising applications. Therefore, we adopt the following augmentations:

Background Noise: We distort the clean audio fragment of 1 second with background noise with varying SNR (see 3.4.1) in dB. In particular, with probability $p = 1/2$ we add noise with an SNR chosen uniformly from $[0, 5]$ dB. With $p = 0.3$ with an SNR $\sim U([5, 10])$, and with $p = 0.2$ with an SNR $\sim U([10, 15])$. The probability of applying the background noise transformation is 0.8.

Impulse Response: To simulate the effect of diverse spatial and microphone environments, microphone and room impulse responses are sequentially applied using the convolution operation as in 3.4.2. Impulse response is added with probability equal to 1.

Time offset modulation: As we have already mentioned, the fingerprints (representations) are extracted for each 1 sec audio segment duration with a hop size of 500ms. However, in a realistic scenario the captured audio stream may be shifted. To this end, we shift the audio fragment with an offset of up to 200 ms with probability $p = 0.3$.

Low/High Pass Filters with Decaying Energy: Low and high pass filters as introduced in 3.4.3 are utilized to model the degradation of certain frequencies when the audio is captured through the microphone. We apply both low and high pass filters with decaying energy with probability $p = 0.4$. In the low filter case, we randomly choose a threshold $v_c \in [2000, 3000]$ Hz and we decay the frequencies of each octave above that threshold with increasing fixed decibel numbers $a \in \{6k : k = 2, \dots 6\}$. Similarly, in the high pass filter, we randomly choose a threshold $v_c \in [500, 1000]$ Hz and we decay the frequencies below that threshold in the same manner as in case of the low pass filter. As we will see, these kind of transformations increased by a huge margin the performance of the overall system on the microphone test.

We denote the above transformations by $\mathcal{M}(\cdot)$. By this notation, $\mathcal{M}(x)$ corresponds to the output of the clean audio sample x when distorted with the aforementioned transformations. The idea of the contrastive loss is to bring as closely as possible the distorted audio sample $\mathcal{M}(x)$ to the

original sample x in the latent space. This is achieved by maximizing the inner product

$$\langle g \circ f\left(\mathcal{M}(x)\right), g \circ f(x) \rangle.$$

More formally, the contrastive loss is defined as follows: Let a sample $\{x_1, \ldots, x_N\}$ of clean audio spectrograms corresponding to 1 sec and let $\{x_{N+1}, \ldots, x_{2N}\}$ their corresponding transformations, i.e., $x_{i+N} = \mathcal{M}(x_i), i = 1, \ldots, N$. Then, the loss that measures how close is x_i to x_j on the embedding space $\mathcal{S}^{d-1}$ is given by

$$\ell(i,j) = -\log \frac{e^{\langle g \circ f(x_i), g \circ f(x_j) \rangle / T}}{\sum_{k=1}^{2N} \mathbb{1}_{k \neq i} \cdot e^{\langle g \circ f(x_i), g \circ f(x_k) \rangle / T}}, \tag{5.2.6}$$

where $T > 0$ is a hyperparameter called the temperature. Observe that ℓ is not symmetric, i.e., $\ell(i,j) \neq \ell(j,i)$. The contrastive loss is then given by

$$\mathcal{L} = \frac{1}{2N} \sum_{k=1}^{N} [\ell(k, N+k) + \ell(N+k, k)]. \tag{5.2.7}$$

Algorithm 5.1 provides a high level overview of the training pipeline with the contrastive loss. We use the custom dataset as described in section 6.1 comprising of 26016 audio clips in total. 17 songs are hand-picked to serve as the test songs to evaluate the performance of the deep audio fingerprinting approach. The results are presented in paragraph 6.3.1. We train the model according to algorithm 5.1 for a maximum of 120 epochs. We use a large batch size equal to $N = 500$ since the contrastive loss benefits from large batch sizes. We use LAMB(4.3.5) as an optimizer since it performs better for large batch size values. The learning rate is equal to $10^{-3} \cdot N/640$ and decays with a cosine function reaching a minimum of 10^{-7} at epoch 120. The temperature parameter is $T = 0.05$. Figure 5.4 shows the training and validation curves for each training epoch. The minimum validation loss occurred at epoch 68 reaching a value of 0.2670. Early stopping was used with a patience corresponding to 15 epochs. The next step is to use the best model to extract the representations (fingerprints) for the 26016 songs. We extract the audio fingerprints corresponding to 1 sec duration with a hop length equal to 500 ms. This results to a total of 1973384 128-dimensional fingerprints.

Algorithm 5.1: *Training of the CNN with Contrastive Loss*

Input: Even number of batch size N, temperature $T > 0$, transformations $\mathcal{M}(\cdot)$.
for each sampled mini-batch $\{x_k\}_{k=1}^{N/2}$.
  for all $k \in \{1, \ldots, N/2\}$ **do**
    compute the transformed samples $x_{N/2+k} = \mathcal{M}(x_k)$.
    $z_k \leftarrow g \circ f(x_k)$.
    $z_{N/2+k} \leftarrow g \circ f(x_{N/2+k})$.
  Create the set $\{z_1, \ldots, z_{N/2}, z_{N/2+1}, \ldots, z_N\}$.
  for all $k \in \{1, \ldots, N/2\}$
    Compute $\ell(k, N/2+k), \ell(N/2+k, k)$.
  Compute $\mathcal{L} = \frac{1}{N} \sum_{k=1}^{N/2} [\ell(k, N/2+k) + \ell(N/2+k, k)]$.
  Update the parameters of $g \circ f$ to minimize $\mathcal{L}$.

One way to verify that the model has learned useful representations is to plot the fingerprints for two different songs. In Figure 5.5 you can see the fingerprints of two different songs: 1) "Sorry" by Madonna, and 2) "Queen of the Night Aria" by Wolfgang Amadeus Mozart projected in the 2D plane using PCA. A 5-sec audio fragment from the Queen of the Night Aria has been distorted with background noise to produce 9 query fingerprints.

