

# **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 computing 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

<b>1</b>	<b>Introduction</b>	<b>5</b>
1.1	Motivation . . . . .	5
1.1.1	Challenges with collaborative filtering methods . . . . .	5
1.1.2	Challenges with content based methods . . . . .	6
1.2	Overview . . . . .	6
1.2.1	Signal Representation . . . . .	6
1.2.2	Dimensionality reduction . . . . .	6
1.2.3	Temporal approximation . . . . .	7
1.3	Outline of the report . . . . .	7
<b>2</b>	<b>Formalisms</b>	<b>9</b>
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
<b>3</b>	<b>Literature Survey and Model Selection</b>	<b>31</b>
3.1	Literature Review . . . . .	31
3.1.1	From hand-crafting to feature learning . . . . .	32
3.1.2	Transfer Learning by supervised pre-training . . . . .	34

3.1.3	Convolutional Neural Networks . . . . .	35
3.2	Model Selection . . . . .	38
3.2.1	Supervised learning of Reduction, Temporal approximation and Classification operators . . . . .	38
3.2.2	Supervised learning of Temporal approximation and Classification operators . . . . .	39
3.2.3	Supervised learning only for Classification operators . . . . .	40
<b>4</b>	<b>Experiments and Results</b>	<b>43</b>
4.1	Dataset . . . . .	43
4.1.1	Dataset for source task . . . . .	43
4.1.2	Dataset for target task . . . . .	44
4.2	Evaluation . . . . .	44
4.3	Experiments . . . . .	45
4.3.1	Representation Parameters : . . . . .	46
4.3.2	Perceptron Settings . . . . .	47
4.3.3	RNN Settings . . . . .	48
4.3.4	Analyses of CNN . . . . .	50
4.4	Summary of results . . . . .	53
<b>5</b>	<b>Conclusion</b>	<b>55</b>
<b>A</b>	<b>Appendix</b>	<b>57</b>
A.1	Basis Transformation . . . . .	57
A.2	Convolution . . . . .	57
A.2.1	1D Convolution . . . . .	57
A.2.2	2D Convolution . . . . .	58
<b>B</b>	<b>Experiments</b>	<b>59</b>
B.1	Validation Tags . . . . .	59
B.2	Validation Set . . . . .	60
	<b>Bibliography</b>	<b>65</b>

# 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. But semantic description of a song is not straightforward and there is this gap between music audio and listener's description. 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, an 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 Challenges 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 Challenges with content based methods

Using information from audio content to overcome the cold-start problem resulted in *content-based* recommendation methods. In such algorithms, acoustic cues required for discriminating the semantics are extracted from the audio signal. These acoustic cues have to be encoded in the lower dimensional approximation of audio signal, also known as features. However, to provide descriptions that could come closer to human intelligence, the features should also encode the perceptual information in the song. The current state-of-art content based music tagging algorithms[17][23] use large sets of data to train an algorithm to automatically detect the optimal features from a spectrogram representation of the audio signal. This kind of training is known as *feature learning*. Unsupervised learning (training data is not labelled) used in [17] may not extract features that would be optimal for all contexts of music tagging. Supervised learning used in [23][22] can extract features optimal for context used, but need large amount of labelled data for training.

## 1.2 Overview

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 labelled training data containing short excerpts of music of fixed size (29.1s). The aim of this thesis is to find out

- If the CNN models trained on large data can be exploited to show similar performance gains with *transfer-learning* on smaller dataset. That is, the CNN is first trained on a large labelled dataset and the converged weights are used as initialization for training on smaller dataset.
- Models that can extract features by approximating signals of arbitrary length.

For the purpose of analysing feature learning, the feature extraction pipeline for music information retrieval is formalised into following stages: *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 by analysing the frequency content. Motivated by the way ear-brain handles the frequency information, myriad of features extracted from spectrogram representations were evolved. In this thesis, following the analyses from current literature [22][18][17][23], only *mel-spectrogram* representation is used. *Mel-spectrograms* exploit the fact that our ear cannot distinguish adjacent frequencies (say we cannot differentiate 300 KHz and 310 KHz), and the frequencies values are binned according to a what to popularly known as *mel-scale*. In chapter 2, section 2.1.2, representation of digital audio in time-frequency format is explained. In section 2.1.3, computations of mel-spectrogram as a convolution operation is elaborated.

### 1.2.2 Dimensionality reduction

The signal representation is divided into short frames and the dimension of each frame is reduced systematically by discarding information that does not contribute to discrimination. The math-

emational operators that discard information not contributing to variance in our context specific classification task can be obtained by solving for parameters of CNNs. In chapter 2, section 2.2.4, formalism of CNN as feature extractor for MIR tasks have been illustrated and in section 2.5, challenges in training CNN have been pointed out. The training dataset for our thesis (target dataset) is small. Therefore CNN is first trained on the publicly available *MagnaTagATune* dataset[8] and *transfer-learning* on our target dataset is comparatively analysed in chapter 4, section 4.3.4.

