Analyses of Convolutional Neural Networks for Automatic tagging of music tracks

Master Thesis

Aravind Sankaran

Supervisors
Prof. Paolo Bientinesi
Prof. Marco Alunno

Examiners
Prof. Paolo Bientinesi
Prof. Bastian Leibe

Abstract

Describing music can be quite tricky and talking about music may simply require as much vocabulary as any technical subject. Musicians and composers usually discuss their work with jargon describing certain aesthetics of the song. As the amount of music recordings are constantly growing, finding a song that matches these aesthetic description is challenging. This work is an attempt to take a step towards developing algorithms that could tag music like an artist. The currently available state-of-art algorithms are trained and tested on datasets with tags that are socially biased. Moreover these datasets contain just short clips of songs, but an artist describes a song as a whole. Therefore, in this thesis, a repertoire of 900 songs with carefully labelled data describing the whole track is used and the aim is to find the model settings that would best approximate the audio features for such a dataset. The Mel-Frequency-Cepstral-Coefficients (MFCC) features are compared with features extracted by pre-trained Convolution Neural Networks (CNN) over mel-log power spectrogram. These features are extracted every 29.1s and approximated over time to a fixed size representation. The temporal approximation by sequence to one Long Term Short Memory (LSTM) Recurrent Neural Network is compared with approximation by Bag of Frames (BoF) approach and the weighted area under receiver operating characteristic curve (AUC) is reported. The experiments show that MFCC features summarized by LSTM outperforms pre-trained convolutions counterpart. It is also seen that LSTM perform better than Bag of Frames features for temporal approximation.

Acknowledgments

I would like to thank *Prof. Paolo Bientinesi* (High performance and automatic comupting group, AICES, RWTH Aachen) for doing one of the most expensive work - *Listen to almost thousand songs and tag them.* I also thank *Prof. Marco Alunno* (Professor of Composition and Theory at University EAFIT, Columbia) for his valuable association and advices.

Contents

1 Introduction **5**

1.1 Motivation . . . 5

1.1.1 Cold start problem with collaborative filtering methods . . . 5

1.1.2 Problems with content based methods . . . 6

1.2 Overview . . . 6

1.2.1 Signal Representation . . . 7

1.2.2 Dimensionality reduction . . . 7

1.2.3 Temporal approximation . . . 8

1.3 Outline of the report . . . 8

2 Formalisms **9**

2.1 Representation of music signal . . . 9

2.1.1 Sampling of continuous-time signal . . . 9

2.1.2 Time-Frequency transformations . . . 10

2.1.3 STFT, Mel-Spectrogram . . . 11

2.2 Dimensionality Reduction . . . 14

2.2.1 Domain engineering Vs Unsupervised learning Vs Supervised learning . . . 15

2.2.2 Principal Component Analysis (PCA) . . . 16

2.2.3 Mel-frequency cepstrum coefficients (MFCC) . . . 17

2.2.4 Convolution neural network . . . 18

2.3 Temporal Approximation . . . 21

2.3.1 Bag Of Frames . . . 21

2.3.2 Recurrent Neural Networks . . . 22

2.4 Multi-label Classifier . . . 24

2.4.1 Two-layer perceptron . . . 25

2.5 Training . . . 25

2.5.1 First-order gradient descent . . . 26

2.5.2 Deep learning . . . 28

3 Literature Survey and Model Selection **31**

3.1 Literature Review . . . 31

3.1.1 From classifier to feature emphasis . . . 32

3.1.2 From hand-crafting to feature learning . . . 32

3.1.3 Transfer Learning by supervised pre-training 34
3.1.4 Convolutional Neural Networks 35
3.2 Model Selection 38
3.2.1 Transfer learning Vs MFCC 38
3.2.2 Bag Of Frames vs RNN 38

4 Experiments and Results **41**
4.1 Dataset and Evaluation 41
4.1.1 Dataset for source task 41
4.1.2 Dataset for target task 42
4.1.3 Evaluation metrics 42
4.2 Experiments 43
4.2.1 Experiments with pre-trained CNNs as feature extractor 44
4.2.2 Experiments with MFCCs as feature extractor 46
4.3 Summary of Results 47

5 Conclusion **49**

A Appendix **51**
A.1 Basis Transformation 51
A.2 Convolution 51
A.2.1 1D Convolution 51
A.2.2 2D Convolution 52

Bibliography **55**

Chapter 1

Introduction

Computers have been used to automate discovery and management of music in so many different ways. Automating the task of attaching a semantic meaning to a song is popularly known as *music auto-tagging*. Automatic tagging algorithms have been used to build recommendation systems that allow listeners to discover songs that match their taste. It also enables online music stores to filter their target audience. But semantic description of a song is not straightforward and there is this gap between music audio and listener's description, both linguistically and emotionally, which we term as *audio-semantic gap*. In this thesis, we test the state-of-art approaches of music auto-tagging on a dataset that is least affected by *audio-semantic* noise and find out the efficient model settings. In section 1.1, the need for this dedicated research is explained by describing some shortcomings of the currently available solution approaches. In section 1.2, a top to bottom overview of the contents of this research is presented.

1.1 Motivation

Developing an algorithm that imitates the human way of describing auditory scene is an interesting application. Although great technical progress have been made to enable efficient retrieval and organization of audio content, analysing music and communicating it's musical syntax that respects multidimensional qualities of sound is still challenging. Current music recommendation systems fall short in providing recommendations by respecting the criteria emerging from perceptual qualities of music because of the reasons described below,

1.1.1 Cold start problem with collaborative filtering methods

When the usage data is available, one can use collaborative filtering to recommend the tracks on a community-based trending lists (say, a community of experts). That is, if a listener liked songs A and B and you liked A, you might also like B. Such algorithms have proven to be extremely efficient and out-perform those algorithms that works by extracting acoustic cues from audio signal for the task of finding similar songs [16]. However, in absence of such usage data, one resorts to content-based methods, where just the audio signal is used for generating recommendations. Thus collaborative filtering methods suffer from what is called a *cold start problem*, making it less efficient for new and unpopular songs.

1.1.2 Problems with content based methods

Using information from audio content to overcome the cold-start problem resulted in *content-based* recommendation methods. In such algorithms, a classifier is trained on some training data to learn acoustic cues. But a recommendation system can also be built without requiring such acoustic labels by combining *content based* and *collaborative* techniques[9] and training it from a collaborative view point. However, this is not sufficient if a recommendation system has to designed for artists and composers who search for songs based on some musical criteria. In such cases, the current *content-based* methods fall short because of following training assumption,

Temporal summarization of audio content

The current state of art music tagging algorithms[23][17] are established by training on datasets that contain just short music excerpts. In run-time, these algorithms classify each short section separately and merge tags across different sections. But a composer might describe the music as a whole and not in sections. Hence there is a need to test algorithms that extract features approximating greater length of the song.

Restricted glossary

Current recommendation systems including the ones that use collaborative filtering, restrict the user with the choice of tags. Moreover, it is not guaranteed that all users will perceive all the tags in the same way. A recent study in idiographic music psychology have indicated that different people use different aesthetic criteria to make judgement about music[24]. Hence it would be interesting to study if these models [22][23] trained on large datasets can be exploited for training on personal repertoire, which are usually small.

1.2 Overview

In an attempt to get rid of the *audio-semantic* noise, we use a dataset tagged by an expert and assumptions about song length are not made (Let us call them *aesthetic tags*). But such datasets are usually small because gathering them is expensive. Convolution neural networks (CNN) have recently gained popularity for content-based multi-label classification task achieving state-of-art performance[22][23] on established datasets[13][8]. But these models were trained on large amount of training data containing short excerpts of music and it is not clear if section-wise merging of descriptions can approximate actual description of the whole song. So the aim of this thesis is to find out

- If the celebrated CNN models trained on large data can be exploited to show similar performance gains when *fine-tuned* on a small dataset (Generally termed as *transfer learning*).
- Models that can best approximate signals of arbitrary length to a fixed size representation (needed for classification).

The classification is done on a lower dimensional approximation of the audio signal known as *feature*. The general pipeline for music feature extraction is *signal representation*, systematic *dimensionality-reduction* followed by *temporal approximation*.

1.2.1 Signal Representation

Music is distinguished by the presence of many relationships that can be treated mathematically, including rhythm and harmony, by analysing the frequency content. In Chapter 2 (Sec. 2.1.2), representation of digital audio in time-frequency format is explained. Motivated by the way ear-brain handles the frequency information, myriad of features extracted from spectrogram representations were evolved. Each one of them will fit into one of these broad categories:

- **Mel based spectrogram :** Exploiting the fact that our ear cannot distinguish adjacent frequencies (say we cannot differentiate 300 KHz and 310 KHz), the information pertaining to frequencies are binned according to a what to popularly known as *mel-scale*. The features (MFCCs, etc). obtained from this representation are useful for any general purpose application like speech recognition, genre classification etc.
- **Chromagram :** Frequencies are binned by taking advantage of the periodic perception of *pitch* (perceived frequency). Features (Tonnetz, etc) following from this representation find applications in music mixing, chord recognition etc.
- **Tempogram :** For applications like tempo estimation and onset detection, the representation should encode the *change* of frequencies over time. This resulted in a set of features (Novelty curve, beat centroid etc) calculated from *differential* spectrograms.

In this thesis, we only study *mel-based spectrogram* representations. Chroma and tempo features can be obtained after binning to mel-scale (not necessarily), and in this sense mel-spectrogram can be viewed as a super set. But one should look at this categorization in terms of mathematical operation. Features resulting from *mel-spectrogram* means *explicit* operations involving periodic binning and temporal differentiation are not involved. I use the word *explicit* because it will be shown later that tempo and chroma information can be *implicitly* modelled by *feature learning* with CNNs (see Chapter 2, 2.2.4).

1.2.2 Dimensionality reduction

Features are usually obtained as a result of some dimensionality reduction operation on the spectrogram. In Chapter 2 (Sec. **??**), dimensionality reduction through *basis transformation* is introduced in terms of convolution operation and the extraction of Mel-Frequency-Cepstral-Coefficients (MFCCs) is explained. Then in section 2.2.4, generalizing the MFCC computation into a learning problem by replacing the basis functions with learnable convolutional filters is discussed and eventually explaining the success of convolution neural networks. In Chapter 3, some of the successful algorithms reported for music-auto-tagging are discussed. Transfer-learning using CNN model in [22] which reports close to state-of-art performance is compared with MFCC features for our target task in Chapter 4 (Sec. 4.2) .

1.2.3 Temporal approximation

We are looking for an algorithm that will approximate features for songs of arbitrary length without losing much of the rhythmic information. In the current literature, this is handled using *Bag of Frame* approach which is explained in Chapter 2, section 2.3.1. But as the reported performance are for 29.1s song clips, their optimality for songs of greater length is questioned. So we test *Recurrent neural network (Sequence to One LSTM)* which is more motivating to use for rhythmic information extraction (see 2.3.2). In Chapter 4 (Sec. 4.2), the performance of approximation by *Bag of Frames* and *Recurrent neural network* is reported.

1.3 Outline of the report

In chapter 2, the fundamentals required for understanding remaining contents are explained and formalism of notations are introduced. In chapter 3, a detailed overview of previous research, their shortcomings for the current problem along with justification for proposed models are discussed. In chapter 4, details of the dataset and the experiment results of proposed models are discussed. In chapter 5, the inference from these experiments in understanding the development of algorithms for tagging music with aesthetic tags is explained and future directions are discussed.

Chapter 2

Formalisms

To map a music to it's semantic descriptions, the signal is first transformed to a lower dimensional space. The resulting transformation is called *feature*. The classifier then takes these features as input and performs the classification task. The performance of the classifier can only be as good as the information encoded in the features. In this chapter, formalisms required for the analysis of feature extraction are introduced. The raw signal is first changed to a representation that can be mathematically analysed (see Sec. 2.1). This representation is then transformed to a lower dimensional space by discarding information that does not contribute to discrimination (see Sec. 2.2). The resulting reduction is then approximated to a fixed size representation (see Sec. 2.3) which is then taken as input by a multi-label classifier (see Sec. 2.4). In section 2.5, training procedure that can optimize feature representations for supervised classification task is discussed.

2.1 Representation of music signal

The observed signal is traditionally represented in the time domain. The time domain is a record of what happened to a parameter of the system versus time. Standard formats use amplitude versus time. The observed signal is then discretised by sampling and stored in digital format (see 2.1.1). This signal in the time domain is then changed to frequency domain (see 2.1.2). This is simply because our ear-brain combination is an excellent frequency domain analyser. Currently used music signal representations for general MIR tasks are explained in section 2.1.3.

2.1.1 Sampling of continuous-time signal

The digital formats contain the discrete version of the signal obtained by sampling continuous-time signal. Sampling is performed by measuring the value of the continuous signal every T seconds. This interval T is called the sampling interval or the sampling period. The sampling frequency or sampling rate (f_s) is the number of samples obtained in one second (samples per second),

$$f_s = \frac{1}{T}.$$

The optimum sampling rate is given by Nyquist-Shannon sampling theorem which says, the sampling frequency f_s should be at least twice the highest frequency contained in the signal. Given the human hearing range lies between 20Hz - 20KHz, most of the digital audio formats use a standard sampling frequency of 44.4Khz. Sampling rate determines the initial dimension of the raw signal. The signal may be further down sampled if higher sampling rate does not contribute to classification performance.

2.1.2 Time-Frequency transformations

The digital signal is represented in the time domain with amplitude values at each time t. This representation has to be changed to frequency domain. Mapping from time-domain to frequency-domain is looked up on as basis transformation.

Basis transformation from time to frequency domain

The signal in the time domain $\mathbf{a}$ is a set of ordered n-tuples of real numbers $(a_1, a_2, ..., a_N) \in \mathbb{R}^N$ in the vector space V , specifically *Euclidean n-space.* That is to say, a discrete-time signal can be represented as a linear combination of Cartesian basis vectors. The coefficients in linear combination are then the co-ordinates of the corresponding basis system.

$$\mathbf{a} = (a_1, a_2, ..., a_N) = a_1\mathbf{e}_1 + a_2\mathbf{e}_2 + ... + a_N\mathbf{e}_N = \sum_{t=1}^{N} a_t\mathbf{e}_t = \mathbb{I}\mathbf{a}$$

Where $(a_1, a_2, ..., a_N)$ are the co-ordinates of Cartesian basis formed by basis vectors $\mathbf{e}_1...\mathbf{e}_N$ with $t \in \{1, 2, ..N\}$. The unit vector $\mathbf{e}_t \in \mathbb{R}^N$ has 1 in the t^{th} index and 0 everywhere else.

To transform to frequency domain, we need to find a set of basis vectors $\boldsymbol{\phi}_\omega$ that are functions of frequencies (ω). Then the co-ordinates of this basis system c_ω represents the signal in frequency domain.

$$\mathbf{a} = \sum_{\omega=1}^{M} c_\omega\boldsymbol{\phi}_\omega = \boldsymbol{\Phi}\mathbf{c} \qquad \boldsymbol{\Phi} \in \mathbb{C}^{N\times M}, \mathbf{c} \in \mathbb{C}^M \tag{2.1}$$

If $\boldsymbol{\Phi}$ is known, then the transformed co-ordinates $\mathbf{c} = (c_1, c_2, .., c_M)$ can be computed as,

$$\mathbf{c} = \boldsymbol{\Phi}^{-1}\mathbf{a}$$

$\mathbf{c}$ is the transformed representation. $\boldsymbol{\Phi}^{-1}$ is the operator that transforms the signal.