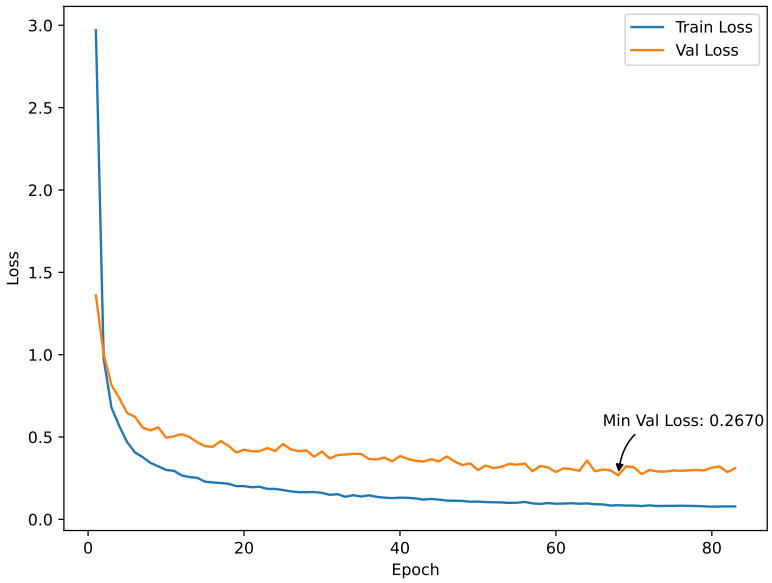

Figure 5.4. *Train and validation loss curves.*

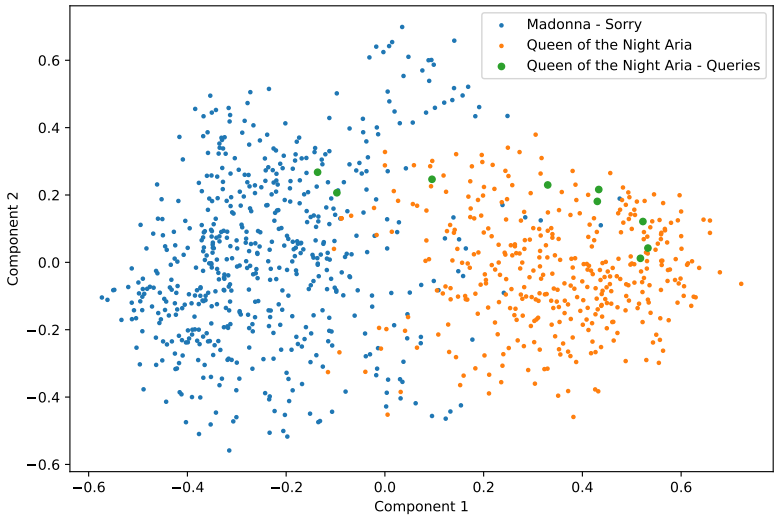

Figure 5.5. *Fingerprints of the songs "Sorry" (blue) by Madonna and "Queen of the Night Aria" (orange) by Wolfgang Amadeus Mozart project in 2D with PCA. The green dots correspond to fingerprints of a 5-sec audio fragment of the Queen of the Night distorted with background noise with 0 SNR in dB.*

5.3 Creating the Database

In this section, we describe how to set up the database of fingerprints to allow for fast and efficient retrievals. As stated in the previous section, 26,016 songs yield a total of 1,973,384 128-dimensional fingerprints. Each entry of the feature vector is stored as a 32-bit floating point, resulting in a total of 1.1 GB. However, approximately 25,000 of the songs correspond to 30-second audio fragments. In a realistic scenario, assuming each song lasts about 3 minutes on average (i.e., 180 seconds), the database will contain approximately 9 million fingerprints, with a total size of 5 GB. Considering scalability, these numbers are prohibitive for applications that will run on small devices requiring fast retrievals. In the next paragraph, we introduce the concept of *product quantization*, a technique which significantly reduces the memory footprint of the database. To address the challenge of searching in large databases with millions of fingerprints, we employ a non-exhaustive search technique based on inverted file indexing. The synergy between these two techniques enables us to establish a highly efficient recognition system capable of operating on small devices.

5.3.1 Indexing the Database

Computing Euclidean distances, as well as other metric quantities, between high-dimensional vectors is a fundamental requirement in many applications. Just like in our case, numerous recommendation systems need to search for similar items represented as vectors in the database. This paradigm it is known as nearest neighbor (NN) search. Nearest neighbor search is inherently expensive due to the *curse of dimensionality* [45, 46]. Our goal in NN search in $\mathbb{R}^d$ is to find the element $\mathrm{NN}(x)$, in a finite set $\mathcal{B} \subseteq \mathbb{R}^d$ of n vectors, minimizing the distance to the query vector $x \in \mathbb{R}^d$:

$$\mathrm{NN}(x) = \arg\min_{y \in \mathcal{B}} d(x, y). \tag{5.3.1}$$

An obvious way to address this problem is by employing an exhaustive search on the database, i.e., search one by one the items $y \in \mathcal{B}$. However, this approach is not effective when the cardinal number of $\mathcal{B}$ is large. A lot of algorithms [47, 48, 49] have been invented to address this issue by performing approximate nearest neighbor (ANN) search. On the other hand, *vector quantization* methods aim to reduce the cardinality of the representation space and, thus, reduce the memory footprint of the fingerprints to be stored in the database. A survey on quantization methods can be found in [50]. In this paragraph, we focus on the methods presented in [44], where product quantization is combined with an inverted file indexing technique to simultaneously reduce retrieval time and the total memory footprint of the database.

Product Quantization

Quantization methods reduce the cardinality of the representations through *quantizers*. Formally, a quantizer is a function q mapping a d-dimensional vector $x \in \mathbb{R}^d$ to a vector

$$q(x) \in C = \{c_i : i \in \mathcal{I}\},$$

where $\mathcal{I}$ is a finite index set, e.g., $\mathcal{I} = \{0, \ldots, k-1\}$. The reproduction values c_i are called *centroids*. The set of reproduction values C is the *codebook* of size k. The set V_i of vectors mapped to a given index i is referred to as a (Voronoi) *cell*, and defined as

$$V_i = \{x \in \mathbb{R}^d : q(x) = c_i\} \tag{5.3.2}$$

The k cells of a quantizers form a partition of $\mathbb{R}^d$, i.e. $\mathbb{R}^d = \bigcup_{i=1}^{k} V_i$, and $V_i \cap V_j = \emptyset$ for $i \neq j$. In the context of product quantization, the approach differs: instead of quantizing the entire vector $x \in \mathbb{R}^d$, we partition the vector into m subvectors u_j of dimension $d^* = d/m$, where d is a multiple of m, and apply separate quantization to each resulting subvector. Therefore, a given vector x is mapped as follows:

$$\underbrace{x_1, \ldots, x_{d^*}}_{u_1(x)}, \ldots, \underbrace{x_{d-d^*+1}, \ldots, x_d}_{u_m(x)} \mapsto q_1(u_1(x)), \ldots, q_m(u_m(x)), \tag{5.3.3}$$

where q_j is a low-complexity quantizer associated with the j-th subvector. With the subquantizer q_j we associate the index $\mathcal{I}_j$, the codebook C_j and the corresponding reproduction values (centroids) c_{ji}. A reproduction value of the product quantizer is identified by an element of the product index set $\mathcal{I} = \mathcal{I}_1 \times \cdots \times \mathcal{I}_m$. The codebook is therefore defined as the cartesian product

$$C = C_1 \times \cdots \times C_m,$$

and a centroid of this set is the concatenation of centroids of the m subquantizers. From now on, we assume that all subquantizers have the same finite number k^* of reproduction values. In that case, the total number of centroids is given by $k = (k^*)^m$. Then, each subvector u_j is associated with a unique centroid c_{ji}. The key part of this correspondence is that we only need to store each centroid c_{ji} once, and for each subjector u_j we only need to store the index $i \in \mathcal{I}_j$ associated with the centroid c_{ji}. Figure 5.6 illustrates this idea with a simple example where a 16-dimensional vector is mapped to a 4-dimensional vector with integer values. The centroids c_{ji} are learned at a preprocessing step via the K-means algorithm. In this way, the complexity of learning the quantizer is m times the complexity of performing K-means clustering with k^* centroids of dimension d^*.