### 1.2.3 Temporal approximation

The reduced information from each frame are temporally approximated to a fixed sized representation. The classifier then takes these fixed sized features and outputs the tags. The resulting approximation should not lose the perceptual information required for discrimination. To do this, supervised learning with *sequence to one* Recurrent Neural Network (RNN) is used, that sequentially combines reductions from each frame to a fixed size feature. Formalisms of RNN are presented in chapter 2, section 2.3.2 and comparative analyses of transfer learning with CNN + RNN models are discussed in chapter 4, section 4.3.4

## 1.3 Outline of the report

In chapter 2, the fundamentals of mathematical concepts used in this thesis are explained and formalism of notations are introduced. In chapter 3, a detailed overview of previous research and justification for our proposed models for analyses are discussed. In chapter 4, details of the datasets, evaluation metrics, parameter settings and the experiment results are analysed. In chapter 5, the inference from these experiments and future directions are proposed.





# Chapter 2

## Formalisms

To map a music to its 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  $(a_1, a_2, \dots, a_N) \in \mathbb{R}^N$  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.

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

Where  $(a_1, a_2, \dots, a_N)$  are the co-ordinates of Cartesian basis formed by basis vectors  $\mathbf{e}_1 \dots \mathbf{e}_N$  with  $t \in \{1, 2, \dots, N\}$ . 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  $\phi_\omega$  that are functions of frequencies ( $\omega$ ). Then the co-ordinates of this basis system  $c_\omega$  represents the signal in frequency domain.

$$\mathbf{a} = \sum_{\omega=1}^M c_\omega \phi_\omega = \Phi \mathbf{c} \quad \Phi \in \mathbb{C}^{N \times M}, \mathbf{c} \in \mathbb{C}^M \quad (2.1)$$

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

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

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

#### Exponential Fourier Series and Fourier Transform

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

$$\phi_k(t) = \frac{1}{\sqrt{T}} e^{ik\omega t} \quad t \in \{1, 2, \dots, T\}$$

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  $\omega$  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*

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

Thus the transformation operator is,

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

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  $O(N \log N)$  [4].

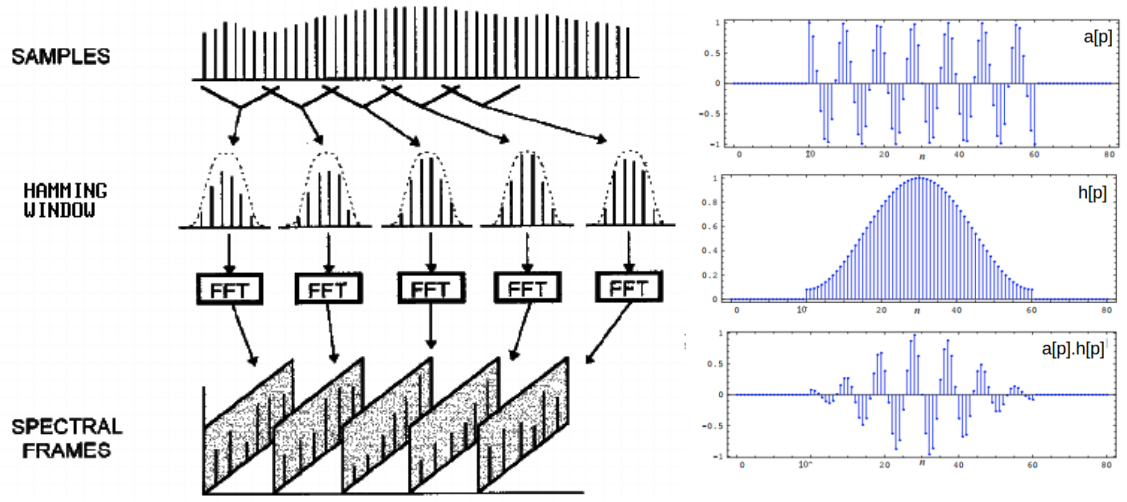
### 2.1.3 STFT, Mel-Spectrogram

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,

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

where  $n \in \{1, 2, \dots, F\}$  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  $\mathbf{a}$  is mathematically represented as a **convolution**. The resulting spectrogram has  $P$  frames. The discrete STFT (*slow*) for  $p^{th}$  frame of signal  $\mathbf{a}$  is obtained as,

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

Where:

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

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

$F$ : is the frame length

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

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

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

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

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

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

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

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

where  $\mathbf{W}_{STFT} = \mathbf{h}\Phi^{-1}$  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] :

$$\|\mathbf{c}\|^2 \propto \|\mathbf{a}\|^2 \quad (2.5)$$

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,

$$Energy \propto amplitude^2 \quad (2.6)$$

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*.