Exponential Fourier Series and Fourier Transform

From the definition of *exponential Fourier Series*, any function *periodic* in $\{1, 2, .., T\}$ can be expanded with series of complex exponentials[4]. These complex exponentials which are functions of harmonically related frequencies($k\omega$) form basis

$$\boldsymbol{\phi}_k(t) = \frac{1}{\sqrt{T}}e^{ik\omega t} \qquad t \in \{1, 2, ..T\}$$

It is difficult to assume periodicity for a generalized signal. Hence, the Fourier series can not be applied directly and Fourier Transform was developed. By Fourier transform, quantity of each frequency component ω in an arbitrary signal $\mathbf{a}(t)$ can be computed by dividing by $e^{i\omega t}$. Application of Fourier transform to a discrete signal is called *Discrete Fourier Transform*

$$c_\omega = \sum_{t=1}^{N} a_t e^{-i\omega t} \qquad t \in \{1, 2, ..., N\}$$

Thus the transformation operator is,

$$\mathbf{\Phi}^{-1}[\omega] = e^{-i\omega \mathbf{t}} \qquad \mathbf{t} \in \mathbb{R}^N, \omega \in \{1, 2, .., M\}$$

Fast Fourier Transform(FFT) is an efficient implementation of Discrete Fourier Transform(DFT) which exploits the symmetry of *sines* and *cosines* in the complex exponential. While DFT requires $O(N^2)$ operations, FFT requires only $O(NlogN)$ [4].

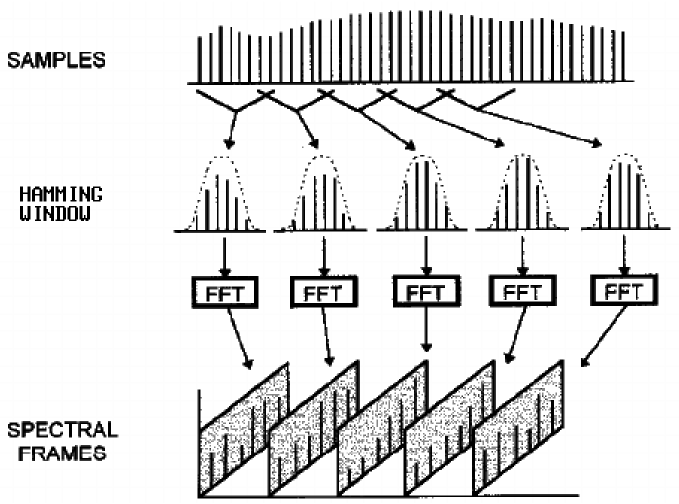
2.1.3 STFT, Mel-Spectrogram

It is useful to perform FFT locally over short segments. This is because we are more interested in the evolution of frequency content rather than the frequency content of the entire signal. Hence, the full length signal is divided into short segments, and FFT is computed separately for each segment. This is known as **Short Time Fourier Transform (STFT)**. One common problem with STFT is the *spectral leakage*, which is addressed by modifying the original signal with some window function. The most common window function is the **Hamming Window** defined as,

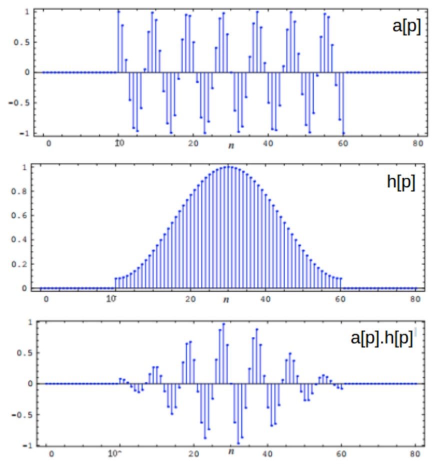
$$h[n] = 0.54 - 0.46cos(\frac{2\pi n}{N-1}) \tag{2.2}$$

where $n \in \{1, 2, .., F\}$ and F is the size of window function. The signal approaches zero near $n = 1$ and $n = N$, but reaches peak near $n = N/2$ [32]. To overcome the information loss at the ends of the window, signal is divided into segments that are partly *overlapping* with each other. The distance between the start of two adjacent segments is called *hop-length*. Figure 2.1a shows the extraction of spectral frames of a spectrogram. Thus, the parameters of STFT include

- Choice of window function
- Size of each segment in $\mathbf{a}$ (F)
- Hop length or stride (s) of the transformation operator
- Size of frequency dimension M (also known as FFT size)



(a) Windowing is applied on overlapping segments followed by FFT



(b) Application of Hamming Window on a segment of input signal

Figure 2.1: [30] (a) Shows STFT Pipeline. (b) Shows the application of Window function

STFT as Convolution

The strided operation over the signal **a** is mathematically represented as a **convolution**. The resulting spectrogram has P frames. The discrete STFT (*slow*) for p^{th} frame of signal **a** is obtained as,

$$\mathbf{C}[p,\omega] = \sum_{n=p.s}^{p.s+F} \mathbf{a}[n]\Big(\mathbf{h}[n-p.s].e^{-i\omega(n-p.s)}\Big) \tag{2.3}$$

Where:

P: is the number of spectral frames; $p \in [1, 2..., P]$

M: is the dimension of discrete frequency space ; $\omega \in \mathbb{R}^M$

F: is the frame length

s: is stride (or) hop-length for the next segment

$\mathbf{a} \in \mathbb{R}^N$; $n \in [1, 2, ..., N]$

$\mathbf{h} \in \mathbb{R}^F$

$\omega \in \mathbb{R}^M$

$\mathbf{C} \in \mathbb{C}^{M\times P}$

Equation 2.3 can be seen as a **discrete convolution** over the signal **a** with the operator $\mathbf{W}_{STFT}$ which has finite support over the set $\{1.., F\}$ with stride s

$$\mathbf{C} = \mathbf{a} \star \left(\mathbf{h}\boldsymbol{\Phi}^{-1}\right)$$

$$\mathbf{C} = \mathbf{a} \star \mathbf{W}^{(s)}{}_{STFT} \tag{2.4}$$

where $\mathbf{W}_{STFT} = \mathbf{h}\mathbf{\Phi}^{-1}$ is the STFT operator that transforms the signal $\mathbf{a}$ from time to frequency domain.

Power spectrogram

It is important to note that the coefficient matrix $\mathbf{C}$ may be complex valued. To obtain useful metrics, we need to extract some physical quantity from the coefficients. This is where **Parseval's theorem** is used, which relates time and frequency domain components in DFT as follows [4] :

$$\|\mathbf{c}\|^2 \propto \|\mathbf{a}\|^2 \tag{2.5}$$

If $\mathbf{a}$ represents amplitude in the time-domain, then we know that the energy of a wave is related to it's amplitude as,

$$Energy \propto amplitude^2 \tag{2.6}$$

Thus, it can be inferred that **square** of the Fourier coefficients is proportional to the energy distributed in the corresponding frequencies. This spectrogram with squared coefficients is called the **Power Spectrum (P)**. It is often motivating to use this representation because *loudness* is proportional to *energy.*

$$\mathbf{P} = \mathbf{C} \odot \mathbf{C} \tag{2.7}$$

The frequencies in the considered range are grouped into bins. It is useful to do so, due to the aliasing effect of human auditory system. This is motivated by the human cochlea (an organ in the ear) which vibrates at different spots depending on the frequency of the incoming sounds.

Mel Power Spectrogram

The *mel-scale* was developed to express measured frequency in terms of psychological metrics (i.e perceived pitch). The mel-scale was developed by experimenting with the human ears interpretation of a pitch. The experiment showed that the pitch is linearly perceived in the frequency range 0-1000 Hz. Above 1000 Hz, the scale becomes logarithmic. There are several formulae to convert Hertz to mel. The following formula is used in this thesis[1]

$$\omega_m = 2595 log_{10}\left(1 + \frac{\omega}{700}\right) \tag{2.8}$$

Where ω is the frequency in Hertz. In a mel spectrogram, the frequencies and converted to mels and then grouped into mel-spaced bins. This is done by multiplying the spectrum with some **filter bank ($\mathbf{W}_{MEL}$)**. For details about computation of mel-filter banks, refer [11]. Each filter bank is centered at a specific frequency. Hence, to compute R mel bins, we need R mel-filter banks. The resulting mel power spectrogram ($\mathbf{X}$) is

$$\mathbf{X} = \mathbf{W}_{MEL}\mathbf{P} \tag{2.9}$$

$$\mathbf{W}_{MEL} \in \mathbb{R}^{R\times M}, \mathbf{P} \in \mathbb{R}^{M\times P}, \mathbf{X} \in \mathbb{R}^{R\times P}$$

The above Matrix-Matrix multiplication can be represented as a convolution with window size and stride equal to M. We can re-write equation 2.9 as,

$$\mathbf{X}[p, \omega_m] = \sum_{k=p.M}^{p.M+K} \mathbf{p}[k]\mathbf{W}_{MEL}(k - p.M) \tag{2.10}$$

Where:

$\mathbf{p}[k] = \mathbf{P}[i, j]$; $i = floor(\frac{k}{M})$; $j = k - floor(\frac{Mk}{M-1})$

$\omega = k - p.M \in \mathbb{R}^M$

$\mathbf{X} \in \mathbb{R}^{M \times P}$

$\mathbf{p} \in \mathbb{R}^{M.P}$

$\omega_m \in \mathbb{R}^R$

Hence, we can represent mel-power spectrogram as **M-strided convolution** over *flattened* $\mathbf{P}$ with mel filters $\mathbf{W}_{MEL}$ (i.e, the frequency axis of $\mathbf{P}$ is contracted with each mel-filter)

$$\boxed{\mathbf{X} = \mathbf{P} \star \mathbf{W}_{MEL}} \tag{2.11}$$

Thus the extraction of mel-power spectrogram can be summarized in the following algorithm

Algorithm 1 $\mathbf{X} = R_{(MEL)}(\mathbf{a})$

```
   Input : $\mathbf{a} \in \mathbb{R}^N$
   Output : $\mathbf{X} \in \mathbb{R}^{R \times P}$
1: $\mathbf{C} = \mathbf{a} \star \mathbf{W}_{STFT}$
2: $\mathbf{P} = \mathbf{C} \odot \mathbf{C}$
3: $\mathbf{X} = \mathbf{P} \star \mathbf{W}_{MEL}$
```

2.2 Dimensionality Reduction

The objective of dimensionality reduction is to retain only the information that contribute to discrimination and discard the rest. This is done because the *representation* ($\mathbf{X}$) can be large for longer audio tracks (because number of frames P depends on length of the audio). In this thesis, only *mel-spectrogram representation* is considered. The dimensionality reduction of input signal $\mathbf{a}$ is generalized as follows,

$$\begin{aligned} \mathbf{X} &= R(\mathbf{a}) \qquad R : \mathbb{R}^N \rightarrow \mathbb{R}^{R \times P} \\ \mathbf{Y} &= D(\mathbf{X}) \qquad D : \mathbb{R}^{R \times P} \rightarrow \mathbb{R}^{T \times W} \end{aligned} \tag{2.12}$$

The computation of mel-spectrogram defined in the previous section can be a part of the function R. The dimensionality reduction is defined by function D. The output of reduction $\mathbf{Y} \in \mathbb{R}^{T \times W}$ will be the reduced representation ($T < R$ or $W < P$). Depending on how the function D is defined, the following three methods will be discussed

- **Principal Component Analysis** (PCA) : Reduction by *unsupervised learning.*
- **Mel-Frequency Cepstrum Coefficients** (MFCC) : Reduction by domain engineering.
- **Convolution Neural Networks** (CNN) : Reduction by *supervised learning*

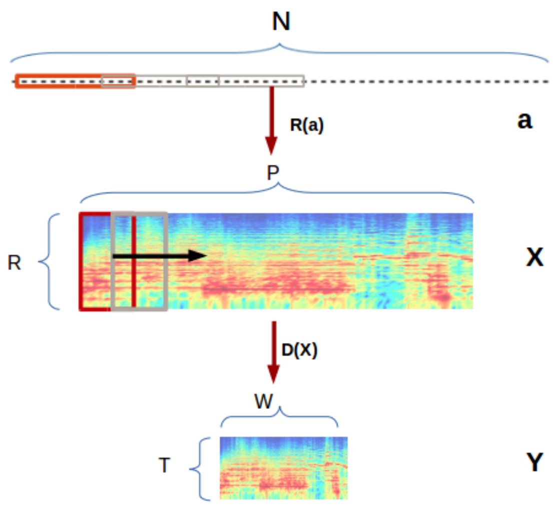


Figure 2.2: Dimensionality Reduction Pipeline

2.2.1 Domain engineering Vs Unsupervised learning Vs Supervised learning

Reduction by Domain engineering : When the operators performing reduction *do not* use any information from the data, but are rather derived from knowing the domain specific properties of the data, then the resulting reduction is said to be *engineered.* Coming up with such operators is usually time-consuming and requires expert knowledge.

Reduction by Unsupervised learning : When the operators performing reduction are computed by exploiting the representation *structure* of the data, then the resulting reductions are said to be *learned* from the data. When this learning problem *does not* require any labelled data, then the reduction is said to be *unsupervised.*

Reduction by Supervised learning : When the operators performing reduction computed by exploiting the information from labelled data, then the resulting reduction is said to be obtained by *supervised learning*

2.2.2 Principal Component Analysis (PCA)

The representation $\mathbf{X}$ is changed to a *basis* that are functions of variance between the frequency components. This is done by computing the co-variance matrix $\mathbf{\Sigma}$ from the data and performing *orthogonal decomposition* to compute it's basis. The co-ordinates of the resulting basis system are called *principal components*. The key idea for reduction is to discard the information corresponding to *low variance* basis. The computation of PCA reduction operator $\mathbf{W}_{PCA}$ is elaborated below,

(a) Usually, large samples (say S samples) of representations from the dataset ($\mathbf{S} = [\mathbf{X}_1, \mathbf{X}_2, ..., \mathbf{X}_S]$) are used to compute the co-variance matrix. The columns of $\mathbf{S}$ are centred by their mean and the covariance matrix is computed as,

$$\mathbf{\Sigma} = \mathbf{E}[(\mathbf{S} - \mathbf{E}[\mathbf{S}])(\mathbf{S} - \mathbf{E}[\mathbf{S}])^T] = \frac{1}{\sum_s P_s} \hat{\mathbf{S}}\hat{\mathbf{S}}^T \quad \in \mathbb{S}^{R\times R}$$

The covariance matrix $\mathbf{\Sigma}$ is symmetric positive definite and hence belongs to space of symmetric operators $\mathbb{S}$.

(b) The eigen values and eigen vectors of $\mathbf{\Sigma}$ are computed. At this point, we use the Orthogonal Eigenvector Decomposition Theorem and infer that eigen vectors of symmetric matrix ($\mathbf{\Sigma}$) form an orthogonal basis in $\mathbb{R}^R$.

$$\mathbf{\Sigma} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^T \qquad \mathbf{V} \in \mathbb{O}^{R\times R}, \quad \mathbf{\Lambda} \in \mathbb{D}^{R\times R}$$

The columns of matrix $\mathbf{V}$ form the basis. Since the columns are orthogonal to each other, $\mathbf{V}$ belongs to a space of orthogonal operators $\mathbb{O}$. $\mathbf{\Lambda}$ is a diagonal matrix of eigen values.