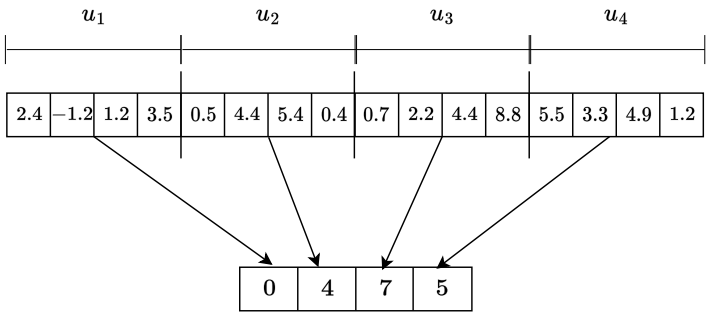

Figure 5.6. *Illustration of product quantization. A 16-dimensional vector is mapped to 4-dimensional integer valued vector, where each entry corresponds to the index of the associated centroid.*

To calculate the space savings achieved by this representation, we first note that since each index set $\mathcal{I}_j$ contains at most k^* integers, its space requirement is $O(\log_2 k^*)$. With m index sets I_j, the complexity of representing a single data item is $m \cdot \log_2 k^*$, referred to as the *code length* of the quantizer. Additionally, we need to store the mk^* centroids c_{ji} as 32 bit floating point numbers. Therefore, the memory footpring for a database of size n is $32mk^*d + nm \log_2 k^*$ bits. In practice, k^* is often chosen to be a power of 2, with $k^* = 256 = 2^8$ being a reasonable choice. Table 5.2 displays the space requirements of product quantization for various values of m and the number of songs in the database. The trade-off lies between the number of subquantizers m and the system's performance. Smaller values for m correspond to higher compression at the cost of accuracy in retrieval time.

Table 5.2. *Space requirements in Product Quantization ($d = 128$)*

Songs	**Fingerprints (N)**	m	$n\,(k^* = 2^n)$	**Database Size**
2.7 K	~ 1 M	64	8	97 MB
2.7 K	~ 1 M	32	8	67 MB
5.5 K	~ 2 M	64	8	136 MB
27 K	~ 10 M	64	8	648 MB
140 K	~ 50 M	64	8	3.2 GB
140 K	~ 50M	8	8	400 MB

Now, on query time, the distance calculation between a query vector $x \in \mathbb{R}^d$ and the quantized database vectors $q(y) = (q_1(u_1), \ldots, q_m(u_m))$ proceeds as follows: Denote by $d(x, y)$ the distance between x, y in $\mathbb{R}^d$, where d is the Euclidean distance. Then, $d(x, y)$ is approximated by the distance $\widetilde{d}(x, y) = d(x, q(y))$, which is computed using the decomposition

$$\widetilde{d}(x, q(y)) = d(x, q(y)) = \sqrt{\sum_{j=1}^{m} d\left(u_j(x), q_j(u_j(y))\right)^2}, \tag{5.3.4}$$

where the squared distances $d(u_j(x), c_{ji})^2$, for $j = 1, \ldots, m, i = 1, \ldots, k^*$, are computed prior to the search. However, searching with product quantization still examines all elements of the database, which can lead to a time overhead during the retrieval process. Below, we present the inverted file index that addresses this issue.

Inverted File Index

Inverted file index introduces another quantizer q_c, referred to as *coarse quantizer*, that associates the database items to a predetermined number of centroids c_j, for $j = 1, \ldots, k'$. The number of centroids associated with q_c typically ranges from $k' = 1,000$ to $k' = 1,000,000$. Then, for each item $y \in \mathbb{R}^d$ in the database, the product quantizer q_p is used to encode the residual vector

$$r(y) = q - q_c(y), \tag{5.3.5}$$

corresponding to the offset in the Voronoi cell. In this representation the vector $y \in \mathbb{R}^d$ is approximated by

$$\hat{y} = q_c(y) + q_p(y - q_c(y)). \tag{5.3.6}$$

In this case, the estimator of $d(x, y)$, where x is the query and y the database vector, is computed as the distance $\hat{d}(x, y)$ between x and $\hat{y}$:

$$\hat{d}(x, y) = d\left(x - q_c(y), q_p\left(y - q_c(y)\right)\right). \tag{5.3.7}$$

Denoting by q_{p_j} the j-th subquantizer, 5.3.7 can be calculated via

$$\hat{d}(x, y)^2 = \sum_{j=1}^{m} d\left(u_j(x - q_c(y)), q_{p_j}\left(u_j(y - q_c(y))\right)\right)^2. \tag{5.3.8}$$

Similar to the case of the product quantizer strategy, for each subquantizer q_{p_j} the distances between the partial residual vector $u_j(x - q_c(y))$ and all centroids c_{ji} of q_{p_j} are preliminary computed and stored. The product quantizer is learned on a set of residual vectors collected from a learning set. In detail, the construction of the inverted file index system along with the product quantization on residuals is the following: Suppose we have a finite set of database items $\mathcal{B} = \{y_1, \ldots, y_N\}$, a coarse quantizer q_c with k' centroids, and a product quantizer q_p. We first cluster each database vector according to q_c using K-means. This clustering yields k' inverted lists $\mathcal{L}_1, \ldots, \mathcal{L}_{k'}$, where each list $\mathcal{L}_i$ stores the indexes of the database items associated with the centroid c_i. Formally, $\mathcal{L}_i$ can be expressed as

$$\mathcal{L}_i = \{1 \leq i \leq N : \ q_c(y_i) = c_i\}. \tag{5.3.9}$$

Then, a single product quantizer q_p is produced by the residuals $y_i - q_c(y_i)$, $1 \leq i \leq N$. As opposed to the product quantization (without the inverted file index) an additional storage of $32dk'$ bits is needed to store the k' centroids produced by the coarse quantizer q_c. On query time, the vector x is assigned to the closest centroid c_i according to the encoding $q_c(x)$, and only the inverted list $\mathcal{L}_i$ corresponding to $q_c(x)$ is scanned. However, x and its nearest neighbor are often not quantized to the same centroid, but to nearby ones. To address this problem, a technique called *probing* is used. The query x is assigned to ω indexes instead of only one, which correspond to the ω nearest neighbors of x in the codebook of q_c. Algorithms 5.2, 5.3 (as presented in [44]) describe the indexing, and searching process in the inverted file system with product quantization.

Algorithm 5.2: *Indexing a vector y*

Input: Database items $\mathcal{B}$, trained coarse quantizer q_c, trained product quantizer q_p

1. quantize y to $q_c(y)$
2. compute residual $r(y) = y - q_c(y)$
3. quantize $r(y)$ to $q_p(r(y))$, which, for the product quantizer, amounts to assigning $u_j(y)$ to $q_j(u_j(y))$, for $j = 1, \ldots m$.
4. add a new entry to the inverted list corresponding to $q_c(y)$. It contains the vector identifier and the binary code (the product's quantizer's indexes).

ALGORITHM 5.3: *Searching on the database*

Input: Database items $\mathcal{B}$, trained coarse quantizer q_c, trained product quantizer q_p

Input: query vector x

1. quantize x to its ω nearest neighbors in the codebook q_c. Denote by $r(x)$ the residuals associated with these ω assignments. The following steps are applied to all ω assignments.

2. compute the squared distance $d(u_j(r(x)), c_{ji})^2$ for each subquantizer j and each of its centroids c_{ji}

3. compute the squared distance between $r(x)$ and all the indexed vectors of the inverted list. Using the subvector-to-centroid distances computed in the previous step, this consists in summing up m looked-up values.

4. select the K nearest neighbors of x based on the estimated distances. This is implemented efficiently by maintaining a Maxheap structure of fixed capacity, that stores the K smallest values seen so far. After each distance calculation, the point identifier is added to the structure only if its distance is below the largest distance in the Maxheap.