$$\mathbf{P} = \mathbf{C} \odot \mathbf{C} \quad (2.7)$$

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]

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

where  $\omega$  is the frequency in Hertz. In a mel spectrogram, the frequencies are converted to mels and then grouped into mel-spaced bins. This is done by multiplying the spectrum with **mel 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

$$\mathbf{X} = \mathbf{W}_{MEL} \mathbf{P} \quad (2.9)$$

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

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,

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

Where:

$$\begin{aligned} \mathbf{p}[k] &= \mathbf{P}[i, j] ; i = \text{floor}(\frac{k}{M}) ; j = k - \text{floor}(\frac{Mk}{M-1}) \\ \omega &= k - p.M \in \mathbb{R}^M \\ \mathbf{X} &\in \mathbb{R}^{M \times P} \\ \mathbf{p} &\in \mathbb{R}^{M.P} \\ \omega_m &\in \mathbb{R}^R \end{aligned}$$

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)

$$\boxed{\mathbf{X} = \mathbf{P} \star \mathbf{W}_{MEL}} \quad (2.11)$$

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,

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

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*

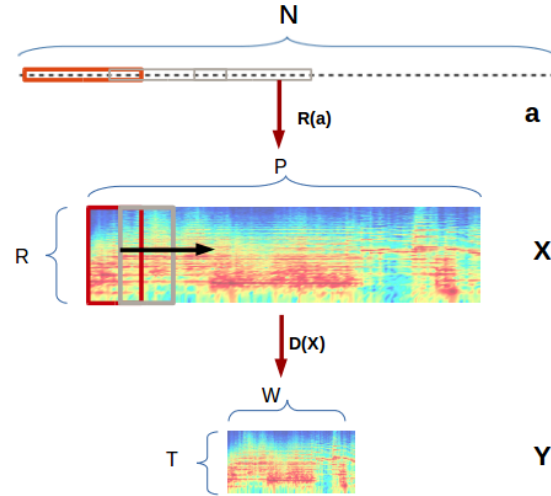


Figure 2.2: Dimensionality Reduction Pipeline

### 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 ( $\mathbf{S} = [\mathbf{X}_1, \mathbf{X}_2, \dots, \mathbf{X}_S]$ ) 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,

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

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  $\mathbb{R}^R$ .

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

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**  $\mathbf{W}_{PCA} = \text{PCA}(\mathbf{X}_1, \mathbf{X}_2, \dots, \mathbf{X}_S)$ 

---

**Input :**  $\mathbf{S} = [\mathbf{X}_1, \dots, \mathbf{X}_S]$       $\mathbf{X}_s \in \mathbb{R}^{R \times P_s}, \mathbf{S} \in \mathbb{R}^{R \times Q}, \quad Q = \sum_s P_s$

**Output :**  $\mathbf{W}_{PCA} \in \mathbb{R}^{T \times R}$

- 1:  $\mathbf{\Sigma} = \frac{1}{Q} \mathbf{S} \mathbf{S}^T$   $\triangleright \mathbf{\Sigma} \in \mathbb{S}^{R \times R}$
  - 2:  $\mathbf{V}^T \mathbf{\Lambda} \mathbf{V} = \mathbf{\Sigma}$   $\triangleright \mathbf{\Lambda} \in \mathbb{D}^{R \times R}, \mathbf{V} \in \mathbb{O}^{R \times R}$
  - 3:  $\mathbf{V} \leftarrow \mathbf{V}[:, 1:T]$   $\triangleright \mathbf{V} \in \mathbb{O}^{R \times T}$
  - 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,

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

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 simply 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.

This reduction is engineered only for log mel spectrogram representation.  $\mathbf{W}_{MFCC}$  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:  $\mathbf{X} = R_{(MEL)}(\mathbf{a})$   $\triangleright \mathbf{X} \in \mathbb{R}^{R \times P}$
- 2:  $\mathbf{X} \leftarrow \ln(\mathbf{X})$
- 3: **for**  $\omega \in \{1, \dots, T\}$  **do**
- 4:      $\mathbf{W}_{MFCC}[\omega] \leftarrow \cos(\omega \mathbf{t})$   $\triangleright \mathbf{W}_{MFCC} \in \mathbb{R}^{R \times T}, \mathbf{t} \in \mathbb{R}^R$
- 5: **end for**
- 6:  $\mathbf{Y} \leftarrow \mathbf{W}_{MFCC} \mathbf{X}$

---

## 2.2.4 Convolution neural network

Transformation of input  $\mathbf{a}$  by shifted contractions of an operator  $\mathbf{w}$  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  $\mathbf{w}$  is known as a *filter function* and the length of shift  $s$  is known as *stride*. Usually  $\mathbf{a}$  is convolved with multiple filter functions. For  $K$  filters,