(c) The eigen values represent the magnitude of variance for each frequency dimension. Hence, eigenvectors corresponding to large eigen values gives the coordinates corresponding to greater variance. So eigen vectors corresponding to top T eigen values are retained, while ignoring coordinates of lower variance. The resulting change of coordinates matrix is then $\hat{\mathbf{V}} \in \mathbb{O}^{R\times T}$

(d) Since $\hat{\mathbf{V}}$ is orthogonal, $\hat{\mathbf{V}}^{-1} = \hat{\mathbf{V}}^T$, and the resulting reduction for *each sample* can computed as $\mathbf{Y} = \hat{\mathbf{V}}^T\mathbf{X}$, where $\mathbf{X} \in \mathbb{R}^{R\times P}$, $\mathbf{Y} \in \mathbb{R}^{T\times P}$ and $T < R$. Thus the reduction operator is

$$\mathbf{W}_{PCA} = \hat{\mathbf{V}}^T$$

Algorithm 2 $\mathbf{W}_{PCA} = \text{PCA}(\mathbf{X}_1, \mathbf{X}_2, .., \mathbf{X}_S)$

```
    Input : $\mathbf{S} = [\mathbf{X}_1, ..\mathbf{X}_S]$    $\mathbf{X}_s \in \mathbb{R}^{R \times P_s}, \mathbf{S} \in \mathbb{R}^{R \times Q}$,    $Q = \sum_s P_s$
    Output : $\mathbf{W}_{PCA} \in \mathbb{R}^{T \times R}$
1: $\mathbf{\Sigma} = \frac{1}{Q}\mathbf{S}\mathbf{S}^T$    ▷ $\mathbf{\Sigma} \in \mathbb{S}^{R \times R}$
2: $\mathbf{V}^T \mathbf{\Lambda} \mathbf{V} = \mathbf{\Sigma}$    ▷ $\mathbf{\Lambda} \in \mathbb{D}^{R \times R}$ , $\mathbf{V} \in \mathbb{O}^{R \times R}$
3: $\mathbf{V} \leftarrow \mathbf{V}[:][: T]$    ▷ $\mathbf{V} \in \mathbb{O}^{R \times T}$
4: $\mathbf{W}_{PCA} = \mathbf{V}^T$
```

Since the operator $\mathbf{W}_{PCA}$ is computed from the data without requiring labelling, this method falls under *unsupervised learning.* The columns of representation are first centred before applying the PCA reduction operation. An illustration of PCA reduction function $D_{(PCA)}$ is shown below,

Algorithm 3 $\mathbf{Y} = D_{(PCA)}(\mathbf{X})$

```
    Input : $\mathbf{X} \in \mathbb{R}^{R \times P}$
    Output : $\mathbf{Y} \in \mathbb{R}^{T \times P}$
1: $\hat{\mathbf{X}} = \mathbf{X} - \mathbf{E}[\mathbf{X}]$
2: $\mathbf{Y} = \mathbf{W}_{PCA}\hat{\mathbf{X}}$
```

Sometimes, to make the resulting reduction $\mathbf{Y}$ uncorrelated (the resulting transformation should have identity co-variance matrix), the corresponding dimensions are divided by their eigen values. This is because eigen values are proportional to the magnitude of variance in each frequency direction. This operation is known as **PCA Whitening** and the reduction operator is,

$$\mathbf{W}_{PCAW} = \mathbf{\Lambda}^{-1}\mathbf{V}^T$$

The reduction function $D_{(PCAW)}$ is same as algorithm 3, except that the operator $\mathbf{W}_{PCAW}$ is used instead of $\mathbf{W}_{PCA}$

2.2.3 Mel-frequency cepstrum coefficients (MFCC)

It has been studied that the basis functions resulting from co-variance matrix of log-mel spectrogram representation are similar to cosine transform on log-mel spectrogram[28]. Therefore, instead of explicitly computing the basis functions from the data, one can simple use cosine basis. The coordinates of corresponding basis system are known as *Mel-Frequency Cepstrum Coefficients.* The coordinates of high frequency cosine functions are discarded because they correspond to low-variance information.

Since this reduction is engineered only for log mel spectrogram representation, the reduction function $MFCC$ takes the signal as input. $\mathbf{W}_{MFCC}$ is the reduction operator.

Algorithm 4 $\mathbf{Y} = MFCC(\mathbf{a})$

```
    Input : $\mathbf{a} \in \mathbb{R}^N$
    Output : $\mathbf{Y} \in \mathbb{R}^{T \times P}$
1: $\mathbf{X} = R_{(MEL)}(\mathbf{a})$                                  ▷ $\mathbf{X} \in \mathbb{R}^{R \times P}$
2: $\mathbf{X} \leftarrow ln(\mathbf{X})$
3: for $\omega \in \{1, .., T\}$ do
4:     $\mathbf{W}_{MFCC}[\omega] \leftarrow \mathbf{cos}(\omega \mathbf{t})$        ▷ $\mathbf{W}_{MFCC} \in \mathbb{R}^{R \times T}, \mathbf{t} \in \mathbb{R}^R$
5: end for
6: $\mathbf{Y} \leftarrow \mathbf{W}_{MFCC}\mathbf{X}$
```

2.2.4 Convolution neural network

Transformation of input **a** by shifted contractions of an operator **w** is termed as *discrete convolution* and can be mathematically represented as one dimensional convolution operation,

$$\mathbf{y} = \mathbf{a} \star \mathbf{w}^{(s)}$$

The operator **w** is known as a *filter function* and the length of shift s is known as *stride*. Usually **a** is convolved with multiple filter functions. For K *filters*,

$$\mathbf{Y}[k] = \mathbf{a} \star \mathbf{w}_k{}^{(s)} = \mathbf{a} \star \mathbf{W}^{(s)} \qquad k \in 0, 1..K-1$$

The index based notations for this operation are shown in appendix A.2.1. If the filters $\mathbf{w}_k$ are *defined*, then computation of **Y** is straight forward. All the operations explained in the previous section are forward convolutions with defined filters,

- **STFT** in equation 2.4, where the filter functions are complex negative exponentials.
- Transformation to **Mel** frequency scale in equation 2.11, where a set of mel-frequency centred filters are used.
- **Principal components** and **MFCC** can be realized as a convolution of the input with transformation matrix $\mathbf{V}^T$ defined in section 2.2.2 (Recall that matrix multiplication can be represented as a convolution of stride equal to column length. An illustration was shown in section 2.1.3)

However, it is not clear if such defined filters really encodes the information in **Y** relevant for certain context-based classification task. But, when a set of task-specific observations are available, it is possible to solve for the filters $\mathbf{w}_k$, so that the resulting transformation **Y** is optimal for the considered task. This is done using iterative optimization techniques, starting by random initializing of $\mathbf{w}_k$ and updating it's values after every iteration. Computational models that solves by **first-order gradient descent** methods (a class of optimization techniques) are represented as first order **artificial neural networks**. The filters $\mathbf{w}_k$ which we are attempting to solve are also termed as *parameters* or *weights* of the network. The iterative steps involving finding these *weights* is termed as *training* the neural network (Details of training neural network are explained in section 2.5). Since the transformation operation by the *filters* are represented as convolution over the input, the neural network is termed as **convolution neural network**.

Approximating MFCC with CNN

Supervised feature learning has an advantage over *unsupervised* feature learning when we wish to find filters $\mathbf{w}_k$ optimal for context based classification tasks. Feature learning with CNN fall under *supervised* feature learning category. To show that CNN can do atleast as good as MFCCs, an illustration is discussed by approximating MFCC with CNN. Setting up a CNN architecture requires defining the *hyper parameters* of like number of filters, filter dimensions and stride of the filter. The domain knowledge of MFCC computation is used to set these hyper parameters.

To compute MFCC features, the input signal $\mathbf{a}$ is convolved with complex negative exponentials ($\mathbf{W}_{STFT}$). After element-wise squaring operation, the resulting transformation is convolved with mel-filters ($\mathbf{W}_{MEL}$). Logarithm of the resulting transformation is then convolved with cosine filters ($\mathbf{W}_{MFCC}$). The motivation and description of these so called *engineered* filters were described in section 2.1.2 and 2.2.

Algorithm 5 $\mathbf{Y} = \text{MFCC}(\mathbf{a})$

```
   Input : $\mathbf{a} \in \mathbb{R}^N$
   Output : $\mathbf{Y} \in \mathbb{R}^{T \times Q}$
1: $\mathbf{C} = \mathbf{a} \star \mathbf{W}_{STFT}{}^{(s)}$        $\triangleright\ \mathbf{W}_{STFT} \in \mathbb{R}^{M \times F}, \mathbf{C} \in \mathbb{C}^{M \times Q}$
2: $\mathbf{C} \leftarrow \mathbf{C} \odot \mathbf{C}$
3: $\mathbf{X} = \mathbf{C} \star \mathbf{W}_{MEL}{}^{(M)}$        $\triangleright\ \mathbf{W}_{MEL} \in \mathbb{R}^{R \times M}, \mathbf{X} \in \mathbb{R}^{R \times Q}$
4: $\mathbf{X} \leftarrow ln(\mathbf{X})$
5: $\mathbf{Y} = \mathbf{X} \star \mathbf{W}_{MFCC}{}^{(R)}$        $\triangleright\ \mathbf{W}_{MFCC} \in \mathbb{R}^{T \times R}$
```

An equivalent of MFCC computation can be realized with three layers of convolution by replacing the engineered filters by learnable filters and setting the following hyper parameters :

- $\mathbf{W}_{L1}$: M filters (representing discrete frequencies) of size F (STFT window size) and stride s (STFT hop length)
- $\mathbf{W}_{L2}$: R filters (for mel-frequencies) of size M and stride M.
- $\mathbf{W}_{L3}$: T filters (for mel coefficients) of size R and stride R.

The non-linearity in between each layer is needed. Otherwise, the filters accross layers can be combined into a representation for single layer. Setting $\mathbf{\Phi}_1$ as element-wise squaring operation and $\mathbf{\Phi}_2$ as logarithm should result in a CNN architecture that *may* realize MFCCs. That is, the solution will converge to the defined filters ($\mathbf{W}_{STFT}, \mathbf{W}_{MEL}, \mathbf{W}_{MFCC}$) if they are optimal for the task considered. Otherwise, one could expect either richer representation or sub-optimal representation because of local convergence.

Algorithm 6 $\mathbf{Y} = \text{CNN}(\mathbf{a})$

```
    Input : $\mathbf{a} \in \mathbb{R}^N$
    Output : $\mathbf{Y} \in \mathbb{R}^{T\times Q}$
1: $\mathbf{C} = \mathbf{\Phi}_1(\mathbf{a} \star \mathbf{W}_{L1}{}^{(s)})$        $\triangleright\ \mathbf{W}_{L1} \in \mathbb{R}^{M\times F}, \mathbf{C} \in \mathbb{C}^{M\times Q}$
2: $\mathbf{X} = \mathbf{\Phi}_2(\mathbf{C} \star \mathbf{W}_{L2}{}^{(M)})$        $\triangleright\ \mathbf{W}_{L2} \in \mathbb{R}^{R\times M}, \mathbf{X} \in \mathbb{R}^{R\times Q}$
3: $\mathbf{Y} = \mathbf{\Phi}_3(\mathbf{X} \star \mathbf{W}_{L3}{}^{(R)})$        $\triangleright\ \mathbf{W}_{L3} \in \mathbb{R}^{T\times R}$
```

It is possible to with-hold the engineered filters at earlier levels and just introduce learning at later stages. For instance, it is sometimes optimal to compute the STFT and mel-filters and just perform convolutions over the mel-spectrogram[18][22]. Sometimes it is also useful to generalize the convolution as a 2 dimensional operation[26].(see Chapter 3, Sec. 3.1.4). 2D convolution operation is shown in appendix A.2.2.

CNN as a general purpose feature extractor

In music information retrieval, several task-specific features have been engineered. The MFCC features along with it's derivatives (the derivative is useful to encode the temporal evolution) are often used for genre and mood recognition tasks. Instead of convolving with mel-filters on STFT representation, one might opt for filters that would result in chroma-gram (combines frequencies by exploiting the periodicity of pitches) or tempo-gram (encodes change of frequencies over time). Features derived from chroma-gram find application for automatic mixing, chord recognition tasks. Features derived from tempo-gram find applications for tasks like on-set detection, tempo-estimation etc.. But more often, combination of features are used to boost classifier performance. For all these features that can be realized in terms of hierarchy of convolution operations, it is possible to find mathematical equivalence in convolution neural network.

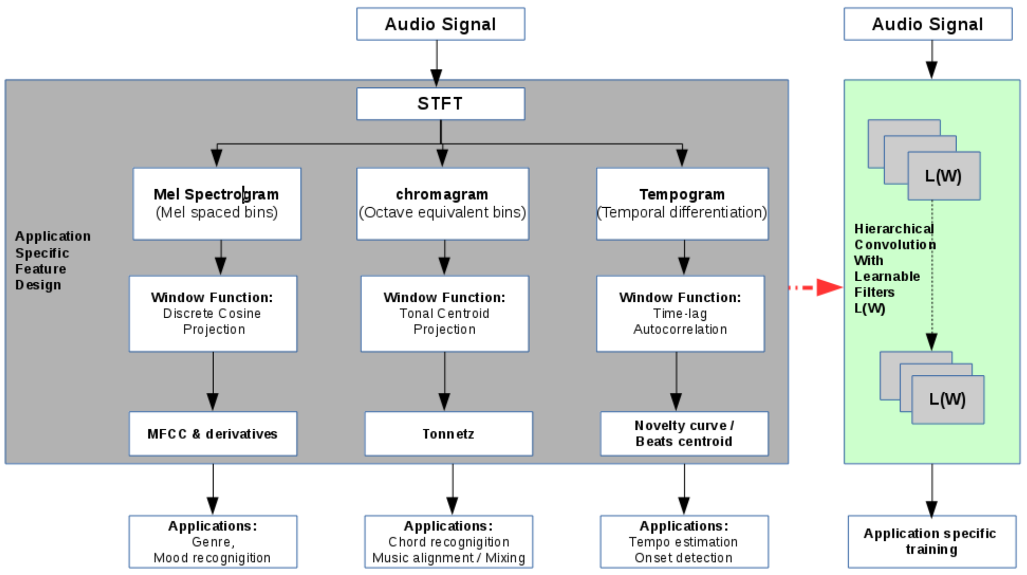

Audio Signal
STFT
Mel Spectrogram (Mel spaced bins)
chromagram (Octave equivalent bins)
Tempogram (Temporal differentiation)
Application Specific Feature Design
Window Function: Discrete Cosine Projection
Window Function: Tonal Centroid Projection
Window Function: Time-lag Autocorrelation
MFCC & derivatives
Tonnetz
Novelty curve / Beats centroid
Applications: Genre, Mood recognigition
Applications: Chord recognigition Music alignment / Mixing
Applications: Tempo estimation Onset detection
Audio Signal
L(W)
Hierarchical Convolution With Learnable Filters L(W)
L(W)
Application specific training

Figure 2.3: General purpose feature extractor

2.3 Temporal Approximation

The size of resulting features computed through reduction operations discussed in the previous section depends on length of the audio. But classifiers like *support vector machines* and *multi layer perceptron* requires features of fixed size to compute the parameters while training. Therefore, frame-wise features over time are approximated to a fixed size feature representation. Temporal approximation T of the reduced representation $\mathbf{Y}$ is

$$\mathbf{f} = T(\mathbf{Y}) \qquad \mathbf{f} \in \mathbb{R}^Z, \mathbf{Y} \in \mathbb{R}^{T \times W}$$

W depends on length of the audio. $\mathbf{f}$ is usually the final fixed size feature that is given as input to the classifier.

The transformation by approximation function T is done by using one of the methods,

- *unsupervised* methods like *statistical pooling* or *clustering techniques* (Bag Of Frames, Gaussian mixture models)
- *supervised* methods with *recurrent neural network*.

2.3.1 Bag Of Frames

In Bag of Frames(BoF) model[5], the *frequency* of each reduced frame is used as a feature for training a classifier. To reduce to a fixed size representation (of size Z), Z cluster centres are computed from the data. Each frame-wise feature is assigned to it's nearest cluster. The number of assignments to each cluster now becomes the feature $\mathbf{f}$.

Algorithm 7 $\mathbf{f}$ = BagOfFrames($\mathbf{Y}$)

```
    Input : $\mathbf{Y} \in \mathbb{R}^{T \times W}$
    Output : $\mathbf{f} \in \mathbb{R}^{Z}$
 1: $\mathbf{f} = 0$
 2: for $i \in \{0, 1, .., W-1\}$ do
 3:     $j = \arg\min_{j} \|\mathbf{Y}[:, i] - \boldsymbol{\mu}_j\|^2$        ▷ $\boldsymbol{\mu}_j \in \mathbb{R}^T, j \in \{0, 1..Z-1\}$
 4:     $\mathbf{f}[j] \leftarrow \mathbf{f}[j] + 1$
 5: end for
```

The cluster centres are computed by clustering algorithms likes gaussian mixture models or K-Means clustering. Computation of $\boldsymbol{\mu}_j$ by K-Means is illustrated in algorithm 8. The cluster centres are computed for the entire dataset. Each of the N sample track could have W frame-wise features. To compute these centres, $\boldsymbol{\mu}_j$ are first randomly initialized. Then each frame level feature is assigned to the nearest centre. After the assignment of all frame features, the cluster centres $\boldsymbol{\mu}_j$ are re-computed. The frame features are then re-assigned to these new centres. This re-assignment and re-computation of $\boldsymbol{\mu}_j$ is iterated until convergence.