Assuming that the database items are uniformly distributed to the centroids produced by the coarse quantizer, the time complexity is reduced by a factor of k' in case the probing is equal to 1. In case where ω lists have to be scanned, about $n \times \omega/k'$ entries have to be parsed, which is again significantly lower compared to the exhaustive approach.

Chapter 6

Datasets and Experiments

In this chapter we present the experiments conducted for the purposes of the thesis. We first begin by introducing the dataset that we used for setting up the database for both approaches, and to train the deep neural network as well. In Section 6.2 we introduce two experiments for evaluation the perfomance of both music recognition system. In Section 6.3 we implement by using an open source library a music recognition system based on the ideas of Shazam's algorithm as described in 3.5. We evaluate the performance of this system in two different scenarios. Next, in paragraph 6.4.1 we present the implementation details for the deep audio fingerprinting approach and the results in the off-line evaluation. We conclude this chapter in paragraph 6.4.2 where we compare the performance of these two system in a realistic scenario where the audio fragments are captured through the microphone of a portable device. Furthermore, we evaluate the impact of the augmentations on training the CNN. The results show that the music recognition system based on the ideas of deep learning exhibits superior performance when compared to the classic method.

6.1 Dataset

For the purposes of the experiments we constructed a custom dataset consisting of 26,016 songs. 24,985 songs were obtained from the well-known FMA dataset [51]. In particular, these 24,985 audio clips correspond to 30-sec audio fragments from various music genres. Furthermore, 1,031 songs (full-length) were downloaded from YouTube with yt-dlp API[1] to resemble as closely as possible a realistic scenario where the songs cannot be easily distinguished according to their genre. Two collections of 17 songs, and 15 songs, out of the 1,031 were hand-picked according to the author's personal preferences to serve as the test songs to measure, and compare the performance of the music recognition systems in the off-line evaluation, and in the microphone evaluation protocols, respectively (see 6.2). In Tables 6.1, 6.2 you can see these songs and some general information such as the artist's name, duration, and genre. All songs where downsampled to 8KHz in a mono format with FFmpeg[2] resulting to 16GB of total size. In addition, 100 audio clips with various ambient noises like cars, train stations, and people talking in various places (bars, clubs, cafeterias, etc.) were collected to simulate the background noise that these music recognition systems may encounter in a real scenario. The total duration of the background noises

[1] https://github.com/yt-dlp/yt-dlp
[2] https://ffmpeg.org/documentation.html

is 20 hours, 42 minutes, and 8 seconds. Finally, to train the deep neural network to exhibit robustness against the reverbaration effects that may arise in a realistic scenario, 292 audio clips of impulse responses were hand picked according to the quality to simulate these distortions[3].

6.2 Evaluation Protocols

To evaluate and compare the performance of the two music recognition systems we introduce two different evaluation protocols:

(i) An off-line evaluation protocol where the 17 test songs are distorted in an off-line manner with background noise of fixed SNR. By an off-line manner we mean that the clean audio signals are distorted with background noise as described in Algorithm 3.1. With this test we are able to measure the robustness of the systems for fixed SNRs.

(ii) In a realistic scenario more distortions than the background noise arise. For that reason, we developed an evaluation protocol that aims to capture these distortions as well.

Below we give more details about these two protocols.

6.2.1 Off-line Evaluation

In the off-line evaluation we used the songs 17 songs listed in Table 6.1. For both systems, we performed the experiment by varing the query duration and the SNR of the background noise relative to the clean audio fragment. We examined the performance of both systems by considering all possible combinations of SNRs and query durations from the sets $\{0, 5, 10, 15\}$ (in dB) and $\{1, 2, 3, 4, 5, 10\}$ (in seconds) for the SNR, and query duration, respectively. Usually in these tasks, one is not only interested in determining which song is being played, but also finding the exact time position of the query with respect to the original audio recording. To this end, one introduces the *Top-1 hit rate (%)*, which is equivalent to *recall* with the extra requirement that the true positives not only correspond to the correct identification of the query audio fragment but also determining the time position within an acceptance tolerance corresponding to 500 ms. Formally, Top-1 hit rate is defined as follows: Suppose we have a finite sequence of queries $Q_1, \ldots, Q_M$, and the respective output predictions $P_1, \ldots, P_M$ of the system. Furthermore, denote by $q_1, \ldots, q_m$ the time position relative to the original audio fragment of the queries $Q_1, \ldots Q_M$, and by $p_1, \ldots, p_M$, the respective time positions of the output predictions $P_1, \ldots, P_M$. Then, Top-1 hit rate is given by

$$100 \times \sum_{j=1}^{M} \frac{\mathbb{1}_{Q_j = P_j} \cdot \mathbb{1}_{|q_j - p_j| < 0.5}}{M}, \tag{6.2.1}$$

where $\mathbb{1}_A$ is the indicator function on the set A.

6.2.2 Microphone Evaluation

For the microphone evaluation experiment[4], 15 songs from various genres were carefully selected to simulate a diverse range of audio sounds. These songs were normalized to achieve equal energy

[3] see [52] and https://micirp.blogspot.com

[4] Many thanks to Antonia, the creator of this experiment. GitHub: https://github.com/apetrogianni.

levels and then concatenated to create a single audio recording lasting 52 minutes. Similarly, an audio recording of the same duration, containing a variety of background noises, was generated. Subsequently, the recording containing the clean songs was played through a laptop device.

Meanwhile, a smartphone was connected to a bluetooth speaker, with the noisy recording starting to play alongside the query recording. The resulting sound was captured by the laptop's microphone. The speaker was positioned in three different locations relative to the laptop's position. As a result, three recordings (namely, low.wav, mid.wav, high.wav) were obtained, each exhibiting increasing distortions based on the position of the speaker. The high.wav recording corresponds to the closest position of the speaker, while the low.wav recording corresponds to the farthest position. We evaluate the performance of both systems by varying the query duration in the set $\{2, 5, 10, 15\}$. We report the accuracy of each system and evaluate the importance of applying the low/high pass filters on the neural network. As it is evident from the results presented in the upcoming sections, this augmentation improves the performance of the deep audio system by a huge margin, making it ideal for real-time applications.

Table 6.1. *Test Songs in Off-line Evaluation*

Song	Artist	Duration	Genre
Buffalo Soldier	Bob Marley	4:21	Reggae
Bad Like Jesse James	John Lee Hooker	5:21	Blues
On Melancholy Hill	Gorillaz	4:13	New wave, Synth-pop
The Promide You Made	Cock Robin	3:52	Jazz
Girls Like You	Maroon 5	3:52	Pop
Blinding Lights	The Weeknd	3:19	New wave, Synth-pop
Sorry	Madonna	4:41	Pop
Neglect	Mr. Kitty	3:35	Synth-pop, Darkwave
Running Up That Hill	Kate Bush	4:53	New Wave, Synth-pop
Set Fire to the Rain	Adele	4:02	Pop
Όταν σε είδα ξανά	Ταφ Λάθος	4:50	Greek Hip-Hop
Φίλα Με	Σταμάτης Κραουνάκης	3:10	Greek Entechno
Σβήσε το Φεγγάρι	Δημήτρης Μητροπάνος	4:01	Greek Laiko
Συγχαρητήρια	Άννα Βίσση	5:09	Greek Laiko, Entechno
Φίλε έλα απόψε που πονάω	Γιάννης Πουλόπουλος	3:21	Greek Laiko
Ανισόπεδη Ντίσκο	Pan Pan	4:06	Electro-pop
Queen of the Night Aria	Wolfgang Amadeus Mozart	3:10	Classical