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

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  $\mathbf{Y}$  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  $\mathbf{Y}$  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  $\mathbf{Y}$  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* 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  $\Phi_1$  as element-wise squaring operation and  $\Phi_2$  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: $\mathbf{C} = \Phi_1(\mathbf{a} \star \mathbf{W}_{L1}^{(s)})$	▷ $\mathbf{W}_{L1} \in \mathbb{R}^{M \times F}, \mathbf{C} \in \mathbb{C}^{M \times Q}$
2: $\mathbf{X} = \Phi_2(\mathbf{C} \star \mathbf{W}_{L2}^{(M)})$	▷ $\mathbf{W}_{L2} \in \mathbb{R}^{R \times M}, \mathbf{X} \in \mathbb{R}^{R \times Q}$
3: $\mathbf{Y} = \Phi_3(\mathbf{X} \star \mathbf{W}_{L3}^{(R)})$	▷ $\mathbf{W}_{L3} \in \mathbb{R}^{T \times R}$

---

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. 3.1.3). 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.

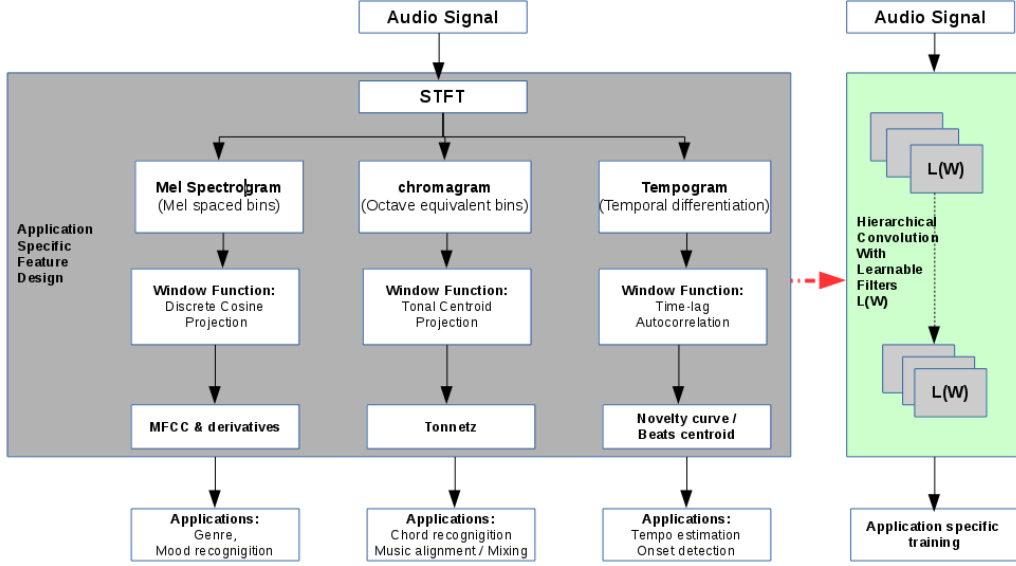


Figure 2.3: General purpose feature extractor

## 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

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

$W$  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 involving *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**  $\mathbf{f} = \text{BagOfFrames}(\mathbf{Y})$

---

**Input :**  $\mathbf{Y} \in \mathbb{R}^{T \times W}$   
**Output :**  $\mathbf{f} \in \mathbb{R}^Z$

- 1:  $\mathbf{f} = 0$
- 2: **for**  $i \in \{0, 1, \dots, W - 1\}$  **do**
- 3:      $j = \arg \min_j \|\mathbf{Y}[:, i] - \boldsymbol{\mu}_j\|^2$   $\triangleright \boldsymbol{\mu}_j \in \mathbb{R}^T, j \in \{0, 1, \dots, Z - 1\}$
- 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( $\mathbf{Y}_0, \mathbf{Y}_2, \dots, \mathbf{Y}_{N-1}$ )

---

**Input :**  $\mathbf{Y}_n \in \mathbb{R}^{T \times W}, n \in \{0, 1, \dots, N - 1\}$   
**Output :**  $\boldsymbol{\mu}_j \in \mathbb{R}^T, j \in \{0, 1, \dots, Z - 1\}$

- 1: Randomly initialize  $\boldsymbol{\mu}_j$
- 2: **while**  $\boldsymbol{\mu}_j$  have not converged **do**
- 3:     **for**  $n \in \{0, 1, \dots, N - 1\}$  **do**
- 4:         **for**  $i \in \{0, 1, \dots, W - 1\}$  **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.

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

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

$\mathbf{h}_W$  is the internal state after combining  $W$  columns of  $\mathbf{Y}$ .  $\Theta$  is the internal state function which sequentially combines the input. The operator  $\mathbf{W}$  and other operators resulting from the function  $\Theta$  are solved for optimality using a labelled dataset. These operators can be computed by training an RNN (see Section 2.5). The function  $\Theta$  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  $\Theta$  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  $\mathbf{c}_w$ .