Algorithm 8 K-MEANS($\mathbf{Y}_0, \mathbf{Y}_2, ..., \mathbf{Y}_{N-1}$)

```
    Input : $\mathbf{Y}_n \in \mathbb{R}^{T \times W}, n \in \{0, 1, ..., N-1\}$
    Output : $\boldsymbol{\mu}_j \in \mathbb{R}^{T}, j \in \{0, 1, .., Z-1\}$
 1: Randomly initialize $\boldsymbol{\mu}_j$
 2: while $\boldsymbol{\mu}_j$ have not converged do
 3:     for $n \in \{0, 1, .., N-1$ do
 4:         for $i \in \{0, 1, .., W-1\}$ do
 5:             $j = \arg\min_{j} \|\mathbf{Y}_n[:, i] - \boldsymbol{\mu}_j\|^2$
 6:             Assign $\mathbf{Y}_n[:, i]$ to cluster $j$
 7:         end for
 8:     end for
 9:     Recompute cluster means $\boldsymbol{\mu}_j$
10: end while
```

2.3.2 Recurrent Neural Networks

The idea behind RNN is to learn a feature representation by sequentially combining the input into an internal state $\mathbf{h}$. The resulting feature ($\mathbf{f}$) is a projection from this internal state.

$$\mathbf{f} = \mathbf{W}\mathbf{h}_W \qquad \mathbf{W} \in \mathbb{R}^{Z \times K}, \mathbf{h}_W \in \mathbb{R}^{K}$$

$$\mathbf{h}_W = \boldsymbol{\Theta}(\mathbf{Y}[:, W], \mathbf{h}_{W-1}) \qquad \mathbf{Y} \in \mathbb{R}^{T \times W}$$

$\mathbf{h}_W$ is the internal state after combining W columns of $\mathbf{Y}$. $\mathbf{\Theta}$ is the internal state function which sequentially combines the input. The operator $\mathbf{W}$ and other operators resulting from the function $\mathbf{\Theta}$ are solved for optimality using a labelled dataset. These operators can be computed by training an RNN (see Section 2.5). The function $\mathbf{\Theta}$ should sufficiently hold the information from beginning to end of the sequence. RNNs face challenge in remembering the information in the earlier part of the sequence. Because of this, different flavours of RNN have been developed to hold the sequence information longer and to battle the vanishing gradient problem while training neural network (see Section 2.5). Depending on how $\mathbf{\Theta}$ is defined, at least two RNN architectures are popular in the literature: Long-Short Term Memory (LSTM) RNN and Gated Recurrent Unit(GRU) RNN. In this thesis, only LSTMs[2] are considered.

Long-Short Term Memory RNN

The LSTM internal state is contained by three gates that control the information flow. This is done by multiplying the output of each sequence $\mathbf{o}_w$ with the corresponding cell state function $\mathbf{c}_w$.

$$\mathbf{h}_w = \mathbf{o}_w \odot \sigma_h(\mathbf{c}_w) \qquad w \in \{1, 2, ..., W\}$$

σ_h is hyperbolic tangent function which projects the values between -1 and 1. The output gate $\mathbf{o}_w$ combines the w^{th} vector in sequence with the previous internal state ($\mathbf{h}_{w-1}$) and projects the result between 0 and 1 with sigmoid activation (σ). This indicates the contribution of current sequence w to the cell state.

$$\mathbf{o}_w = \sigma(\mathbf{W}_o\mathbf{Y}[:, w] + \mathbf{U}_o\mathbf{h}_{w-1})$$

The cell state $\mathbf{c}_w$ acts as a conveyor belt where the information can either flow unchanged or get modified with update ($\mathbf{i}_w$) and forget functions ($\mathbf{g}_w$).

$$\mathbf{c}_w = \mathbf{g}_w \odot \mathbf{c}_{w-1} + \mathbf{i}_w \odot \sigma_h(\mathbf{W}_c\mathbf{Y}[:, w] + \mathbf{U}_c\mathbf{h}_{w-1})$$

The operators of forget gate $\mathbf{W}_g$ and $\mathbf{U}_g$ control the deletion of information from the *previous* sequence. The output of g_w is between 0 (delete the information) and 1 (keep the information).

$$\mathbf{g}_w = \sigma(\mathbf{W}_g\mathbf{Y}[:, w] + \mathbf{U}_g\mathbf{h}_{w-1})$$

The operators of update gate $\mathbf{W}_i$ and $\mathbf{U}_i$ control the addition of information from the *current* sequence. The output of i_w is also between 0 and 1.

$$\mathbf{i}_w = \sigma(\mathbf{W}_i\mathbf{Y}[:, w] + \mathbf{U}_i\mathbf{h}_{w-1})$$

The operators $\mathbf{W}_o, \mathbf{U}_o, \mathbf{W}_i, \mathbf{U}_i, \mathbf{W}_g, \mathbf{U}_g, \mathbf{W}_c$ and $\mathbf{U}_c$ are solved by training the RNN.

To get a fixed size temporal approximation, the RNN should project all the input sequence to a single output. This architecture of RNN is called *Sequence to One* RNN.

Algorithm 9 $\mathbf{f} = Seq2One\_LSTM(\mathbf{Y})$

```
    Input : $\mathbf{Y} \in \mathbb{R}^{T \times W}$
    Output : $\mathbf{f} \in \mathbb{R}^{Z}$
1: Initialize $\mathbf{h}_0$
2: for $w \in \{1, 2, .., W\}$ do
3:     $\mathbf{h}_w \leftarrow \mathbf{\Theta}_{LSTM}(\mathbf{Y}[:, w], \mathbf{h}_{w-1})$
4: end for
5: $\mathbf{f} = \mathbf{W}\mathbf{h}_W$
```

Often times, multiple layers of RNN are used. An illustration of two layer LSTM with a *sequence to sequence* LSTM in between is shown in algorithm 10. The input sequence transformed in to a hidden sequence, which is then projected to a single output by the second layer. The functions in between (Φ_1, Φ_2) represents the transition operation between the layers. Several non-linear activations and transition operations have been developed to address the problems while training deep neural network (see Sec. 2.5).

Algorithm 10 $\mathbf{f} = LSTM2(\mathbf{Y})$

```
    Input : $\mathbf{Y} \in \mathbb{R}^{T \times W}$
    Output : $\mathbf{f} \in \mathbb{R}^{Z}$
1: $\mathbf{F}_1 = \Phi_1(Seq2Seq\_LSTM(\mathbf{Y}))$
2: $\mathbf{f} = \Phi_2(Seq2One\_LSTM(\mathbf{F}_1)$
```

Algorithm 11 $\mathbf{F} = Seq2Seq\_LSTM(\mathbf{Y})$

```
    Input : $\mathbf{Y} \in \mathbb{R}^{T \times W}$
    Output : $\mathbf{F} \in \mathbb{R}^{Z \times W}$
1: Initialize $\mathbf{h}_0$
2: for $w \in \{1, 2, .., W\}$ do
3:     $\mathbf{h}_w \leftarrow \mathbf{\Theta}_{LSTM}(\mathbf{Y}[:, w], \mathbf{h}_{w-1})$
4:     $\mathbf{F}[:, w] = \mathbf{W}\mathbf{h}_W$
5: end for
```

2.4 Multi-label Classifier

The classifier takes the feature $\mathbf{f}$ as input and performs the classification task. A *discriminative* [1] binary classifier can be formalised as,

$$\ell_i = b(\zeta_i) = \begin{cases} 1, & \text{if } \zeta_i > \epsilon \\ 0, & \text{otherwise} \end{cases} \qquad i \in \{1, 2, .., L\}, \ell_i \in \{0, 1\}, \mathbf{f} \in \mathbb{R}^Z$$

where b is a binary classifier, given the feature vector $\mathbf{f}$. The output of a binary classifier ℓ_i is either 0 or 1. There are L binary classifier outputs for L labels. ζ_i is the i^{th} output of the classification function C and the classifier output ℓ_i is 1 if ζ_i is greater than certain threshold ϵ

$$\boldsymbol{\zeta} = C(\mathbf{f}) \qquad \boldsymbol{\zeta} \in \mathbb{R}^L$$

The final prediction (**pred**) is an index set of all classifier outputs that is equal to 1

$$\mathbf{pred} = \{\ell_i | \ell_i = 1\}$$

[1] The classification model can either be *generative* (approximates class distribution) or *discriminative* (approximates class boundaries). For a binary classifier, the classes are either 0 or 1

The following equivalent notation will be used in algorithms of upcoming chapters

$$\mathbf{pred}_{(\epsilon)} = \{b(\zeta_i) | b(\zeta_i) = 1\} \qquad i \in \{1, 2, .., L\}$$

The classification function C projects the feature $\mathbf{f}$ to the vector $\boldsymbol{\zeta}$ in the label space. Depending on how this function is defined, there are several discriminative classifiers like *multi-layer perceptrons*, *support vector machines*, *random-forest*. In this thesis, only *multi-layer perceptron* is considered.

2.4.1 Two-layer perceptron

A two layer perceptron first projects the features $\mathbf{f}$ to a hidden layer $\mathbf{h}$ and some element-wise non-linear operation Φ is applied.

$$\mathbf{h} = \Phi(\mathbf{W}_{L1}\mathbf{f}) \qquad \mathbf{h} \in \mathbb{R}^{L1}, \mathbf{W}_{L1} \in \mathbb{R}^{L1 \times Z}$$

Without the non-linear operation Φ, the operators $\mathbf{W}_{L1}$ and $\mathbf{W}$ can multiply out to generalize a single layer perceptron. The motivation for using a two layer perceptron is because single layer perceptron can approximate only linear class boundries. But *universal approximation theorem*[31] states that a single hidden layer with some non-linear activation function can approximate any function. The choice of Φ will be explained in section 2.5. The second operator $\mathbf{W}$ performs the projection from hidden vector $\mathbf{h}$

$$\boldsymbol{\zeta} = \sigma(\mathbf{Wh}) \qquad \boldsymbol{\zeta} \in \mathbb{R}^{L}, \mathbf{W} \in \mathbb{R}^{L \times L1}$$

where σ is a sigmoid function, which projects the output of second layer between 0 and 1.

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.13}$$

Thus $0 \leq \zeta_i \leq 1$ and threshold ϵ can be set in-between 0 and 1 and eventually the final prediction **pred** can be computed.

2.5 Training

The iterative steps involving the computation of the operators that is optimal for transformations for our context-based classification task is called *training*. When we train only the operators of classifying function C (that is, $\mathbf{W}$ and $\mathbf{W}_{L1}$), the classification performance will only be as good as the information encoded in the feature $\mathbf{f}$. Let us assume that the features $\mathbf{f}$ obtained as a result of transformations R, D, and T is optimal for the task at hand. Hence, only the classifier C is trained. The abstract prediction model is shown in algorithm 12.

Algorithm 12 **pred** $= Model(\mathbf{a})$

```
    Input : $\mathbf{a} \in \mathbb{R}^N$
    Output : pred                                   ▷ indices of predicted labels
1:  $\mathbf{X} = R(\mathbf{a})$                                ▷ $\mathbf{X} \in \mathbb{R}^{R \times P}$
2:  $\mathbf{Y} = D(\mathbf{X})$                                ▷ $\mathbf{Y} \in \mathbb{R}^{T \times W}$
3:  $\mathbf{f} = T(\mathbf{Y})$                                ▷ $\mathbf{f} \in \mathbb{R}^{Z}$
4:  $\boldsymbol{\zeta} = C(\mathbf{f} \quad |\mathbf{W}, \mathbf{W}_{L1})$                  ▷ $\boldsymbol{\zeta} \in \mathbb{R}^{L}$
5:  $\mathbf{pred} = \{b(\zeta_i) | b(\zeta_i) = 1\}$          ▷ $i \in \{1, 2, .., L\}, b(\zeta_i) \in \{0, 1\}$
```

Considering a two layer perceptron for classification, $\boldsymbol{\zeta}$ is computed as,

$$\boldsymbol{\zeta} = \sigma(\mathbf{W}\Phi(\mathbf{W}_{L1}\mathbf{f}))$$

Computing $\mathbf{W}$ and $\mathbf{W}_{L1}$ is an inverse problem. That is, we need a target $\mathbf{t}$ that approximates $\boldsymbol{\zeta}$ for true classifications from a set of observations. To do this, recall that $\boldsymbol{\zeta}$ is a L dimensional vector and L is the number of labels in consideration. ζ_i is the classifier output for i^{th} label and $i \in \{1, 2, .., L\}$. With sigmoid (σ) projection we know that $0 \leq \zeta_i \leq 1$. If we assume t_i equal to 1 for true classifications and 0 for the rest, then the transformation operators ($\mathbf{W}$ and $\mathbf{W}_{L1}$) can be solved such that the classifier output ζ_i is close to 1 for true classifications.

2.5.1 First-order gradient descent

Solving for the operators by *first-order gradient descent* involves the following steps :

1. Initialize $\mathbf{W}, \mathbf{W}_{L1}$
2. Run the model for a sample and compute loss $E = loss(\boldsymbol{\zeta}, \mathbf{t})$
3. Compute the gradient of loss with respect to the operators $(\frac{\partial E}{\partial \mathbf{W}}, \frac{\partial E}{\partial \mathbf{W}_{L1}})$
4. Update $\mathbf{W}$ and $\mathbf{W}_{L1}$
5. Recompute loss, gradients and update the parameters until convergence.

Loss function

To train a classifier, a loss function (or error function) have to be defined. The *least square* error function is sensitive to outliers in the training data. Hence *cross-entropy*[25] loss function is considered. Error function defined as the negative log-likelihood is the *cross-entropy* error function. *Likelihood* that the parameters $(\mathbf{W}, \mathbf{W}_{L1})$ approximate a set of targets for label i for N training samples $({t_i}^1, {t_i}^2, ..., {t_i}^N)$ is

$$\mathcal{L}(\mathbf{W}, \mathbf{W}_{L1} | {t_i}^1, {t_i}^2, ..., {t_i}^N) = \prod_{n=1}^{N} {\zeta_{i(n)}}^{{t_i}^{(n)}} (1 - \zeta_{i(n)})^{1 - {t_i}^{(n)}} \qquad t_i \in \{0, 1\}$$

where the *likelihood* is 1, as the classifier output ζ_i approaches the target t_i. The log-likelihood is taken to get rid of the multiplication that would cause numerical problems over large N. The negative of the log-likelihood is taken to pose the optimization as a *minimization* problem. Therefore, minimizing the *cross-entropy* loss for label i is equivalent to minimizing the *negative log likelihood*

$$\text{Minimize } E_i = -ln(\mathcal{L}) = -\sum_{n=1}^{N}\{t_i{}^{(n)} ln\zeta_{i(n)} + (1 - t_i{}^{(n)})ln(1 - \zeta_{i(n)})\}$$

For L labels, the total loss is minimized,

$$\text{Minimize } \sum_{i=1}^{L} E_i$$

Moreover, with gradient descent optimization, the loss is minimized for a batch of samples (B) for every iteration. B is called the *batch-size.* Thus, loss for every iteration would be,

$$E_i{}^{(B)} = -\sum_{i=1}^{L}\sum_{n=1}^{B}\{t_i{}^{(n)} ln\zeta_{i(n)} + (1 - t_i{}^{(n)})ln(1 - \zeta_{i(n)})\}$$

Computing gradients

After every iteration, the gradient of the total loss (E) with respect to the parameters ($\mathbf{W}, \mathbf{W}_{L1}$) have to computed. This gradient will then be used to update the parameters for next iteration. The gradients are computed by applying the chain rule. An illustration for computing the gradients for the two layer perceptron is shown

$$\frac{\partial E}{\partial \mathbf{W}} = \frac{\partial E}{\partial \boldsymbol{\zeta}}\frac{\partial \boldsymbol{\zeta}}{\partial \sigma}\frac{\partial \sigma}{\partial \mathbf{W}} \tag{2.14}$$

$$\frac{\partial E}{\partial \mathbf{W}_{L1}} = \frac{\partial E}{\partial \boldsymbol{\zeta}}\frac{\partial \boldsymbol{\zeta}}{\partial \sigma}\frac{\partial \sigma}{\partial \Phi}\frac{\partial \Phi}{\partial \mathbf{W}_{L1}} \tag{2.15}$$

Efficient computation of the gradients is achieved with *back-propagation* algorithm. To ease the computations, the non-linearities and loss functions are usually chosen in such a way that the variables required for gradient computation are known in the forward pass.