Table 6.2. *Test Songs in Microphone Evaluation*

Song	Artist	Duration	Genre
Viva La Vida	Coldplay	4:02	Baroque Pop
Lookin Out My Back Door	Creedence Clearwater Revival	2:35	Country Rock
We Danced	Brad Paisley	3:46	Country
How Do You Sleep?	Sam Smith	3:22	Pop
The Four Seasons - Spring- Allegro	Antonio Vivaldi	3:15	Classical
It Must Have Been Love	Roxette	4:15	Pop
Love of a Lifetime	Fireouse	4:47	Glam metal
I've got you under my skin	Frank Sinatra	3:46	Jazz
You and your hand	Pink	3:33	Pop
Rude boy	Rihanna	3:44	Pop
Sweet Nothin's	Brenda Lee	2:20	Country
I don't wanna live without your love	Chicago	3:57	Soft Rock
A bird in the hand	Ice Cube	2:20	Hip-hop/rap
Summer is crazy	Alexia	4:21	Eurodance
Bad guy	Billie Eilish	3:14	electro-pop

6.3 dejavu: An Open Source Library for Shazam's Algorithm

In this section, we set up a music recognition system that make use of the fingerprinting method described in Section 3.5 to recognize short audio fragments captured in noisy and reverberant environments. To implement the audio fingerprinting system we use dejavu[5], an open source library which is written in Python. The code including all the experiments conducted for the purposes of the thesis is available on a public GitHub repository.[6]

6.3.1 Creating the Database

To setup the database we used the 26,016 songs as discussed in 6.1. These songs yielded a total of 58,659,291 fingerprints. The database corresponding to the audio fingerprints is represented by two MySQL tables. The first table, named SONGS, is created to keep track of all the necessary metadata related to the songs in the database, and the second table, named FINGERPRINTS, keeps the fingerprint hashes of the songs. Below, we describe these two tables in more detail.

(i) **SONGS:** The first table keeps track of all the relative metadata of the songs in database. The fields of this table are:

(1) *song_id*: A unique non-negative integer value serving as the primary key of the table..

(2) *song_name*: A string representing the name of the song.

[5] https://github.com/worldveil/dejavu
[6] https://github.com/ChrisNick92/deep-audio-fingerprinting

(3) *file_sha1*: A 10-byte binary string, represented in hexadecimal form, indicating the hash value of the file from which the fingerprints are extracted for the song.

(4) *total_hashes*: A non-negative integer value representing the total number of hashes stored in the database for the song..

(5) *date_created*: A date indicating when the song was added to the database.

(6) *date_modified*: A date indicating the most recent modification date for the song.

(ii) **FINGERPRINTS:** The second table corresponds to the audio representation of the database. It stores all the hashes that are extracted from the songs. Dejavu follows the method of Wang described in 3.5.3, where the hashes are generated by the triplets $(k_0, k_1, n_1 - n_0)$ as defined in (3.5.9). The table consists of the following fields:

(1) *hash*: A 10-byte binary string represented in hexadecimal form, corresponding to the hash resulting from the triplet $(k_0, k_1, n_1 - n_0)$.

(2) *song_id*: A non-negative integer value corresponding to the foreign key resulting from a one-to-many relationship with the SONGS table.

(3) *offset*: A non-negative integer corresponding to the time frame on the spectrogram in which the triplet $(k_0, k_1, n_1 - n_0)$ belongs. This index is important since we can retrieve the time stamp of the query, i.e., the time offset of the query.

(4) *data_created*: A date indicating when the fingerprint was added to the database.

(5) *date_modified*: A date indicating the most recent modification date for the fingerprint.

Figure 6.1 provides a graphical representation of Dejavu's database schema, and Table 6.3 the hyperparameters used to create the database with Dejavu.

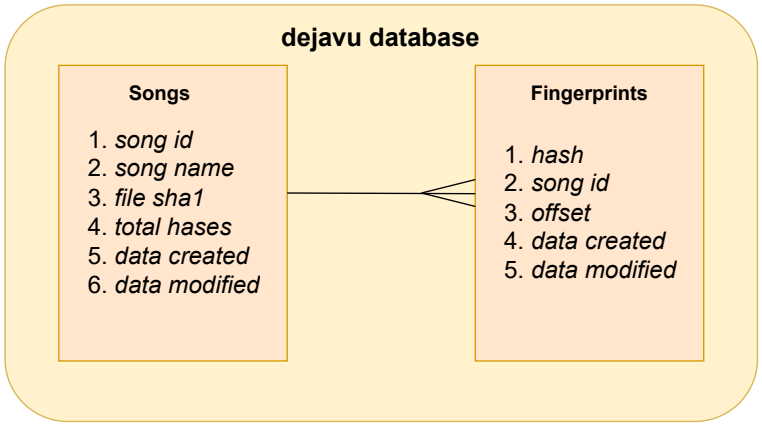

Figure 6.1. *Dejavu Database Schema.*

Table 6.3. *Hyperparameters and Statistics of Dejavu Database.*

Sampling Rate	$F_s = 8000$
Window length	$N = 4096$
Hop length	$H = 2048\,(50\%)$
Size of local maximum neighborhood	10×10 (cell)
Fan value	$F = 5$
Number of songs	26016
Number of fingerprints	58659291
Database size	6786.7 (MB)

As we can see from Table 6.3, the requirements of Dejavu with respect to required storage space are quite expensive since the size of the database is 6,7 GB for a total audio duration of 275 hours, 36 minutes, and 26 seconds. This requirement is not easily scalable to larger databases, and the implementation of such systems requires moving from a single device to multiple devices by setting up large clusters on the cloud. Tables 6.4 & 6.6 show some rows from these two MySql tables.

Table 6.4. *Table FINGERPRINTS.*

hash	sond_id	offset
0x0000FB78C9A4DBF91A15	1	1
0x07B98A2AF4E0B0E92045	1	1
0x08DBA50D5FB57DB14077	1	1
0x0BB512C2AD06356FEBE8	1	1
0x0C9BFDB024CC4EAE4C4F	1	1
0x0DFA37209D0902C33E85	1	1
0x0F81A32E220163D9EF73	1	1

Table 6.5. *Table SONGS.*

song_id	song_name	total_hashes
1	Kool & The Gang - Tonight (12')	14550
2	New Kids On The Block - I'll Be Loving You	13678
3	Levert - Casanova	13966
4	Teena Marie - Lover Girl	20606
5	Bob Seger - Fire Lake	11646
6	Cyndi Lauper - Time After Time	12854
7	Hall & Oates - You've Lost That Lovin' Feelin'	14422
8	Tina Turner - Better Be Good To Me	12898
9	Rick Springfield - Don't Talk To Strangers	10330
10	Wham - Careless Whisper	19306

6.3.2 Performance in the Off-line scenario

To evaluate the performance of the music recognition system based on dejavu we perform the off-line evaluation as described in subsection 6.2.1. The best matching part of the database is the one with the highest score in $\Delta_{\mathcal{Q}}(m)$ as defined in (3.5.7). Dejavu uses the term "input confidence" for $\Delta_{\mathcal{Q}}(m)$. Figure 6.2 shows how the input confidence varies with respect to the background noise with fixed SNR.