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

$\sigma_h$  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 ( $\sigma$ ). 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  $\mathbf{g}_w$  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  $\mathbf{i}_w$  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**  $\mathbf{f} = \text{Seq2One\_LSTM}(\mathbf{Y})$ 


---

**Input :**  $\mathbf{Y} \in \mathbb{R}^{T \times W}$   
**Output :**  $\mathbf{f} \in \mathbb{R}^Z$   
1: Initialize  $\mathbf{h}_0$   
2: **for**  $w \in \{1, 2, \dots, W\}$  **do**  
3:      $\mathbf{h}_w \leftarrow \Theta_{LSTM}(\mathbf{Y}[:, w], \mathbf{h}_{w-1})$   
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 ( $\Phi_1, \Phi_2$ ) 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**  $\mathbf{f} = \text{LSTM2}(\mathbf{Y})$ 


---

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

---



---

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


---

**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**  $w \in \{1, 2, \dots, W\}$  **do**  
3:      $\mathbf{h}_w \leftarrow \Theta_{LSTM}(\mathbf{Y}[:, w], \mathbf{h}_{w-1})$   
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*<sup>1</sup> binary classifier can be formalised as,

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

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

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

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

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

---

<sup>1</sup>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

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

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  $\Phi$  is applied.

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

Without the non-linear operation  $\Phi$ , 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 boundaries. But *universal approximation theorem*[\[31\]](#) states that a single hidden layer with some non-linear activation function can approximate any function. The choice of  $\Phi$  will be explained in section 2.5. The second operator  $\mathbf{W}$  performs the projection from hidden vector  $\mathbf{h}$

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

where  $\sigma$  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  $\epsilon$  can be set in-between 0 and 1 and eventually the final prediction  $\mathbf{pred}$  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**  $\text{pred} = \text{Model}(\mathbf{a})$ 


---

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

---

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

$$\boldsymbol{\zeta} = \sigma(\mathbf{W}\Phi(\mathbf{W}_{L1}\mathbf{f})) \quad (2.14)$$

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.  $\zeta_i$  is the classifier output for  $i^{th}$  label and  $i \in \{1, 2, \dots, L\}$ . With sigmoid ( $\sigma$ ) 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  $\zeta_i$  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  $E = \text{loss}(\boldsymbol{\zeta}, \mathbf{t})$
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  $(t_i^1, t_i^2, \dots, t_i^N)$  is

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

where the *likelihood* is 1, as the classifier output  $\zeta_i$  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 be 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

$$\frac{\partial E}{\partial \mathbf{W}} = \frac{\partial E}{\partial \zeta} \frac{\partial \zeta}{\partial \sigma} \frac{\partial \sigma}{\partial \mathbf{W}} \quad (2.15)$$

$$\frac{\partial E}{\partial \mathbf{W}_{L1}} = \frac{\partial E}{\partial \zeta} \frac{\partial \zeta}{\partial \sigma} \frac{\partial \sigma}{\partial \Phi} \frac{\partial \Phi}{\partial \mathbf{W}_{L1}} \quad (2.16)$$

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  $\eta$  is the learning rate. But choosing a proper learning rate is difficult because if  $\eta$  is too small, convergence can be too slow, while  $\eta$  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.  $\beta_1$  and  $\beta_2$  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  $\beta_1$ , 0.999 for  $\beta_2$ , and  $10^{-8}$  for  $\epsilon$

## 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  $\mathbf{f}$ . But, if it is possible to solve for the transformation operators that compute the features  $\mathbf{f}$ , 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  $\mathbf{f}$ . Because, now the operators of RNN can be solved for optimality in addition to the operators of 2 layer perceptron, and therefore  $\mathbf{f}$  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 $\text{pred} = \text{Model}(\mathbf{a})$

---

<b>Input :</b> $\mathbf{a} \in \mathbb{R}^N$ <b>Output :</b> $\text{pred}$ 1: $\mathbf{X} = R(\mathbf{a})$ 2: $\mathbf{Y} = D(\mathbf{X})$ 3: $\mathbf{f} = \text{Seq2One\_LSTM}(\mathbf{Y} \mid \mathbf{W}_k, \mathbf{U}_k)$ 4: $\zeta = C(\mathbf{f} \mid \mathbf{W}, \mathbf{W}_{L1})$ 5: $\text{pred} = \{b(\zeta_i) \mid b(\zeta_i) = 1\}$	$\triangleright$ indices of predicted labels $\triangleright \mathbf{X} \in \mathbb{R}^{R \times P}$ $\triangleright \mathbf{Y} \in \mathbb{R}^{T \times W}$ $\triangleright k = \{i_r, o, g, c\}, \mathbf{f} \in \mathbb{R}^Z$ $\triangleright \zeta \in \mathbb{R}^L$ $\triangleright i \in \{1, 2, \dots, L\}, b(\zeta_i) \in \{0, 1\}$
---	---