Updating the parameters

Standard stochastic gradient descent update for every iteration is,

$$\mathbf{W} = \mathbf{W} - \eta\frac{\partial E}{\partial \mathbf{W}}$$

where η is the learning rate. But choosing a proper learning rate is difficult because if η is too small, convergence can be too slow, while η that is too high can hinder convergence. Additionally, the same learning rate is applied to all parameters across the layers. Moreover, the neural networks lead to highly non convex optimization problem and to avoid getting trapped in a local optima is challenging. Therefore, several optimization algorithms specialized for neural network training emerged to deal with the aforementioned challenges. In this thesis, Adaptive Moment (ADAM) Estimation updates[19] will be used. ADAM uses the exponentially decaying average of past moments (first moment m and second moment v) and squared gradients (g) to update the parameters. Update for iteration t is,

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1)g_t$$

$$v_t = \beta_2 v_{t-1} + (1-\beta_2){g_t}^2$$

As m_t and v_t are initialized as vectors of 0's, the authors of ADAM observe that they are biased towards zero, especially during the initial time steps, and especially when the decay rates are small (i.e. β_1 and β_2 are close to 1). They counteract these biases by computing bias-corrected first and second moment estimates[27]:

$$m_t = \frac{m_t}{1-{\beta_1}^t}$$

$$v_t = \frac{v_t}{1-{\beta_2}^t}$$

Thus, update of each parameter for the next iteration is,

$$W_{t+1} = W_t - \frac{\eta}{\sqrt{v_t}+\epsilon} m_t$$

The authors propose default hyper-parameter values of 0.9 for β_1, 0.999 for β_2, and 10^{-8} for ϵ

2.5.2 Deep learning

As mentioned before, if we train only the operators of classifying function C, then the classification performance will only be as good as the information encoded in the feature **f**. But, if it is possible to solve for the transformation operators that compute the features **f**, then it is possible to obtain features optimal for the considered task. That is, if we push the supervision into the temporal approximator function T with a *sequence to one* LSTM, then the classifier performance is no longer limited by the encodings in the feature **f**. Because, now the operators of RNN can be solved for optimality in addition to the operators of 2 layer perceptron, and therefore **f** can be optimal for the task. Now we have to update every iteration not just $\mathbf{W}, \mathbf{W}_{L1}$, but also $\mathbf{W}_o, \mathbf{U}_o, \mathbf{W}_i, \mathbf{U}_i, \mathbf{W}_g, \mathbf{U}_g, \mathbf{W}_c$ and $\mathbf{U}_c$

Algorithm 13 $\mathbf{pred} = Model(\mathbf{a})$

```
   Input : $\mathbf{a} \in \mathbb{R}^N$
   Output : pred                                        ▷ indices of predicted labels
1: $\mathbf{X} = R(\mathbf{a})$                          ▷ $\mathbf{X} \in \mathbb{R}^{R\times P}$
2: $\mathbf{Y} = D(\mathbf{X})$                          ▷ $\mathbf{Y} \in \mathbb{R}^{T\times W}$
3: $\mathbf{f} = Seq2One\_LSTM(\mathbf{Y} \quad | \mathbf{W}_k, \mathbf{U}_k)$   ▷ $k = \{i_r, o, g, c\}, \mathbf{f} \in \mathbb{R}^Z$
4: $\boldsymbol{\zeta} = C(\mathbf{f} \quad | \mathbf{W}, \mathbf{W}_{L1})$       ▷ $\boldsymbol{\zeta} \in \mathbb{R}^L$
5: $\mathbf{pred} = \{b(\zeta_i) | b(\zeta_i) = 1\}$    ▷ $i \in \{1, 2, .., L\}, b(\zeta_i) \in \{0, 1\}$
```

However, the performance is still limited by the frame-wise features **Y**. But the supervision can be further pushed inside by using convolution neural networks and solving for it's operators ($\mathbf{W}_{C1}, \mathbf{W}_{C2}, \mathbf{W}_{C3}$) in addition to the operators of RNN and perceptron.

Algorithm 14 **pred** $= Model(\mathbf{a})$

```
    Input : $\mathbf{a} \in \mathbb{R}^N$
    Output : pred                                        ▷ indices of predicted labels
1: $\mathbf{X} = R(\mathbf{a})$                                                   ▷ $\mathbf{X} \in \mathbb{R}^{R \times P}$
2: $\mathbf{Y} = CNN(\mathbf{X} \quad | \mathbf{W}_{C1}, \mathbf{W}_{C2}, \mathbf{W}_{C3})$                 ▷ $\mathbf{Y} \in \mathbb{R}^{T \times W}$
3: $\mathbf{f} = Seq2One\_LSTM(\mathbf{Y} \quad | \mathbf{W}_k, \mathbf{U}_k)$             ▷ $k = \{i_r, o, g, c\}, \mathbf{f} \in \mathbb{R}^Z$
4: $\boldsymbol{\zeta} = C(\mathbf{f} \quad | \mathbf{W}, \mathbf{W}_{L1})$                              ▷ $\boldsymbol{\zeta} \in \mathbb{R}^L$
5: $\mathbf{pred} = \{b(\zeta_i) | b(\zeta_i) = 1\}$            ▷ $i \in \{1, 2, .., L\}, b(\zeta_i) \in \{0, 1\}$
```

From the illustration in algorithm 6, it can be seen that the learning problem can pushed up to the point of even replacing the $STFT$ operators. That is, the model can be now trained to detect the pattern directly from the raw audio signal for our classification task. Since the training data is now able to affect the operators of feature computations, the context of learning problem is now called *deep learning*

Issues

As exciting as it may sound, deep learning has two major issues,

Requires large training data :

Looking at all the application domains where deep learning is successful (image /speech recognition), they are the ones where acquiring a lot of data is feasible. As the number of parameters to optimize increases, not only that more iterations are needed to converge, but there is also a risk that the solutions converge to learning the noise in data. This is called *over-fitting.*

Transfer-learning : This is one way to address this issue for smaller datasets. *Transfer learning* is possible only when an alternate large dataset for similar task (*source task*) is available. It is then possible to train with the smaller dataset by initializing the weights with the values converged in the source task. This is called *fine-tuning* the model.

Drop-out : This is a regularizer that counter over fitting by randomly setting parameters to zero. This is called *dropping* the connection. At each training iteration, the connections can be *dropped out* with probability p. ($Drop_{(p)}$)

Vanishing gradients :

As the number of layers in the neural network increases, the risk of gradients approaching zero in the earlier layers increases. This will increase the number of iterations required for convergence. As an illustration, looking at the gradient computation equations for the final layer in equation 2.14 and the penultimate layer in equation 2.15, it can be seen that as we move deeper, the number of multiplications in the chain rule required for calculating the gradient increases. If one of those gradients in chain have a value close to zero, then the gradient with respect to the parameters will also be close to zero. Thus, the non-linearities Φ are chosen such that the gradient is boosted. The non-linearities - Rectified linear units ($ReLU$)[12] and Exponential linear units (ELU)[21] have been used in this thesis.

$$ReLU(x) = max(0, x)$$

$$ELU(x) = \begin{cases} x, & \text{if } x \geq 0 \\ a(e^x - 1), & \text{otherwise} \end{cases}$$

where $a > 0$ is a hyper-parameter.

Chapter 3

Literature Survey and Model Selection

Using content-based music information for solving several music information retrieval tasks is not new, but a decade long research efforts have been put. Hunting for the right model for our task and to justify it to be superior to the rest requires thorough understanding of evolution of such techniques. In section 3.1, the dynamics of the literature that has lead to the use of deep learning techniques for MIR tasks have been discussed. In section 3.2, the inferences from state of art techniques have been used to short list models for the experiments.

3.1 Literature Review

A number of surveys[15][6] amply document what is a decades-long research effort at the intersection of music, machine learning and signal processing. In a broader sense, all techniques have a two-stage architecture: first, features are extracted from music audio signals to transform them into a more meaningful representation. These features are then used as input to a classifier, which is trained to perform the task at hand. This dedicated analysis for music features emerged due to the fact that music signals possess specific acoustic and structural characteristics that distinguish them from spoken language or other non musical signals.

In a general sense, the goal of classification tasks in music informatics is to attach a semantic meaning to the content. The underlying issue is ultimately one of *organization* and *variance* of the features.

Feature organization : The better organized a feature is to answer some question, the simpler it is to assign or infer semantic meaning. Thus a feature representation should explicitly reflect a desired semantic organization. That is, the information about the discriminants (refered in **??**) should not be lost.
Feature variance : A feature representation is said to be *noisy* when variance in the data is misleading or uninformative, and *robust* when it predictably encodes these invariant attributes.

Complicated classifying methods are necessary only to compensate for any noise and hence a *robust* feature representation is important.

In subsection 3.1.1, some of the early works indicating the need for better *feature organization* are discussed. In the remaining subsections, adoption of feature learning techniques for multi-label classification task are elaborated. The general motivation of all the works from section 3.1.2 - 3.1.4, was to use feature learning to obtain *robust* and *organized* features. (In section 2.2.4, how feature learning can increase *robustness* was explained) All the models (except [26]) were experimented on Magna Tag a Tune dataset(MTT)[8] with about 29K clips which are 29.1s long.

3.1.1 From classifier to feature emphasis

Looking back to our history before 2010, there is a clear trend in MIR of applying increasingly more powerful machine learning algorithms to the same feature representations to solve a given task. There are also ample surveys with evidence suggesting that appropriate feature representations significantly reduce the need for complex semantic interpretation methods[3]. Particularly in [7], ten different classifiers were compared on same set of features for genre classification task. It was seen that ceiling performance of 80% was achieved on GTZAN dataset, thereby suggesting the need for robust feature representation for further improvements. In [10], significant improvements were reported even for simplest classifiers by using appropriate filtering of features for chord recognition task.

3.1.2 From hand-crafting to feature learning

Great amount of technical research have been done to develop robust features. A number of features relevant for different MIR tasks were formulated (hand-crafted) and tested. MFCC features (ref. **??**), originally developed for speech recognition task often proved efficient for genre classification and tagging. At the same time, some machine learning algorithms were also used to adopt feature learning on spectrogram frames. Several subsequent works rely on a Bag of Frames approach - where a collection of features are computed for each frame and then statistically aggregated (ref. 2.3.1). Some early feature learning approaches that proved more efficient than MFCCs are discussed in this section.

Temporal pooling and multiscale learning for automatic annotation and ranking of music audio. 2011 [14]:

The pipe line of their algorithm is shown below. The formalism of the notations used are consistent with explanations in chapter 2. The PCA whitened mel-power spectrogram (ref. **??**) is compared with MFCC features on Magna tag a tune dataset. It was shown that the former achieve a performance of **AUC 0.87** out performing MFCCs which was 0.77.

Signal (**a**) is sampled at 22.1 KHz. Then STFT with window length 1024 and stride 512 is computed with FFT algorithm (ref. 2.1.3). This is followed by conversion to mel power-spectrogram with 128 bins, followed by PCA Whitening which selects the top 120 variant frequencies. Another transformation is done by stacking a single layer perceptron ($L(\mathbf{W})$ means the weight matrix $\mathbf{W}$

is learned by training a neural network). The temporal pooling is done by summarizing every 2.3s frame with suitable functions (see [14] for details). The matrix $\mathbf{W}_1$ learns the optimal features for pooling. The resulting feature is then classified by two layer perceptron with 1000 hidden units with sigmoid (σ) activations.

Algorithm 15 $Pred = \text{MODEL}(\mathbf{a})$

```
    Input : $\mathbf{a} \in \mathbb{R}^N$
    Output : $Pred \in \mathbb{R}^L$
 1: $\mathbf{C} = STFT(\mathbf{a})$                          ▷ $\mathbf{C} \in \mathbb{C}^{M \times P}$
 2: $\mathbf{Y}_r = \mathbf{C} \odot \mathbf{C}$              ▷ $\mathbf{Y}_r \in \mathbb{R}^{M \times P}$
 3: $\mathbf{R} = MEL(\mathbf{Y}_r)$                        ▷ $\mathbf{R} \in \mathbb{R}^{128 \times P}$
 4: $\mathbf{X}_1 = PCA\_WHITEN(\mathbf{R})$                 ▷ $\mathbf{X}_1 \in \mathbb{R}^{120 \times P}$
 5: $\mathbf{X}_2 = L(\mathbf{W}_1)\mathbf{X}_1$             ▷ $\mathbf{W}_1 \in \mathbb{R}^{S \times 120}, \mathbf{X}_2 \in \mathbb{R}^{S \times P}$
 6: $\mathbf{y} = POOL(\mathbf{X}_2)$                        ▷ $\mathbf{y} \in \mathbb{R}^{S.W}$
 7: $Pred = \sigma(L(\mathbf{W}_3)\sigma(L(\mathbf{W}_2)\mathbf{y}))$   ▷ $\mathbf{W}_2 \in \mathbb{R}^{1000 \times S.W}, \mathbf{W}_3 \in \mathbb{R}^{L \times 1000}$
```

It is important to note that this algorithm is does not work on audio of arbitrary length because of their design of temporal pooling (because fixed sized features are needed for classification).

Multiscale Approaches To Music Audio Feature Learning. 2012[17]:

The result reported by this model is the current state-of-art on MTT dataset (**AUC 0.898**). Here, features are extracted from mel-power spectrogram by convolving with window functions (*gaussian pyramids*). This is done for W window functions of different sizes (time length). The resulting features are then concatenated. PCA whitened frames in the mel-spectrogram are subjected to unsupervised learning with K-Means to get the Bag of Frames features (see 2.3.1). The efficient performance attributed to use of window functions of different time length suggests the existence of overlapping rhythms. (Recall the discussion about rhythmic traces in **??**)

Algorithm 16 $Pred = \text{MODEL}(\mathbf{a})$

```
    Input : $\mathbf{a} \in \mathbb{R}^N$
    Output : $Pred \in \mathbb{R}^L$
 1: $\mathbf{C} = STFT(\mathbf{a})$                          ▷ $\mathbf{C} \in \mathbb{C}^{M \times P}$
 2: $\mathbf{Y}_r = \mathbf{C} \odot \mathbf{C}$              ▷ $\mathbf{Y}_r \in \mathbb{R}^{M \times P}$
 3: $\mathbf{R} = MEL(\mathbf{Y}_r)$                        ▷ $\mathbf{R} \in \mathbb{R}^{R \times P}$
 4: for $i \in \{1, .., W\}$ do
 5:     $\mathbf{X}_1 \leftarrow GAUSSIAN\_PYRAMID(\mathbf{R}, i)$         ▷ $\mathbf{X}_1 \in \mathbb{R}^{R \times Q1_i}$
 6:     $\mathbf{X}_2 \leftarrow PCA\_WHITEN(\mathbf{X}_1)$               ▷ $\mathbf{X}_2 \in \mathbb{R}^{S1 \times Q1_i}$
 7:     $\mathbf{X}_3 \leftarrow BAG\_OF\_FRAMES(\mathbf{X}_2, S2)$       ▷ $\mathbf{X}_3 \in \mathbb{R}^{S2 \times Q2_i}$
 8:     $\mathbf{Y}[i] \leftarrow MAX\_POOL(\mathbf{X}_3)$               ▷ $\mathbf{Y}[i] \in \mathbb{R}^{S2}, \mathbf{Y} \in \mathbb{R}^{S2 \times W}$
 9: end for
10: $\mathbf{y} = FLATTEN(\mathbf{Y})$                       ▷ $\mathbf{y} \in \mathbb{R}^{S2.W}$
11: $Pred = \sigma(L(\mathbf{W}_3)ReLU(L(\mathbf{W}_2)\mathbf{y}))$   ▷ $\mathbf{W}_2 \in \mathbb{R}^{1000 \times S2.W}, \mathbf{W}_3 \in \mathbb{R}^{L \times 1000}$
```

The take-away is that modelling relation between features at rhythmic intervals does help.