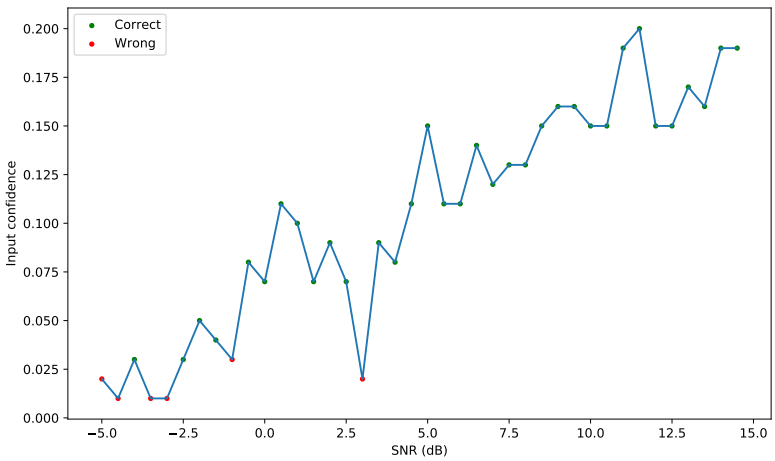

Figure 6.2. *Input confidence vs SNR in Dejavu. The query corresponds to 10 sec from the chorus of Blinding Lights by The Weeknd. Green dots correspond to correct predictions, and red dots to false predictions.*

As we can see, Dejavu is quite robust to high signal distortions as it correctly matches the audio fragment even in SNRs below 0. We can see that Dejavu correctly classified 34/40 queries, yielding a total of 85% accuracy in the song identification. However, usually in these tasks, one is not only interested in determining which song is being played, but also finding the exact time position of the query with respect to the original audio recording. To this end, we use Top-1-hit-rate as defined in 6.2.1 to evaluate the performance in the off-line scenario. Table 6.6 summarizes the results.

Table 6.6. *Performance of dejavu in the Off-line Evaluation.*

Metrics	Query length (s)	0 dB	5 dB	10 dB	15 dB
Accuracy	1	3.91%	5.91%	7.84 %	8.87 %
F1 score		6.86%	11.01%	14.12%	15.92%
Top-1 hit rate		**3.39%**	**4.72%**	**6.89%**	**7.30%**
Accuracy	2	28.06%	36.66%	42.45%	48.23&
F1 score		41.05%	52.94%	59.06%	64.61%
Top-1 hit rate		**24.27%**	**33.32%**	**38.48%**	**44.02%**
Accuracy	3	41.38%	60.70%	68.10%	73.78&
F1 score		57.36%	74.53%	80.50%	84.54%
Top-1 hit rate		**38.65%**	**56.82%**	**63.79%**	**69.32%**
Accuracy	4	52.06%	71.91%	81.30%	86.77&
F1 score		66.22%	82.83%	89.19%	92.79%
Top-1 hit rate		**48.32%**	**67.11%**	**75.84%**	**81.59%**
Accuracy	5	60.02%	79.83%	87.52%	91.60&
F1 score		73.31%	88.38%	93.23%	95.39%
Top-1 hit rate		**55.22%**	**75.15%**	**82.11%**	**86.55%**
Accuracy	10	82.85%	94.69%	98.07%	98.79&
F1 score		90.08%	97.25%	99.05%	99.40%
Top-1 hit rate		**77.78%**	**90.34%**	**93.48%**	**94.44%**

Table 6.6 shows that dejavu is quite sensitive to short queries (less than 4 seconds), scoring less than 50% for low SNRs. However, for 10-second queries, we observe that dejavu exhibits robust performance with a score of 77% even at 0 SNR. As the query duration increases, dejavu's performance undergoes dramatic changes, achieving very high scores. However, these long queries are impractical for real-time applications, where users expect results in just a few seconds.

6.4 Comparing the Music Recognition Systems

In this section, we compare the two music recognition systems in terms of their performance and scalability. We first present the results of the deep audio fingerprinting approach in the off-line evaluation, along with some implementation details. In the last subsection, we compare dejavu with the deep audio fingerprinting approach in the microphone evaluation, a test that better simulates real-world conditions. Additionally, we demonstrate that augmentations related to low and high pass filters are crucial for the deep neural network to achieve robust performance in scenarios where users capture the audio fragments from their portable devices.

6.4.1 Deep Audio Fingerprinting Performance in the Off-line Evaluation

The code that was produced for the purposes of this thesis is publicly available on GitHub[7]. The code is written in Python, and the deep learning framework that we used for the deep audio finger-

[7] https://github.com/ChrisNick92/deep-audio-fingerprinting

printing approach is PyTorch[8]. For the training part, we used the custom dataset as presented in Section 6.1. We split the dataset in 80-20 manner for the training, and validation sets, respectively. The 17 songs used for the off-line evaluation protocol (see 6.2.1) are excluded from these sets. For the data augmentations, Audiomentations[9] API was used. To create of the database of fingerprints based on the inverted file structured along with the product quantization we used faiss[10], an open source library developed by Meta AI. In detail, we create an IVFPQ (inverted file + product quantization) index as described in paragraph 5.3.1. The number of coarse centroids is equal to $k' = 20$, and the number of subquantizers is equal to $m = 64$. A total of $2^8 = 256$ centroids is chosen for each subquantizer. We extract a total of 1,973,384 fingerprints. For each song, we extract 1 fingerprint for each second with a hop size equal to 500 ms. This yield a total of 1,973,384 fingerprints. The database size is 137 MB. Furthermore, an additional structure represented as a Python dictionary keeps track of the correspondence between song names and their corresponding fingerprints in the faiss index. The size of this dictionary is 556 KB. Therefore, the size requirement of this approach is roughly 137 MB, significantly lower compared to dejavu, where the database size was 6.7 GB for the same dataset comprising 26,016 songs. Table 6.7 summarizes some of the most important hyperparameters and some statistics of the resulting database.

Table 6.7. *Hyperparameters and statistics in the deep audio fingerprinting approach*

Parameter	**Value (N)**
Sampling Rate	8000
STFT window function	*Hann*
STFT window and hop	1024, 256
STFT spectrogram size $F \times T$	$512 \times T (T = 32)$
low-power Mel-spectrogram size $F' \times T$	$256 \times T (T = 32)$
Fingerprint {window length, hop}	$\{1s, 0.5s\}$
Fingerprint dimension d	128
Network parameter embedding size h	1024
Batch Size	512
Number of songs	26,016
Number of Fingerprints	1,973,384
Number of subquantizers m	64
Number coarse centroids k'	20
Number of bits n	8

At query time, the first 5 coarse centroids out of 20 are scanned and first 4 nearest neighbors are returned. We measure the distance based on the inner product. This means, that higher distance values correspond to points closest to the query. We employ the same searching technique as introduced in [14], referred to as *sequence search* to determine the starting position of the query audio fragment with a time tolerance of ±500 ms. More formally, for a query sequence $\{x_i\}_{i=0}^{L}$ consisting of L consecutive segments: We first gather the top k segment-level search results indices

[8] https://pytorch.org
[9] https://github.com/iver56/audiomentations
[10] https://github.com/facebookresearch/faiss

I_i from the database, for each query segment x_i. The offset is then compensated by $I'_i = I_i - i$. The set of candidates indices $c \in C$ is determined by taking unique elements of $\bigcup_{i=0}^{L} I'_i$. The sequence-level similarity score is the sum of all segment-level similarities from the segment index range $[c, c + L]$, and the index with the highest score is the output of the system. The name of song is returned using the python dictionary. To test the performance, we use the 17 songs from Table 6.1. In Table 6.8 you can see the performance of the system for varying SNR values and query lengths.