---

However, the performance is still limited by the frame-wise features  $\mathbf{Y}$ . 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**  $\text{pred} = \text{Model}(\mathbf{a})$ 

---

**Input :**  $\mathbf{a} \in \mathbb{R}^N$ **Output :**  $\text{pred}$ 

- |   |   |
|---|---|
| 1: $\mathbf{X} = R(\mathbf{a})$   | $\triangleright$ indices of predicted labels<br>$\triangleright \mathbf{X} \in \mathbb{R}^{R \times P}$ |
| 2: $\mathbf{Y} = \text{CNN}(\mathbf{X} \mid \mathbf{W}_{C1}, \mathbf{W}_{C2}, \mathbf{W}_{C3})$ | $\triangleright \mathbf{Y} \in \mathbb{R}^{T \times W}$   |
| 3: $\mathbf{f} = \text{Seq2One\_LSTM}(\mathbf{Y} \mid \mathbf{W}_k, \mathbf{U}_k)$              | $\triangleright k = \{i_r, o, g, c\}, \mathbf{f} \in \mathbb{R}^Z$                                      |
| 4: $\zeta = C(\mathbf{f} \mid \mathbf{W}, \mathbf{W}_{L1})$                                     | $\triangleright \zeta \in \mathbb{R}^L$   |
| 5: $\text{pred} = \{b(\zeta_i) \mid b(\zeta_i) = 1\}$   | $\triangleright i \in \{1, 2, \dots, L\}, b(\zeta_i) \in \{0, 1\}$                                      |
- 

From the illustration in algorithm 6, it can be seen that the learning problem can be 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  $p$ . ( $\text{Drop}_{(p)}$ )

#### ***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 2.15 and the penultimate layer in equation 2.16, 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  $\Phi$  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. 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 the previous chapter, the abstract algorithm for music classification task was introduced. However, it can be seen that there are plenty of parameters (like sampling rate, STFT parameters, CNN hyper-parameters) to fix. Therefore, in this chapter, we also briefly dive into some of the successful algorithms in the literature to demonstrate the optimal choice parameters or network settings one can use. 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

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 the previous chapter, computing the optimal features for a context-based classification task using deep learning was formalised. However, it is not always true that such *learned* features will perform better than *hand-crafted* features. 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.

**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 the following subsections, adoption of feature learning techniques for multi-label classification task are elaborated. All the models (except [26]) were experimented for *multi-label classification task* on Magna Tag a Tune dataset(MTT)[8] with about 29K clips which are 29.1s long.

### 3.1.1 From hand-crafting to feature learning

A number of features have been engineered in the past that are relevant for different MIR tasks. MFCC features (ref. 2.2.3), 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. Results from some proceedings that compare MFCCs (feature engineering) with feature learning for *multi-label classification task* have been discussed.

#### **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. 2.2.2) (unsupervised learning for frame features) is compared with MFCC features (engineered 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. From this research, we can infer that unsupervised learning can extract features more relevant than MFCCs for music tagging.

**Parameter settings :** 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. The resulting reduction is transformed by the operator  $\mathbf{W}_{L1}$  that learns (supervised) optimal features for some pooling function  $T_{(POOL)}$ . The temporal pooling is done by summarizing every 2.3s frame with suitable functions (see [14] for details). The resulting feature is then classified by two layer perceptron with 1000 hidden units with sigmoid ( $\sigma$ ) activations. This algorithm is also an illustration for how supervised feature learning is pushed beyond the classifier and into the temporal pooler, but not into the frame-wise reduction stage (see chapter 2 2.5).

---

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

---

**Input :**  $\mathbf{a} \in \mathbb{R}^N$   
**Output :**  $\text{pred}$   $\triangleright$  indices of predicted labels

- 1:  $\mathbf{C} = \mathbf{a} \star \mathbf{W}_{STFT}$   $\triangleright \mathbf{C} \in \mathbb{C}^{M \times P}$
- 2:  $\mathbf{C} \leftarrow \mathbf{C} \odot \mathbf{C}$
- 3:  $\mathbf{X} = \mathbf{C} \star \mathbf{W}_{MEL}$   $\triangleright \mathbf{X} \in \mathbb{R}^{128 \times P}$
- 4:  $\mathbf{Y}_1 = D_{(PCAW)}(\mathbf{X})$   $\triangleright \mathbf{Y}_1 \in \mathbb{R}^{120 \times P}$
- 5:  $\mathbf{Y}_2 = \mathbf{W}_{L1} \mathbf{Y}_1$   $\triangleright \mathbf{W}_{L1} \in \mathbb{R}^{S \times 120}, \mathbf{Y}_2 \in \mathbb{R}^{T \times P}$
- 6:  $\mathbf{f} = T_{(POOL)}(\mathbf{Y}_2)$   $\triangleright \mathbf{f} \in \mathbb{R}^{T.W}$
- 7:  $\zeta = \sigma(\mathbf{W}_{L3} \sigma(\mathbf{W}_{L2} \mathbf{f}))$   $\triangleright \mathbf{W}_{L2} \in \mathbb{R}^{1000 \times T.W}, \mathbf{W}_{L3} \in \mathbb{R}^{L \times 1000}$
- 8:  $\text{pred} = \{b(\zeta_i) | b(\zeta_i) = 1\}$   $\triangleright i \in \{1, 2, \dots, L\}, b(\zeta_i) \in \{0, 1\}$