3.1.3 Transfer Learning by supervised pre-training

Stacked feature learning techniques typically require large amounts of training data to work well. But sometimes, features learned on large datasets can be used for other datasets, either as a *black-box* extractor (feature extractor is not further trained on target dataset) or as a *fine-tuned* feature extractor (feature extractor is further trained after initializing weights). To do this, it is essential to have a source task that requires a very rich feature representation, so as to ensure that the information content of this representation is likely to be useful for other tasks

Transfer learning by supervised pre-training for audio-based music classification. 2014[20]:

In this research, features trained on MSD dataset (1000K clips) are used as *black-box* feature extractor while training on MTT dataset and the resulting classification performance still achieved **AUC 0.88** outperforming baseline MFCC. The workflow for source and target are shown below,

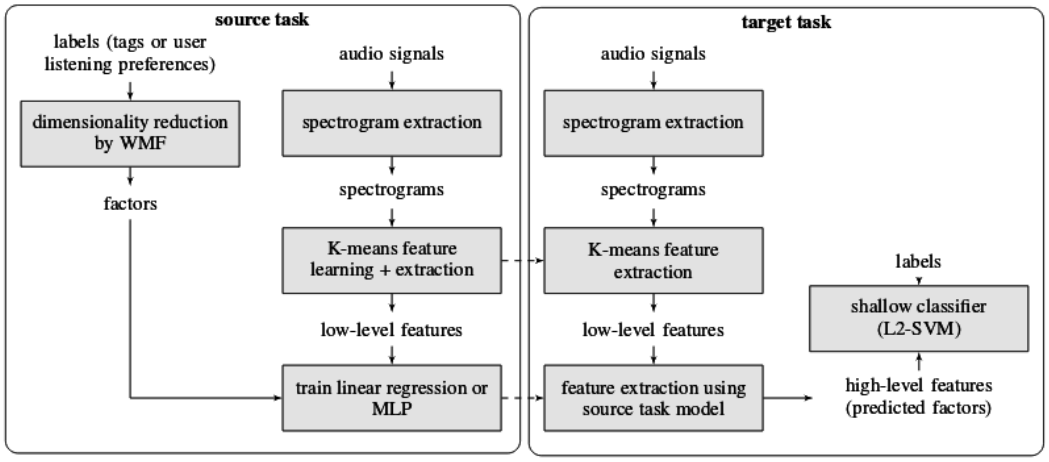

source task
labels (tags or user listening preferences)
dimensionality reduction by WMF
factors
audio signals
spectrogram extraction
spectrograms
K-means feature learning + extraction
low-level features
train linear regression or MLP
target task
audio signals
spectrogram extraction
spectrograms
K-means feature extraction
low-level features
feature extraction using source task model
labels
shallow classifier (L2-SVM)
high-level features (predicted factors)

Figure 3.1: Schematic overview of the workflow of transfer learning[20]

Source task: The low-level features from audio spectrograms are learned through unsupervised learning by spherical K-Means. A multi layer perceptron is then stacked to obtain final prediction. So the output from the penultimate layer of MLP are treated as transferable features. To tackle problems created by redundant and sparse labels, dimensionality reduction is done in the label space using PCA. The model is then trained to predict the reduced label representation.

Target task Next, the trained models are used to extract features from other datasets, which are then passed to train shallow classifiers for different but related target tasks. This workflow is visualized in figure 3.1. Dashed arrows indicate transfer of the learned feature extractors from the source task to the target task.

3.1.4 Convolutional Neural Networks

It can be seen that deep signal processing structures can be realized by stacking multiple shallow architectures (ref. 2.2.4). As feature learning was proving to be more efficient than hand crafted features, stacking learnable layers over one another became a hot area of research. The idea was to replace the application specific dimension reductions with hierarchy of learnable convolution filters.

End-to-end learning for music audio. 2014[18]:

As shown in chapter 2, all operations including FFT can be defined in terms of convolutions. In this research they investigate whether it is possible to apply feature learning directly to raw audio signal. The signal was convolved with 3 layers of 1D convolutions followed by two fully connected layers. Thus, the feature and the classifier was trained in a single pipeline and this is called *end to end learning*. They compared the *end to end learning* approach with convolutions from mel-spectrogram on MTT dataset (i.e, retaining STFT). Their algorithm is described below. Function f is an element-wise logarithmic compression. It was found that, discarding STFT hurt the performance. CNN from mel-spectrogram achieved **AUC 0.8815**, but on including convolutions on audio signal AUC dropped to 0.8487.

Algorithm 17 CNN(raw audio) [**0.84**]

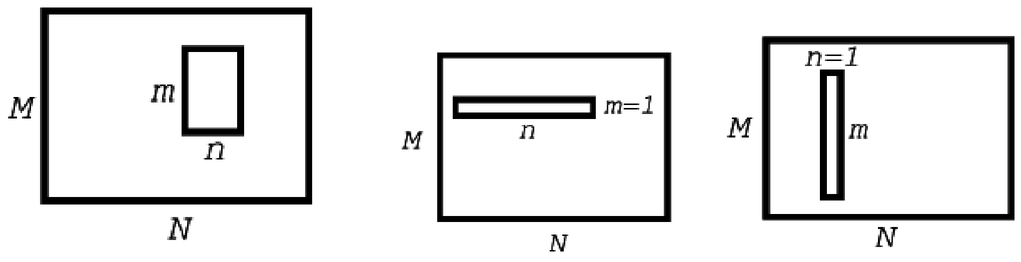
```
    Input : $\mathbf{a} \in \mathbb{R}^N$
    Output : $Pred \in \mathbb{R}^L$
1:  $\mathbf{C}_1 = f(\mathbf{a} \star \mathbf{w}^{(256)}_{(256)})$
2:  $\mathbf{C}_2 = MaxPool(ReLU(\mathbf{C}_1 \star \mathbf{w}^{(1)}_{(32)}))$
3:  $\mathbf{C}_3 = MaxPool(ReLU(\mathbf{C}_2 \star \mathbf{w}^{(1)}_{(32)}))$
4:  $\mathbf{y} = FLATTEN(\mathbf{C}_3)$
5:  $Pred = \sigma(L(\mathbf{W}_2)ReLU(L(\mathbf{W}_1)\mathbf{y}))$
```

Algorithm 18 CNN(Mel-Spectrogram) [**0.88**]

```
    Input : $\mathbf{a} \in \mathbb{R}^N$
    Output : $Pred \in \mathbb{R}^L$
1:  $\mathbf{R} = f(MEL(||STFT(\mathbf{a})||^2))$
2:  $\mathbf{C}_1 = MaxPool(ReLU(\mathbf{R} \star \mathbf{w}^{(1)}_{(32)}))$
3:  $\mathbf{C}_2 = MaxPool(ReLU(\mathbf{C}_1 \star \mathbf{w}^{(1)}_{(32)}))$
4:  $\mathbf{y} = FLATTEN(\mathbf{C}_2)$
5:  $Pred = \sigma(L(\mathbf{W}_2)ReLU(L(\mathbf{W}_1)\mathbf{y}))$
```

Experimenting with musically motivated convolutional neural networks. 2016[26]:

In the previous section, only 1D convolution with filter sizes directly motivated by hand-crafted methods were tested for comparison. But usually, the convolution operation allows flexibility in choosing the filter sizes. In this research, the authors discuss how convolution filters with different shapes can fit specific musical concepts.



(a) Rectangular filter (b) Time filter (c) Frequency filter

Figure 3.2: Different Filter sizes

Time filters can learn temporal cues (Onset, BPM and other rhythmic patterns), while *frequency filters* can differentiate timbre and note. *Rectangular filters* can learn short time sub-bands (Bass, kick, drums)[26]. However, because of hierarchical nature of deep networks, any filter should be theoretically capable of picking up the relevant cues. It was shown in their experiments that *rectangular filters* or combination of time and frequency filters performed better than using just time / frequency filter. These experiments were however done for a genre classification task.

Automatic tagging using deep convolutional neural networks. 2016[22]:

Different CNN architectures were tested and the proposed model achieves close to state of art performance on MTT dataset (**0.894 AUC**). The audio samples were down-sampled to 12 KHz and convolutions were started from mel spectrogram (96 bins). They also compared MFCCs, convolutions over STFT and convolutions over Mel-log power spectrogram and report that the latter performs significantly better.

Model	**AUC**
STFT $\rightarrow$ CNN	0.846
STFT $\rightarrow$ MEL $\rightarrow$ CNN	**0.894**
STFT $\rightarrow$ MEL $\rightarrow$ MFCC	0.862

Also, to exploit the advantage of PCA Whitening proven in [17][14], Batch Normalization of frequency components is done. That is, data is centred to the batch mean and divided by batch variance. In Batch normalization however, the basis is not switched but the data is *learned* to be scaled and shifted.

Algorithm 19 $\hat{\mathbf{X}} = \text{BATCHNORM}(\mathbf{X})$

```
    Input : $\mathbf{X} \in \mathbb{R}^{B \times S \times Q}$,                    ▷ $B$ is batch size
    Output : $\hat{\mathbf{X}} \in \mathbb{R}^{B \times S \times Q}$
    Parameters to learn : $\gamma$ (Scale), $\beta$ (Shift)
1:  $\boldsymbol{\mu}, \boldsymbol{\sigma}^2 = FREQUENCY\_MEAN\_VARIANCE(\mathbf{X})$      ▷ $\boldsymbol{\mu}, \boldsymbol{\sigma}^2 \in \mathbb{R}^S$
2:  for $i \in \{1, .., B\}$ do
3:      for $j \in \{1, .., Q\}$ do
4:          $\mathbf{X}[i, :, j] \leftarrow \frac{\mathbf{X}[i,:,j] - \boldsymbol{\mu}}{\sqrt{\boldsymbol{\sigma}^2} - \epsilon}$
5:      end for
6:  end for
7:  $\hat{\mathbf{X}} = \gamma \mathbf{X} + \beta$
```

The five layer proposed CNN architecture is shown below. The filters **W** in each layer are the weights that will be learned. *Spatial_Bn* is similar to the normalization algorithm mentioned above, except that the normalization is done along the 1st axis of tensors **C**. $MaxPool_{i,j}$ is a dimensionality reduction done by pooling (i, j) elements along S and Q directions respectively. *Elu* is an element-wise non-linearity operation described in section 2.5

Algorithm 20 $\mathbf{y} = CHOI\_CNN(\mathbf{R})$

```
    Input : $\mathbf{R} \in \mathbb{R}^{1 \times 96 \times 1366}$
    Output : $\mathbf{y} \in \mathbb{R}^{1024}$
1:  $\mathbf{R}_n = BatchNorm(\mathbf{R})$
2:  $\mathbf{C}_1 = \mathbf{R}_n \star \mathbf{W1}^{(1,1)}_{(32)}$            ▷ $\mathbf{W1} \in \mathbb{R}^{32 \times 3 \times 3}, \mathbf{C}_1 \in \mathbb{R}^{32 \times S1 \times Q1}$
3:  $\mathbf{C}_1 \leftarrow MaxPool_{(2,4)}(Elu(Spatial\_Bn(\mathbf{C}_1)))$            ▷ $\mathbf{C}_1 \in \mathbb{R}^{32 \times T1 \times W1}$
4:  $\mathbf{C}_2 = \mathbf{C}_1 \star \mathbf{W2}^{(1,1)}_{(128)}$            ▷ $\mathbf{W2} \in \mathbb{R}^{128 \times 3 \times 3}, \mathbf{C}_2 \in \mathbb{R}^{128 \times S2 \times Q2}$
5:  $\mathbf{C}_2 \leftarrow MaxPool_{(2,4)}(Elu(Spatial\_Bn(\mathbf{C}_2)))$            ▷ $\mathbf{C}_2 \in \mathbb{R}^{128 \times T2 \times W2}$
6:  $\mathbf{C}_3 = \mathbf{C}_2 \star \mathbf{W3}^{(1,1)}_{(128)}$            ▷ $\mathbf{W3} \in \mathbb{R}^{128 \times 3 \times 3}, \mathbf{C}_3 \in \mathbb{R}^{128 \times S3 \times Q3}$
7:  $\mathbf{C}_3 \leftarrow MaxPool_{(2,4)}(Elu(Spatial\_Bn(\mathbf{C}_3)))$            ▷ $\mathbf{C}_3 \in \mathbb{R}^{128 \times T3 \times W3}$
8:  $\mathbf{C}_4 = \mathbf{C}_3 \star \mathbf{W4}^{(1,1)}_{(192)}$            ▷ $\mathbf{W4} \in \mathbb{R}^{192 \times 3 \times 3}, \mathbf{C}_4 \in \mathbb{R}^{192 \times S4 \times Q4}$
9:  $\mathbf{C}_4 \leftarrow MaxPool_{(2,4)}(Elu(Spatial\_Bn(\mathbf{C}_4)))$            ▷ $\mathbf{C}_4 \in \mathbb{R}^{192 \times T4 \times W4}$
10: $\mathbf{C}_5 = \mathbf{C}_4 \star \mathbf{W5}^{(1,1)}_{(256)}$            ▷ $\mathbf{W5} \in \mathbb{R}^{256 \times 3 \times 3}, \mathbf{C}_5 \in \mathbb{R}^{256 \times S5 \times Q5}$
11: $\mathbf{C}_5 \leftarrow Elu(Spatial\_Bn(\mathbf{C}_5))$
12: $\mathbf{y} = Flatten(\mathbf{C}_5)$            ▷ $\mathbf{y} \in \mathbb{R}^{1024}$
```

The features from convolutions then pass through a fully connected layer of size equalling number of tags. The authors have then trained this model on MSD dataset and made the weights publicly available.

3.2 Model Selection

In section 3.1, it was stated that when the feature is well *organized* and encodes the *variance* in the data, it becomes easier to attach a semantic meaning. Feature learning can increase robustness, but to learn an organized representation is not guaranteed. That is to say, the extracted feature should encode the information about it's discriminants related to the task. Our brain differentiates sounds with energy changes, and this is approximated by MFCCs (ref **??**) through proportionate energy variance from a mel-spaced spectrogram (ref 2.1.3). But in section **??**, it was argued that the difference between music and speech is that, a music signal is composed of several superimposed *rhythmic traces*. It was not clear if the classifier could decompose the rhythms from engineered features and hence there was this movement from feature engineering to feature learning. The results from the work [17], where fully unsupervised technique is adopted for feature learning, shows that hand-crafted features does lose some information necessary for classification. But even the learned features were extracted from mel-spaced frequency spectrogram that does not exploit the harmonic encodings. That is, we still do not know if the learning algorithms extract the rhythms, thereby questioning the optimality of *feature organization* (i.e, would the features learned for one task be optimal for auxiliary but related task?). However in general, it could be seen that feature learning performs better than MFCCs for multi-label classification task.

Music tagging problem is further complicated by the complexity of semantic assignments that reflect user preference. To get the right discriminants, the training method should be properly defined in the first place. In section 1.1.2, the problems with content based methods that stem from training assumptions were pointed out. One of which was the social-factor assumption resulting from training on large datasets. But we want train on a specialized dataset which is small. This leaves us with the question, if the models trained by supervised learning on larger datasets[22][18] can be used for smaller datasets with different label-context. Secondly, the assumptions resulting by training on short excerpts of music rather than whole song cause vagueness. This is because, the currently available large datasets only contain short clips and the current algorithms generalize the tags for the whole song by merging tags from short sections of the song. Therefore, methods that hold better feature organization for songs of arbitrary length are also explored.

3.2.1 Transfer learning Vs MFCC

To check if models trained on large datasets can be exploited for smaller datasets, *transfer learning* from the model which achieves state of art performance with CNN[22] is compared with MFCC features. (It makes perfect sense to compare the state of art unsupervised feature learning algorithm[17] as well, but in this thesis I stick to analysing CNNs). This will also show if features learned (stacked convolutions - ref. 2.2.4) through supervised training on large dataset attain better *feature organization* than MFCCs. It is important to note that, better *feature organization* simply does not mean that classifier identifies the *rhythmic traces*.