Table 6.8. *Performance of Deep Audio Fingerprinting System.*

Metrics	**Query length (s)**	0 dB	5 dB	10 dB	15 dB
Accuracy		56.80%	81.19%	91.73 %	94.78 %
F1 score	1	70.79%	88.73%	95.10%	96.84%
Top-1 hit rate		**45.30%**	**66.52%**	**78.33%**	**84.48%**
Accuracy		78.11%	92.54 %	97.56 %	98.61 &
F1 score	2	86.80 %	95.54 %	98.43 %	99.06 %
Top-1 hit rate		**63.77 %**	**78.35 %**	**87.09 %**	**89.63 %**
Accuracy		83.41 %	96.12 %	98.49 %	99.07 &
F1 score	3	90.25 %	97.55 %	99.04 %	99.33 %
Top-1 hit rate		**69.25 %**	**83.41 %**	**89.44 %**	**92.82 %**
Accuracy		87.73 %	97.32 %	98.75 %	98.95 &
F1 score	4	92.92 %	98.39 %	99.20 %	99.29 %
Top-1 hit rate		**73.44 %**	**85.52 %**	**91.75 %**	**92.62 %**
Accuracy		89.32 %	97.48 %	98.56 %	99.16 &
F1 score	5	93.58 %	98.35 %	99.07 %	99.39 %
Top-1 hit rate		**75.99 %**	**85.52 %**	**89.68 %**	**93.88 %**
Accuracy		95.65 %	98.07 %	98.55 %	98.31 &
F1 score	10	97.27 %	98.86 %	99.06 %	98.97 %
Top-1 hit rate		**78.50 %**	**84.30 %**	**88.89 %**	**92.51 %**

As we can observe, the deep audio approach surpasses dejavu by a substantial margin, particularly for short queries of length less than 10 seconds. As the query length increases, the performance of dejavu significantly improves compared to the deep audio fingerprinting system. However, this becomes impractical for real-time applications where the retrieval process needs to be as fast as possible. Table 6.9 shows the inference time through the model, the query time to search on the database for the best matching index, and the total time required to return the output.

Table 6.9. *Inference, Query, and Total time*

Query length (sec)	1	2	3	4	5	10
Inference time (ms)	138.13	185.38	226.59	262.43	300.15	502.24
Query time (ms)	3.85	5.07	8.18	10.39	13.05	24.11
Total retrieval time (ms)	141.98	190.46	234.77	272.83	313.20	526.35

6.4.2 Deep Audio vs. Dejavu in Microphone Evaluation

Up to this point, we have evaluated the performance of the music recognition systems by introducing background noise to the clean audio signals. However, in a realistic scenario where the query signal is captured in noisy environments, additional distortions may also occur. Furthermore, these music recognition systems typically operate on portable devices, meaning that query audio fragments are captured through the devices' microphones. Therefore, it is necessary to test the performance of both systems in conditions that reflect these real-world scenarios. To accomplish this, in this subsection, we compare the performance of the music recognition systems using the microphone evaluation, as presented in 6.2.2. Additionally, we test the performance of the deep audio approach, and examine whether the augmentations with regard to the high/low pass filters improve the accuracy of the model. Table 6.10 summarizes the results of the experiment.

Table 6.10. *Performance (Accuracy) of the Music Recognition Systems on Microphone Experiment.*

Metrics	**Query length (s)**	low	mid	high
Deep Audio	2	**85.03%**	**62.65%**	**41.92%**
Deep Audio (No Filters)		67.22%	49.02%	19.47%
Dejavu		12.63%	6.50%	10.73%
Deep Audio	5	**92.66%**	**80.06%**	**62.52%**
Deep Audio (No Filters)		80.38%	65.39%	31.10%
Dejavu		59.21%	40.94%	47.24%
Deep Audio	10	**94.52%**	**90.32%**	**74.19%**
Deep Audio (No Filters)		84.19%	78.39%	41.29%
Dejavu		83.60%	70.98%	71.29%
Deep Audio	15	**97.56%**	**90.24%**	80.98%
Deep Audio (No Filters)		86.34%	80.98%	48.29%
Dejavu		93.33%	77.14%	**83.33%**

Chapter 7

Conclusions

In this thesis, we have presented the basic theory and techniques used in audio signal processing. We focused on the task of song identification and developed two different music recognition systems. The first music recognition system is based on the well-established Shazam's algorithm, representing the traditional approach to song identification. In contrast, the second approach leverages recent advances in deep learning by training a deep neural network to exhibit robust performance against signal distortions.

We presented the underlying theory of each system and compared their performance through a series of experiments. One major drawback of the classic fingerprinting approach is the rapid expansion of the database as more songs are added to the system. Specifically, we found that the database size of dejavu was 6.7 GB for a total of 26,016 songs. In the case of the deep audio fingerprinting approach, we achieved a database size of 137 MB for the same number of songs by employing product quantization techniques. This reduction is possible because the vector representation extracted from the raw audio signals by the model benefits from the extensive theory of Approximate Nearest Neighbor search.

Furthermore, we observed that by applying suitable augmentations to the model, the deep audio approach can outperform the classic approach, especially for short queries, making it well-suited for real-time applications. On the positive side, music recognition systems using the classic fingerprinting approach maintain consistent performance regardless of the distortions that may be present. In contrast, if the model is not specifically trained to handle certain distortions, its performance can significantly deteriorate when those distortions are encountered in a real-world scenario.

However, with the growth in computational power and advancements in deep learning, approaches that leverage these aspects tend to outperform classic methods by a substantial margin in various domains, including music information retrieval. For future directions in the song identification task, an interesting paper to consider is *Self-supervised Product Quantization for Deep Unsupervised Image Retrieval* [53], in which the authors present an end-to-end system that jointly trains a deep neural network to learn both the latent feature representations of images and the binary codes of product quantization. In their paper, they develop an efficient image retrieval system that achieves state-of-the-art performance when compared to existing systems.

Bibliography

[1] Avery Wang et al. An industrial strength audio search algorithm. In *Ismir*, volume 2003, pages 7–13. Washington, DC, 2003.

[2] Meinard Müller. *Fundamentals of music processing: Audio, analysis, algorithms, applications*, volume 5. Springer, 2015.

[3] Yitzhak Katznelson. *An introduction to harmonic analysis*. Cambridge University Press, 2004.

[4] Brad Osgood. The fourier transform and its applications. *Lecture notes for EE*, 261:20, 2009.

[5] Elias M Stein and Guido Weiss. *Introduction to Fourier analysis on Euclidean spaces*, volume 1. Princeton university press, 1971.

[6] Gerald B Folland. *Real analysis: modern techniques and their applications*, volume 40. John Wiley & Sons, 1999.

[7] Halsey Lawrence Royden and Patrick Fitzpatrick. *Real analysis*, volume 32. Macmillan New York, 1988.

[8] Terence Tao. *An introduction to measure theory*, volume 126. American Mathematical Soc., 2011.