---

**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**). In this research, supervision is introduced only in the classifier. The reduction and temporal approximation are done with sophisticated unsupervised learning. Here the mel-spectrogram is transformed to  $W$  gaussian pyramids and features from each level of the pyramid are extracted separately and concatenated. At each level, the spectrogram is divided into  $T$  segments and Bag of Frames features are extracted separately on every segment after PCA whitening operation. The signal is sampled at 16 KHz and the STFT window size is 1024 and hop size 512. The mel-spectrogram has 200 bins.

---

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

---

**Input :**  $\mathbf{a} \in \mathbb{R}^N$   
**Output :**  $\text{pred}$   $\triangleright$  indices of predicted labels

- 1:  $\mathbf{C} = \mathbf{a} \star \mathbf{W}_{STFT}$   $\triangleright \mathbf{C} \in \mathbb{C}^{M \times P}$
- 2:  $\mathbf{C} \leftarrow \mathbf{C} \odot \mathbf{C}$
- 3:  $\mathbf{X} = \mathbf{C} \star \mathbf{W}_{MEL}$   $\triangleright \mathbf{X} \in \mathbb{R}^{R \times P}$
- 4: **for**  $i \in \{1, \dots, Z\}$  **do**
- 5:      $\mathbf{Y} \leftarrow \text{Gaussian\_Pyramid}(\mathbf{X}, i)$   $\triangleright \mathbf{Y} \in \mathbb{R}^{R \times W_{1i}}$
- 6:     **for**  $j \in \{1, \dots, T\}$  **do**
- 7:          $\mathbf{F}_1[:, j] \leftarrow \text{BagOfFrames}(D_{(PCAW)}(\mathbf{Y}_1))$   $\triangleright \mathbf{F}_1 \in \mathbb{R}^{T \times W_i}$
- 8:     **end for**
- 9:      $\mathbf{F}[i] \leftarrow T_{MaxPool}(\mathbf{F}_1)$   $\triangleright \mathbf{F} \in \mathbb{R}^{T \times W}$
- 10: **end for**
- 11:  $\mathbf{f} = \text{Flatten}(\mathbf{F})$   $\triangleright \mathbf{f} \in \mathbb{R}^{T.W}$
- 12:  $\zeta = \sigma(\mathbf{W}_{L3} \text{ReLU}(\mathbf{W}_{L2} \mathbf{f}))$   $\triangleright \mathbf{W}_{L2} \in \mathbb{R}^{1000 \times T.W}, \mathbf{W}_{L3} \in \mathbb{R}^{L \times 1000}$
- 13:  $\text{pred} = \{b(\zeta_i) | b(\zeta_i) = 1\}$   $\triangleright i \in \{1, 2, \dots, L\}, b(\zeta_i) \in \{0, 1\}$

---

### 3.1.2 Transfer Learning by supervised pre-training

Supervised 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, the reduction operators 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,

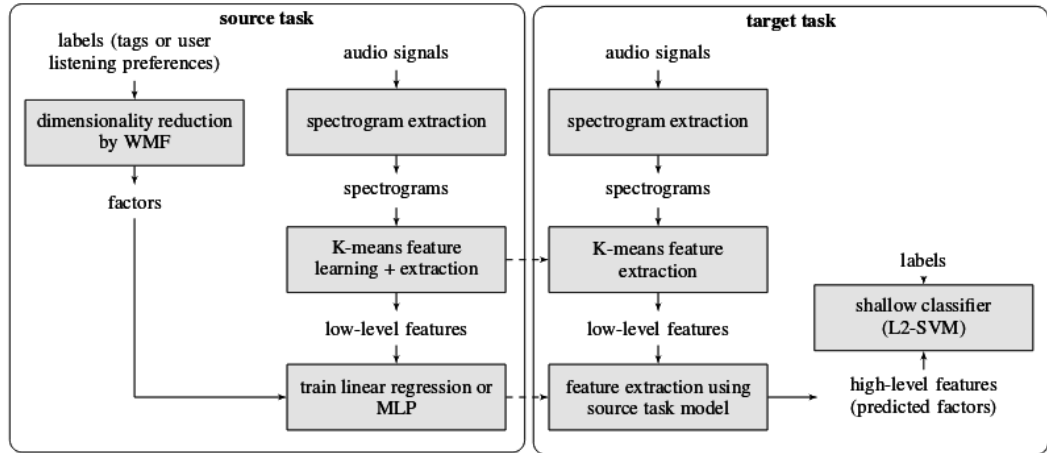


Figure 3.1: Schematic overview of the workflow of transfer learning[20]