3.2.2 Bag Of Frames vs RNN

To summarize tags for songs of arbitrary length, most of the current algorithms classify short sections of the spectrogram separately and finally merge tags across different sections[18]. It is also possible to stack a *decision tree* over section-wise tags and improve performance, but that would not tell anything about the optimality of *feature organization*. Hence for temporal pooling, only methods that directly work on content information are considered. Algorithm in [17] is designed to handle songs of arbitrary length and it was seen that Bag of Frames features trained using K-Means algorithm (see 2.3.1) proved efficient while testing on 29.1s excerpts from MTT dataset. However, it is not clear if these features are optimal choice for identifying rhythms. It is also not known if the efficiency of K-Means will be retrained when tested on songs longer than 30s. Hence, this algorithm is compared with temporal approximation using Recurrent Neural Network (see 2.3.2). Supervised training with RNN might force the classifier to look for rhythmic content.

Chapter 4

Experiments and Results

The aim of this thesis is to find the optimal algorithm for content-based multi-label classification of music tracks, that can be solved with minimal training data. We are looking for a classifier that can discriminate aesthetics in music, but the public datasets are socially biased and contains only short excerpts. Hence the multi-label classifiers have to be tested on our unbiased target dataset which has full clips. In Chapter 3, the state of art models were reviewed and in section 3.2, the short-comings of these algorithms in addressing the problems discussed in Chapter 1 (see 1.1.2) were pointed out. In this chapter, the experiments that will lead to finding the best algorithm using the components short-listed in 3.2 will be described.

4.1 Dataset and Evaluation

More specific to our task than representing audio is finding a proper dataset of labelled pairs. Recalling that our aim is to identify the aesthetic properties of music from the audio content, a dataset that is free from *audio-semantic* noise is needed. Furthermore, to test *transfer learning*, a large dataset is needed for the *source task* (ref. 3.2.1). Popularly used *Million Song Dataset* (MSD) [13] contains a cluster of complimentary datasets, most of them annotated with *social tags*. For instance, *Last.fm* which forms a part of MSD contains annotations from users of an online radio application. But such social tags contribute to the *audio-semantic* noise which we want to eliminate. The dataset that is mostly used for evaluating content-based algorithms is *Magna Tag A Tune* dataset [8], where annotations are gathered through a game application that attracts users who are familiar with technical terms related to music. Hence the tags in this dataset are usually clean. Hence this would be a decent choice for our *source task*.

4.1.1 Dataset for source task

The MagnaTagATune dataset consists of 25,856 clips of 29.1-s, 16 kHz-sampled mp3 files with 188 tags. These annotations are gathered from an online game called *Tag a Tune*. A player is partnered up with another random player who cannot be communicated. Both listen to some track, and have to select appropriate tags. Then the players are asked one simple question : ”Are we listening to same song?”. Answering this correctly will earn them points. This dataset is the largest available that comes close to minimizing the *audio-semantic* noise. However, social factor still plays a role here. This is partly indicated by tag frequency where the most frequent tag is used 4,851 times while the 50^{th} most frequent one used 490 times in the training set. We use only the top-50 tags for training from this dataset.

4.1.2 Dataset for target task

To gather annotations with clean mapping to aesthetic properties, we adopt the most straightforward and costly method - ask someone to listen to songs and tag them. Around 900 songs approximately 5 - 8 min long, were tagged by my supervisor Prof. Paolo Bientinesi in association with Prof. Marco Aluno (Professor of Composition and Theory at University EAFIT, Columbia). Out of 900 songs, 100 are used for validation.

4.1.3 Evaluation metrics

For multi-label classification with L labels, the performance of L binary classifiers is computed and averaged. Each label can belong to one of the class - *positive* (1) or *negative* (0). To discuss about a fair performance measure, lets first recall some important terminologies,

True Positives (TP) : If the classifier admits a label as positive when the ground truth is also positive.
True Negative (TN) : If the classifier admits a label as negative when the ground truth is also negative.
False Positives (FP) : If the classifier admits a label as positive when the ground truth is negative.
False Negative (FN) : If the classifier admits a label as negative when the ground truth is positive.

For a general tagging problem, most of the tags are *negative* for most of the clips (That is, out of 900 songs, if 20 songs have the tag '*electro*', then for this tag there are 20 *ground truth positives* and 880 *ground truth negatives.*

Accuracy : To see why *accuracy* will be an unfair measure, lets look at the definition of *accuracy* for the label 'electro',

$$Accuracy = \frac{\sum TP + \sum TN}{900} = \frac{0 + 880}{900}$$

Even if the classifier did not classify one 'electro' as positive, accuracy will be 0.98 with the contribution from 880 true negative samples.

Precision : This is also not a comprehensive measure to indicate the strength of the classifier because it does not tell anything about the percentage of negatively classified samples (false negatives). That is, even if the classifier *correctly* admits one 'electro' as positive and 899 as negative, *precision* would be 1.0

$$Precission = \frac{\sum TP}{\sum TP + \sum FP} = \frac{1}{1 + 0}$$

Recall : This is also not comprehensive because it does not tell anything about false positives or in other words, it does not say if your classifier is a *liar*. That is, even if the classifier admits 900 samples as positive (20 true positives and 880 false positives), *recall* will be 1.0

$$Recall = \frac{\sum TP}{\sum TP + \sum FN} = \frac{20}{20+0}$$

To strike a balance between *recall* and *precision*, the harmonic mean of both is often used, which is called *F1* score. But to calculate all the metrics mentioned so far, some classifier threshold is required (That is, a binary classifier spits a number between 0 and 1 and if the number is above the threshold, the sample is classified as positive, otherwise negative). If someone changes the threshold then the performance can change. To find a comprehensive measure for classifier performance, the metric should consider all threshold values.

Area under precision-recall curve : When the *precision* and *recall* are plotted for various threshold and the area under the precision-recall curve is found, the metric is termed as *average precision*. Averaging the *average precision* of L labels gives *Mean average precision*, which is a useful metric and can be found in some research works.

Area under receiver operating characteristic curve (AUC) : The *fall-out* and *recall* are plotted for various threshold and the area under the this curve is found. *Fall-out* is defined as

$$Fall\_out = \frac{\sum FP}{\sum FP + \sum TN}$$

Recall answers the question, 'when the ground truth is positive, how often does the classifier admit it as a positive'. *Fall-out* answers the question, 'when the ground truth is negative, how often does the classifier admit it as positive'. A random classifier would have an AUC of 0.5, meaning, for a binary random classification of an unknown sample there is 50% chance for it to be true. Therefore, AUC can be thought of as a probability that a classifier would rank a randomly chosen positive ground truth higher than a randomly chosen negative observation. AUC measures how well the positive and negative classes are separated. AUC is computed for L labels and averaged. Since the publications reviewed in previous chapter report this metric, we will also use the same. But in addition, we use *weighted average* because our validation set is small and number of occurrences of each label is not balanced.

Therefore, in all our experiments, *Weighted averaged AUC* (WAUC) will be reported.

4.2 Experiments

As with any MIR task, the raw audio signal containing amplitude values in time domain is first down sampled and representation parameters are fixed (ref. 2.1.3). This is because computational cost is heavily affected by the size of the input layers.

Sampling Parameters : Although most of the available audio in digital format are sampled at 44KHz (ref. 2.1.1), it is important to note that most of the information lie in the lower range of the spectrum. In [22], a pilot experiment was conducted to demonstrate similar performance with 12KHz and 16KHz for top 50 tags (Recall that MTT clips are already downsampled to 16KHz). Hence we sample all our tracks to 12KHz.

Next step is to extract relevant features. General pipeline is *sampling* (ref. 2.1.1), *representation* (ref. 2.1.3), stacks of *dimensionality reduction* (ref. 2.2) followed by *temporal summarization* (ref. 2.3). Feature learning can be introduced at different stages. Introducing in earlier stages would require training with huge amount of data. Feature learning on raw sampled signal proved sub-optimal in [18]. Same was the case when convolved over STFT frame [22]. Convolutions over mel-spectrogram performed better than MFCC in many previous work [22][18]. Hence we stick to engineered features until the extraction of mel-spectrogram.

Mel-Spectrogram Parameters : The signal in the time domain is converted to frequency domain by Short-Time-Fourier-Transform ($STFT$) using Fast-Fourier-Transform(FFT) algorithm. The arguments for doing this were presented in Chapter 2 (ref. 2.1.3). The parameters for FFT are the size (often referred as FFT Size) and stride (often referred as hop-length) of the window function. FFT size was fixed to 512 (42 ms) and hop-length was fixed to 256. Motivated by the human auditory system, the frequency axis is binned to mel-scale and log of squared STFT coefficients (proportional to loudness) are calculated. In [22], it was stated that 96 mel-bins were optimum.

Feature learning over mel-spectrogram still requires large dataset, but our target dataset is small. Hence by questing the effectiveness of feature learning over spectrogram with MFCCs, we decided to compare both.

4.2.1 Experiments with pre-trained CNNs as feature extractor

In Chapter 2 (ref. 2.2.4), it was shown how Convolution Neural Networks (CNN) are motivating to be used as feature extractor for music signal. In Chapter 3 (ref. 3.1.4) some of the successful mel-spectrogram convolution architectures trained on MTT dataset showing state-of-art performance were discussed. Now we would like to see if such features extracted through these models can be used for auxiliary tasks. That is to say, the learned CNN parameters (weights) from *source* dataset are used as initialization setting for *target* tasks.

The CNN architecture from [22] achieves the best AUC score on MTT dataset and hence this architecture is used for feature extraction. The algorithm for their model ($CHOI\_CNN$) is explained in section 3.1.4 (Algorithm 20). The input to their CNN was 29.1s mel-spectrogram with representation parameters mentioned above (Thus resulting in 1366 time samples). So for our task, features are extracted every 29.1s and sequentially sent to RNN. The temporal summarization is done by 2 layer *Long Short-Term Memory Recurrent Neural Network* (ref. 2.3.2). The RNN module does sequence to one mapping (*Seq2One*) of the input features. This entire model is then trained for 150K iteration on MTT dataset with top 50 tags. Their model was already trained on Million Song Dataset [13] with top 50 *Last.fm* tags. We just fine-tune their model on MTT dataset after merging clips from same song. Features of clips from same song are sequentially given as input to RNN with a dropout (ref. 2.5) of 0.3 after each layer, which then projects to a fixed sized feature vector. The output of RNN is then passed to a fully connected layer with 50 output units and *sigmoid* activation. ADAM optimizer with *binary-cross-entropy* loss function is used for training. The starting learning rate is 0.001, decaying at 1^{-8} and beta 0.99 (ref. 2.5). Algorithm for L labels is described below and the notations used are consistent with formalisms in Chapter 2. The algorithm is implemented in *Torch* [torch] and mel-spectrogram was extracted using *Librosa* [librosa]

Algorithm 21 $Pred = \text{MODEL}(\mathbf{a})$

```
    Input : $\mathbf{a} \in \mathbb{R}^N$
    Output : $Pred \in \mathbb{R}^L$
 1: $\mathbf{C} = STFT(\mathbf{a})$                                   ▷ $\mathbf{C} \in \mathbb{C}^{M \times P}$
 2: $\mathbf{Y}_r = Log(\mathbf{C} \odot \mathbf{C})$                 ▷ $\mathbf{Y}_r \in \mathbb{R}^{M \times P}$
 3: $\mathbf{R} = MEL(\mathbf{Y}_r)$                                  ▷ $\mathbf{R} \in \mathbb{R}^{96 \times P}$
 4: $W = floor(\frac{P}{1366})$
 5: for $i \in \{0, .., W\}$ do
 6:     $\mathbf{X} \leftarrow \mathbf{R}[:][i : (i+1).1366]$        ▷ $\mathbf{X} \in \mathbb{R}^{96 \times 1366}$
 7:     $\mathbf{Y}[i] \leftarrow CHOI\_CNN(\mathbf{X})$              ▷ $\mathbf{Y} \in \mathbb{R}^{1024 \times W}$
 8: end for
 9: $\mathbf{Y}_1 = Drop_{(0.3)}(Seq2Seq\_LSTM(\mathbf{Y}))$          ▷ $\mathbf{Y}_1 \in \mathbb{R}^{1024 \times W}$
10: $\mathbf{y}_2 = Drop_{(0.3)}(Seq2One\_LSTM(\mathbf{Y}_1))$       ▷ $\mathbf{y}_2 \in \mathbb{R}^{1024}$
11: $Pred = \sigma(L(\mathbf{W})\mathbf{y}_2))$                       ▷ $\mathbf{W} \in \mathbb{R}^{L \times 1024}$
```

This CNN model can either be used as a *black-box* feature extractor (That is, weights of the model are not modified while training the *target-task*) or certain layers can be *fine-tuned* (That is, we continue the training on *target-task*). Both the cases are looked separately,

Blackbox CNN + RNN :
The weights of CNN trained on the source task are not modified (no fine-tuning). The weights of RNN are also initiazied with those trained on source task. The fully-connected layer in the source task is changed to 65 output units. (i.e 65 labels). The network is trained by back-propagating through the fully connected layer and RNN with *binary-cross entropy* loss function (ref. 2.5). The optimization parameters are same as that of *source task*. Training is stopped after 25K iterations, after which the model begins to over-fit. Weighted averaged AUC (WAUC) was **0.65**

Fine-tune CNN + RNN :
With the same parameter settings, the last layer of CNN was finetuned after 5K iterations. Training was then continued until 25K iterations and WAUC went up to **0.69**. When last two CNN layers were finetuned WAUC further improved to **0.71**. Fine-tuning earlier layers proved sub-optimal.

This tells us that in the final layers CNN tend to find features specific to task. (Recalling that labels for source and target tasks are different). However, we still do not know if convolutions over mel-spectrogram will be better than MFCCs which is proven to model audio discriminants. Before going there, the effectiveness of RNN should also be questioned. It was hypothesised in Chapter 3 (3.2) that Bag-of-Frames (2.3.1) features using K-Means (which was actually used in [17] to attain state of art performance) may not be suitable for summarizing features for longer audio. So we test this by replacing RNN with Bag Of Frames (BoF) features.

CNN + BoF : The CNN is first finetuned for 10K iterations with algorithm 21. Then 1024 centroids of CNN features are found by unsupervised training on both MTT and our target dataset. This is followed by multi layer perceptron with a hidden layer of 512 units and ReLU activation. Having hidden size of 1024 did not improve the result. WAUC was **0.67**. The algorithm is described below,

Algorithm 22 $Pred = \text{MODEL}(\mathbf{a})$

```
    Input : $\mathbf{a} \in \mathbb{R}^N$
    Output : $Pred \in \mathbb{R}^{65}$
 1: $\mathbf{C} = STFT(\mathbf{a})$                                  ▷ $\mathbf{C} \in \mathbb{C}^{M\times P}$
 2: $\mathbf{Y}_r = Log(\mathbf{C} \odot \mathbf{C})$                  ▷ $\mathbf{Y}_r \in \mathbb{R}^{M\times P}$
 3: $\mathbf{R} = MEL(\mathbf{Y}_r)$                               ▷ $\mathbf{R} \in \mathbb{R}^{96\times P}$
 4: $W = floor(\frac{P}{1366})$
 5: for $i \in \{0, .., W\}$ do
 6:     $\mathbf{X} \leftarrow \mathbf{R}[:][i : (i+1).1366]$           ▷ $\mathbf{X} \in \mathbb{R}^{96\times 1366}$
 7:     $\mathbf{Y}[i] \leftarrow CHOI\_CNN(\mathbf{X})$                 ▷ $\mathbf{Y} \in \mathbb{R}^{1024\times W}$
 8: end for
 9: $\mathbf{y}_1 = BagOfFrames(\mathbf{Y}, 1024)$                      ▷ $\mathbf{y}_1 \in \mathbb{R}^{1024}$
10: $Pred = \sigma(L(\mathbf{W}_2)ReLU(L(\mathbf{W}_1)\mathbf{y}_1))$   ▷ $\mathbf{W}_2 \in \mathbb{R}^{65\times 512}, \mathbf{W}_1 \in \mathbb{R}^{512\times 1024}$
```

4.2.2 Experiments with MFCCs as feature extractor

MFCCs are still de-facto standard for classifications on small datasets. If CNNs had to outperform MFCCs, the learned parameters should have to encode discriminants similar to MFCCs. MFCCs are computed by taking discrete-cosine transform on log mel-spectrogram (ref. **??**). Following the comparison strategies from [22], we retain 30 coefficients, their first and second derivative, resulting in a vector of size 90 for each STFT frame.