[9] Jorge Calvo-Zaragoza, Jan Hajič Jr, and Alexander Pacha. Understanding optical music recognition. *ACM Computing Surveys (CSUR)*, 53(4):1–35, 2020.

[10] Eelco van Der Wel and Karen Ullrich. Optical music recognition with convolutional sequence-to-sequence models. *arXiv preprint arXiv:1707.04877*, 2017.

[11] Elona Shatri and György Fazekas. Optical music recognition: State of the art and major challenges. *arXiv preprint arXiv:2006.07885*, 2020.

[12] Brian McFee, Colin Raffel, Dawen Liang, Daniel P Ellis, Matt McVicar, Eric Battenberg, and Oriol Nieto. librosa: Audio and music signal analysis in python. In *Proceedings of the 14th python in science conference*, volume 8, pages 18–25, 2015.

[13] Beat Gfeller, Ruiqi Guo, Kevin Kilgour, Sanjiv Kumar, James Lyon, Julian Odell, Marvin Ritter, Dominik Roblek, Matthew Sharifi, Mihajlo Velimirović, et al. Now playing: Continuous low-power music recognition. *arXiv preprint arXiv:1711.10958*, 2017.

[14] Sungkyun Chang, Donmoon Lee, Jeongsoo Park, Hyungui Lim, Kyogu Lee, Karam Ko, and Yoonchang Han. Neural audio fingerprint for high-specific audio retrieval based on contrastive learning. In *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 3025–3029. IEEE, 2021.

[15] Anup Singh, Kris Demuynck, and Vipul Arora. Attention-based audio embeddings for query-by-example. *arXiv preprint arXiv:2210.08624*, 2022.

[16] Judith C Brown. Calculation of a constant q spectral transform. *The Journal of the Acoustical Society of America*, 89(1):425–434, 1991.

[17] Xingjian Du, Zhesong Yu, Bilei Zhu, Xiaoou Chen, and Zejun Ma. Bytecover: Cover song identification via multi-loss training. In *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 551–555. IEEE, 2021.

[18] Xingjian Du, Ke Chen, Zijie Wang, Bilei Zhu, and Zejun Ma. Bytecover2: Towards dimensionality reduction of latent embedding for efficient cover song identification. In *ICASSP 2022-2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 616–620. IEEE, 2022.

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

[20] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[21] Chigozie Nwankpa, Winifred Ijomah, Anthony Gachagan, and Stephen Marshall. Activation functions: Comparison of trends in practice and research for deep learning. *arXiv preprint arXiv:1811.03378*, 2018.

[22] Shiv Ram Dubey, Satish Kumar Singh, and Bidyut Baran Chaudhuri. Activation functions in deep learning: A comprehensive survey and benchmark. *Neurocomputing*, 2022.

[23] Steven C Bagley, Halbert White, and Beatrice A Golomb. Logistic regression in the medical literature:: Standards for use and reporting, with particular attention to one medical domain. *Journal of clinical epidemiology*, 54(10):979–985, 2001.

[24] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

[25] Andrew L Maas, Awni Y Hannun, Andrew Y Ng, et al. Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml*, volume 30, page 3. Atlanta, GA, 2013.

[26] Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. Empirical evaluation of rectified activations in convolutional network. *arXiv preprint arXiv:1505.00853*, 2015.

[27] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*, 2015.

[28] Sergios Theodoridis and Konstantinos Koutroumbas. *Pattern recognition*. Elsevier, 2006.

[29] Dongkuan Xu and Yingjie Tian. A comprehensive survey of clustering algorithms. *Annals of Data Science*, 2:165–193, 2015.

[30] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *International conference on machine learning*, pages 1597–1607. PMLR, 2020.

[31] Dami Choi, Christopher J Shallue, Zachary Nado, Jaehoon Lee, Chris J Maddison, and George E Dahl. On empirical comparisons of optimizers for deep learning. *arXiv preprint arXiv:1910.05446*, 2019.

[32] Boris T Polyak. Some methods of speeding up the convergence of iteration methods. *Ussr computational mathematics and mathematical physics*, 4(5):1–17, 1964.

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

[34] Yang You, Jing Li, Sashank Reddi, Jonathan Hseu, Sanjiv Kumar, Srinadh Bhojanapalli, Xiaodan Song, James Demmel, Kurt Keutzer, and Cho-Jui Hsieh. Large batch optimization for deep learning: Training bert in 76 minutes. *arXiv preprint arXiv:1904.00962*, 2019.

[35] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

[36] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.

[37] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015.

[38] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[39] Jianxin Wu. Introduction to convolutional neural networks. *National Key Lab for Novel Software Technology. Nanjing University. China*, 5(23):495, 2017.

[40] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[41] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. pmlr, 2015.

[42] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[43] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[44] Herve Jegou, Matthijs Douze, and Cordelia Schmid. Product quantization for nearest neighbor search. *IEEE transactions on pattern analysis and machine intelligence*, 33(1):117–128, 2010.

[45] Kevin Beyer, Jonathan Goldstein, Raghu Ramakrishnan, and Uri Shaft. When is "nearest neighbor" meaningful? In *Database Theory—ICDT'99: 7th International Conference Jerusalem, Israel, January 10–12, 1999 Proceedings 7*, pages 217–235. Springer, 1999.

[46] Christian Böhm, Stefan Berchtold, and Daniel A Keim. Searching in high-dimensional spaces: Index structures for improving the performance of multimedia databases. *ACM Computing Surveys (CSUR)*, 33(3):322–373, 2001.

[47] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the twentieth annual symposium on Computational geometry*, pages 253–262, 2004.

[48] Aristides Gionis, Piotr Indyk, Rajeev Motwani, et al. Similarity search in high dimensions via hashing. In *Vldb*, volume 99, pages 518–529, 1999.

[49] Marius Muja and David G Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. *VISAPP (1)*, 2(331-340):2, 2009.

[50] Robert M. Gray and David L. Neuhoff. Quantization. *IEEE transactions on information theory*, 44(6):2325–2383, 1998.

[51] Michaël Defferrard, Kirell Benzi, Pierre Vandergheynst, and Xavier Bresson. FMA: A dataset for music analysis. In *18th International Society for Music Information Retrieval Conference (ISMIR)*, 2017.

[52] Marco Jeub, Magnus Schafer, and Peter Vary. A binaural room impulse response database for the evaluation of dereverberation algorithms. In *2009 16th International Conference on Digital Signal Processing*, pages 1–5. IEEE, 2009.

[53] Young Kyun Jang and Nam Ik Cho. Self-supervised product quantization for deep unsupervised image retrieval. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 12085–12094, 2021.

List of Abbreviations

BPF	Band Pass Filter
Hz	Hertz
FT	Fourier Transform
IFT	Inverse Fourier Transform
DFT	Discrete Fourier Transform
DT	Discrete Time Signal
CT	Continuous Time Signal
STFT	Short-Time Fourier Transform
CNN	Convolutional Neural Network
OCR	Optical Character Recognition
OMR	Optical Music Recognition
TOH	Threshold of Hearing
TOP	Threshold of Pain
CQT	Continuous Q Transform
CSI	Cover Song Identification
MIR	Music Information Retrieval
SNR	Signal to Noise Ratio
AI	Artificial Intelligence
MLP	Multi Layer Perceptron
DL	Deep Learning
MSE	Mean Squared Error
MAE	Mean Absolute Error
SGD	Stochastic Gradient Descent
ANN	Approximate Nearest Neighbor

Index