MFCC + RNN :
Now, MFCCs from every STFT window is passed in a *30s Batched sequence* to a Sequence to one LSTM, which results in a projection for every 30s window. These sequence of 30s frames are then passed to another Sequence to One LSTM to get a final projection. This is done because a MFCCs from STFT frame will result in a long sequence and RNNs tend to forget the information in the earlier sequence samples. This network was first trained with MTT dataset before our target dataset. The parameter setting are same as those used while training **CNN+RNN**. The resulting WAUC was **0.74**

Algorithm 23 $Pred = \text{MODEL}(\mathbf{a})$

```
    Input : $\mathbf{a} \in \mathbb{R}^N$
    Output : $Pred \in \mathbb{R}^{65}$
1: $\mathbf{R} = MFCC(\mathbf{a})$                                  ▷ $\mathbf{R} \in \mathbb{R}^{90\times P}$
2: $W = floor(\frac{P}{1366})$
3: for $i \in \{0, .., W\}$ do
4:     $\mathbf{X} \leftarrow \mathbf{R}[:][i : (i+1).1366]$                ▷ $\mathbf{X} \in \mathbb{R}^{90\times 1366}$
5:     $\mathbf{Y}[i] \leftarrow Drop_{(0.3)}(Seq2One\_LSTM(\mathbf{X}))$      ▷ $\mathbf{Y} \in \mathbb{R}^{1024\times W}$
6: end for
7: $\mathbf{y} = Drop_{(0.3)}(Seq2One\_LSTM(\mathbf{Y}))$               ▷ $\mathbf{y} \in \mathbb{R}^{1024}$
8: $Pred = \sigma(L(\mathbf{W})\mathbf{y})$                            ▷ $\mathbf{W} \in \mathbb{R}^{65\times 1024}$
```

MFCC + BoF :
RNN is repalaced with Bag of Frames features. Now WAUC drops to **0.62**

Algorithm 24 $Pred = \text{MODEL}(\mathbf{a})$

```
    Input : $\mathbf{a} \in \mathbb{R}^N$
    Output : $Pred \in \mathbb{R}^{65}$
1: $\mathbf{R} = MFCC(\mathbf{a})$                                  ▷ $\mathbf{R} \in \mathbb{R}^{90\times P}$
2: $\mathbf{y} = BagOfFrames(\mathbf{R}, 1024)$                       ▷ $\mathbf{y} \in \mathbb{R}^{1024}$
3: $Pred = \sigma(L(\mathbf{W}_2)ReLU(L(\mathbf{W}_1)\mathbf{y}))$     ▷ $\mathbf{W}_2 \in \mathbb{R}^{65\times 512}, \mathbf{W}_1 \in \mathbb{R}^{512\times 1024}$
```

4.3 Summary of Results

Summary of results is shown in the table below. It is seen that *transfer learning* of convolutions over mel-spectrogram with architecture in [22] cannot match with MFCCs for small datasets. This indicates that convolutional features from source dataset are more task-specific. It is also seen that *Recurrent Neural Networks* perform better in summarizing features for longer audio. This indicates the existence of rhythmic patterns that discriminate music.

Model	**AUC**
Finetune CNN + RNN	0.71
CNN + BoF	0.67
MFCC + RNN	**0.74**
MFCC + BoF	0.62

Chapter 5

Conclusion

The model settings for the task of *aesthetic tagging* have been analysed in this thesis. From the machine learning end, the performance can be further pushed by

- Searching or testing other CNN models[29]
- Working with the label space. For instance, there can be broad subsets and each tag can be belong to one or many of the subset. Then a separate model is trained for each subset

However, it is seen that the performance gap is still huge to serve any real-time application arising from aesthetic auto-tagging. Hence, to develop an algorithm that could come close to human artist, just a dataset with clean tags is not sufficient but also proper understanding of mathematical modelling of musical discriminants (ref. **??**) is important.

TODO: more ideas, discussion

Appendix A

Appendix

A.1 Basis Transformation

Here we discuss only transformation from standard basis or Cartesian basis.

The standard basis for $\mathbb{R}^N$ is the ordered sequence $\mathbf{I}_n = [\mathbf{e}_1, \mathbf{e}_2, .., \mathbf{e}_n]$, where $\mathbf{e}_i$ is a vector with 1 in i^{th} place and 0 elsewhere. Any vector $\mathbf{x} = [x_1, x_2, ..., x_n] \in \mathbb{R}^N$ can be represented as a linear combination of $\mathbf{I}_n$ as,

$$\mathbf{x} = \sum_{i=1}^{N} x_i \mathbf{e}_i = \mathbf{I}_n \mathbf{x}$$

Basis transformation from standard basis is defined as representing the same vector $\mathbf{x}$ with the new co-ordinates $[y_1, y_2, ..., y_m]$ in basis $\mathbf{V} = [\mathbf{v}_1, \mathbf{v}_2, .., \mathbf{v}_m] \in \mathbf{R}^{N \times M}$.

$$\mathbf{x} = \sum_{i=1}^{M} y_i \mathbf{v}_i = \mathbf{V}\mathbf{y} \qquad \mathbf{y} \in \mathbb{R}^M$$

$\mathbf{V}$ is also known as **change of coordinates matrix** (also stated as any matrix whose columns form a basis). If $\mathbf{V}$ is orthogonal, then $\mathbf{V}^{-1} = \mathbf{V}^T$ and hence $\mathbf{y} = \mathbf{V}^T \mathbf{x}$

A.2 Convolution

Only discrete convolutions with finite support are discussed below,

A.2.1 1D Convolution

Convolution of a vector $\mathbf{f}$ with filter $\mathbf{w}_k$ of stride s is defined as,

$$\mathbf{C}(k, i) = \sum_{n=i.s}^{i.s+F} \mathbf{f}(n) \mathbf{w}_k(n - i.s) \qquad \mathbf{f} \in \mathbb{R}^N, \mathbf{w}_k \in \mathbb{R}^F, \mathbf{C} \in \mathbb{R}^{K \times I} \tag{A.1}$$

$$\mathbf{C}(k,i) = \mathbf{f}(n) \star \mathbf{w}_k(n - i.s)$$

Where:

K is the number of filters. $k \in 0, 1 ... K-1$

I is the number of contractions. $i \in 0, 1 ... I-1$

Short Hand Notation : 1D Convolution of $\mathbf{f}$ with filter $\mathbf{w}_k$ with stride s

$$\boxed{\mathbf{C}(k,:) = \mathbf{f} \star \mathbf{w}_k^{(s)}}$$

A.2.2 2D Convolution

Convolution of a matrix $\mathbf{F}$ with filter $\mathbf{W}_k$ of row-stride s and column-stride t is defined as,

$$\mathbf{C}(k,j,i) = \sum_{n=i.s}^{i.s+F} \sum_{m=j.t}^{j.t+G} \mathbf{F}(m,n) : \mathbf{W}_k(m - j.t, n - i.s) \qquad \mathbf{F} \in \mathbb{R}^{M \times N}, \mathbf{W}_k \in \mathbb{R}^{G \times F}, \mathbf{C} \in \mathbb{R}^{K \times J \times I} \tag{A.2}$$

$$\mathbf{C}(k,j,i) = \mathbf{F}(m,n) \star \mathbf{W}_k(m - j.t, n - i.s)$$

Short Hand Notation : 2D Convolution of $\mathbf{F}$ with filter $\mathbf{W}_k$ with row-stride s and column-stride t

$$\boxed{\mathbf{C}(k,:,;) = \mathbf{F} \star \mathbf{W}_k^{(s,t)}}$$

Glossary

basis A set of linearly independent vectors B are said to form a basis of a vector space V, if every vector in V can be represented as a linear combination of B. 11, 16, 17, 21

basis transformation TODO . 17

linear combination A linear combination of set of vectors $\{\mathbf{v}_1, \mathbf{v}_2, ..., \mathbf{v}_n\}$ is defined as $a_1\mathbf{v}_1 + a_2\mathbf{v}_2 + ... + a_n\mathbf{v}_n$, where $a_1, a_2, .., a_n$ are some scalar coefficients which represent the coordinates relative to the basis B. 11, 21

Bibliography

Proceedings

[10] Taemin Cho, Ron J. Weiss, and Juan P. Bello. "Exploring common variations in state of the art chord recognition systems". In: *In Proc. of the Sound and Music Computing Conf. (SMC.* 2010.

[13] Thierry Bertin-mahieux, Daniel P. W. Ellis, Brian Whitman, and Paul Lamere. "The million song dataset". In: *In Proceedings of the 12th International Conference on Music Information Retrieval (ISMIR.* 2011.

[14] Simon Lemieux, Yoshua Bengio, Douglas Eck, and Universit De Montral. "Temporal pooling and multiscale learning for automatic annotation and ranking of music audio". In: *In: Proc. 12th International Society for Music Information Retrieval Conference.* 2011.

[17] Sander Dieleman and Benjamin Schrauwen. "Multiscale Approaches To Music Audio Feature Learning". In: *ISMIR.* 2013.

[18] S. Dieleman and B. Schrauwen. "End-to-end learning for music audio". In: *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP).* 2014, pp. 6964–6968. DOI: 10.1109/ICASSP.2014.6854950.

[20] Aäron van den Oord, Sander Dieleman, and Benjamin Schrauwen. "Transfer learning by supervised pre-training for audio-based music classification". eng. In: *Conference of the International Society for Music Information Retrieval, Proceedings.* Taipei, 2014, p. 6.

[22] Keunwoo Choi, George Fazekas, and Mark Sandler. "Automatic tagging using deep convolutional neural networks". In: *International Society of Music Information Retrieval Conference. ISMIR.* 2016.

[26] J. Pons, T. Lidy, and X. Serra. "Experimenting with musically motivated convolutional neural networks". In: *2016 14th International Workshop on Content-Based Multimedia Indexing (CBMI).* 2016, pp. 1–6. DOI: 10.1109/CBMI.2016.7500246.

Articles

[2] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-Term Memory". In: *Neural Comput.* 9.8 (Nov. 1997), pp. 1735–1780. ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.8.1735. URL: http://dx.doi.org/10.1162/neco.1997.9.8.1735.

[5] Jean-Julien Aucouturier, Boris Defreville, and Francois Pachet. "The bag-of-frames approach to audio pattern recognition: A sufficient model for urban soundscapes but not for polyphonic music". In: *The Journal of the Acoustical Society of America* 122.1 (2007), p. 881. DOI: `http://dx.doi.org/10.1121/1.2750160`.

[7] Carlos N. Silla, Alessandro L. Koerich, and Celso A. A. Kaestner. "A Machine Learning Approach to Automatic Music Genre Classification". In: *Journal of the Brazilian Computer Society* 14.3 (2008), pp. 7–18. ISSN: 1678-4804. DOI: `10.1007/BF03192561`. URL: `http://dx.doi.org/10.1007/BF03192561`.

[8] Edith Law, Kris West, Michael Mandel, Mert Bay, and J. Stephen Downie. "Evaluation of algorithms using games: The case of music tagging". In: (2009), pp. 387–392.

[9] Luis M. de Campos, Juan M. Fernndez-Luna, Juan F. Huete, and Miguel A. Rueda-Morales. "Combining content-based and collaborative recommendations: A hybrid approach based on Bayesian networks". In: *International Journal of Approximate Reasoning* 51.7 (2010), pp. 785 –799. ISSN: 0888-613X. DOI: `http://dx.doi.org/10.1016/j.ijar.2010.04.001`. URL: `http://www.sciencedirect.com/science/article/pii/S0888613X10000460`.

[11] S. K. Kopparapu and M. Laxminarayana. "Choice of Mel filter bank in computing MFCC of a resampled speech". In: (2010), pp. 121–124. DOI: `10.1109/ISSPA.2010.5605491`.

[12] Vinod Nair and Geoffrey E. Hinton. "Rectified Linear Units Improve Restricted Boltzmann Machines". In: (2010), pp. 807–814. URL: `http://www.icml2010.org/papers/432.pdf`.

[15] M. Muller, D. P. W. Ellis, A. Klapuri, and G. Richard. "Signal Processing for Music Analysis". In: *IEEE Journal of Selected Topics in Signal Processing* 5.6 (2011), pp. 1088–1110. ISSN: 1932-4553. DOI: `10.1109/JSTSP.2011.2112333`.

[16] M. Slaney. "Web-Scale Multimedia Analysis: Does Content Matter?" In: *IEEE MultiMedia* 18.2 (2011), pp. 12–15. ISSN: 1070-986X. DOI: `10.1109/MMUL.2011.34`.

[19] Diederik P. Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization." In: *CoRR* abs/1412.6980 (2014). URL: `http://dblp.uni-trier.de/db/journals/corr/corr1412.html#KingmaB14`.

[21] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. "Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)". In: *CoRR* abs/1511.07289 (2015). URL: `http://arxiv.org/abs/1511.07289`.

[24] Patrik N. Juslin, Laura S. Sakka, Gonalo T. Barradas, and Simon Liljestrm. "No Accounting for Taste? Idiographic Models of Aesthetic Judgmentin Music". In: *Psychology of Aesthetics, Creativity, and the Arts* 10.2 (2016), pp. 157–170. DOI: `10.1037/aca0000034`.

[30] Yandre M.G. Costa, Luiz S. Oliveira, and Carlos N. Silla Jr. "An evaluation of Convolutional Neural Networks for music classification using spectrograms". In: *Applied Soft Computing* 52 (2017), pp. 28 –38. ISSN: 1568-4946. DOI: `http://doi.org/10.1016/j.asoc.2016.12.024`. URL: `http://www.sciencedirect.com/science/article/pii/S1568494616306421`.

Pre-Prints

[23] Keunwoo Choi, Gyorgy Fazekas, Mark Sandler, and Kyunghyun Cho. *Convolutional Recurrent Neural Networks for Music Classification*. Version 3. Dec. 21, 2016. arXiv: `1609.04243`.

Books

[1] D. O'Shaughnessy. *Speech communication: human and machine*. Addison-Wesley series in electrical engineering. Addison-Wesley Pub. Co., 1987. ISBN: 9780201165203.

[4] R.L. Allen and D. Mills. *Signal Analysis: Time, Frequency, Scale, and Structure*. Wiley, 2004. ISBN: 9780471660361.

Misc

[3] Jean julien Aucouturier and Francois Pachet. *Music Similarity Measures: What's The Use ?* 2002.

[6] Michael A. Casey, Remco Veltkamp, Masataka Goto, Marc Leman, Christophe Rhodes, and Malcolm Slaney. *Content-Based Music Information Retrieval: Current Directions and Future Challenges*. 2008.

[25] Bastian Leibe. *Lecture - Machine Learning, RWTH Aachen*. 2016. URL: `http://www.vision.rwth-aachen.de/media/course/SS/2016/machine-learning/ml16-part06-linear_discriminants2.pdf`.

[27] Sebastian Ruder. *An overview of gradient descent optimization algorithms*. 2016. URL: `http://sebastianruder.com/optimizing-gradient-descent/`.

[28] Wikipedia. *Mel-frequency cepstrum — Wikipedia, The Free Encyclopedia*. [Online; accessed 17-April-2017]. 2016. URL: `https://en.wikipedia.org/w/index.php?title=Mel-frequency_cepstrum&oldid=747054068`.

[31] Wikipedia. *Universal approximation theorem — Wikipedia, The Free Encyclopedia*. [Online; accessed 17-April-2017]. 2017. URL: `https://en.wikipedia.org/w/index.php?title=Universal_approximation_theorem&oldid=771987359`.

[32] Lecture Notes. *Spectral Leakage and Windowing*. `https://mil.ufl.edu/nechyba/www/__eel3135.s2003/lectures/lecture19/spectral_leakage.pdf`. Online; accessed 20 March 2017.