**Source task:** The features from audio spectrograms are learned through unsupervised learning by spherical K-Means. A multi layer perceptron (supervised) 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 Weighted Matrix Factorization (WMF). 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.

### 3.1.3 Convolutional Neural Networks

It was shown that supervised learning can be pushed deep into the feature extraction pipeline by stacking multiple layers of learnable convolution filters (ref. 2.2.4). The idea was to replace the application specific dimension reductions with Convolution Neural Networks(CNN). In this section, some works that throw insights into CNN architecture settings are discussed.

#### 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 push supervision directly on to raw audio signal. The signal was convolved with 3 layers of 1D convolutions followed by two layer perceptron. Thus, feature learning is applied directly on the input and this is called *end to end learning*. They compared the *end to end learning* approach with supervised learning after mel-power-spectrogram on MTT dataset (i.e, retaining  $\mathbf{W}_{STFT}$  and  $\mathbf{W}_{MEL}$ ). It was found that, discarding STFT hurt the performance. CNN from mel-spectrogram achieved **AUC 0.8815**, but on including supervision directly on audio signal, AUC dropped to 0.8487.

Algorithm 17 CNN(raw audio) [0.84]	Algorithm 18 CNN(Mel-Power-Spectrogram) [0.88]
<b>Input :</b> $\mathbf{a} \in \mathbb{R}^N$ <b>Output :</b> <b>pred</b> 1: $\mathbf{Y}_1 = \Phi_1(\mathbf{a} \star \mathbf{W}_{C1}^{(256)})$ 2: $\mathbf{Y}_2 = \text{MaxPool}(\text{ReLU}(\mathbf{Y}_1 \star \mathbf{W}_{C2}^{(1)}))$ 3: $\mathbf{Y}_3 = \text{MaxPool}(\text{ReLU}(\mathbf{Y}_2 \star \mathbf{W}_{C3}^{(1)}))$ 4: $\mathbf{f} = \text{Flatten}(\mathbf{Y}_3)$ 5: $\zeta = \sigma(\mathbf{W}_{L2} \text{ReLU}(\mathbf{W}_{L1} \mathbf{f}))$ 6: <b>pred</b> = $\{b(\zeta_i)   b(\zeta_i) = 1\}$ $i \in \{1, 2, \dots, L\}, b(\zeta_i) \in \{0, 1\}$ 7:	<b>Input :</b> $\mathbf{a} \in \mathbb{R}^N$ <b>Output :</b> <b>pred</b> $\triangleright$ indices of predicted labels 1: $\mathbf{C} = \mathbf{a} \star \mathbf{W}_{STFT}$ 2: $\mathbf{C} \leftarrow \mathbf{C} \odot \mathbf{C}$ 3: $\mathbf{X} = \mathbf{C} \star \mathbf{W}_{MEL}$ 4: $\mathbf{Y}_1 = \text{MaxPool}(\text{ReLU}(\mathbf{X} \star \mathbf{W}_{C1}^{(1)}))$ 5: $\mathbf{Y}_2 = \text{MaxPool}(\text{ReLU}(\mathbf{Y}_1 \star \mathbf{W}_{C2}^{(1)}))$ 6: $\mathbf{f} = \text{Flatten}(\mathbf{Y}_2)$ 7: $\zeta = \sigma(\mathbf{W}_{L2} \text{ReLU}(\mathbf{W}_{L1} \mathbf{f}))$ 8: <b>pred</b> = $\{b(\zeta_i)   b(\zeta_i) = 1\}$

Algorithm 17 is an the end-to-end learned classifier. The signal  $\mathbf{a}$  is convolved with filter  $\mathbf{W}_{C1}$  by replacing the STFT operation. Function  $\Phi_1$  is an element-wise logarithmic compression refereed in [18]. Two more layers of CNN are stacked over. The non-linearities in the second and third layer are *ReLU* (see Sec. 2.5.2) followed by a pooling operation (*MaxPool*). Max pooling is done by segmenting the resulting reduction and taking just the maximum value in each segment. This is done to further reduce the dimension. The features from CNN are taken by the classifier. This model expects input of fixed size (29.1s) and hence a temporal approximator is not used. During test-run, predictions for an arbitrary length of signal is done by segmenting it to 29.1s sections are performing classification separately on each section.

#### 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.

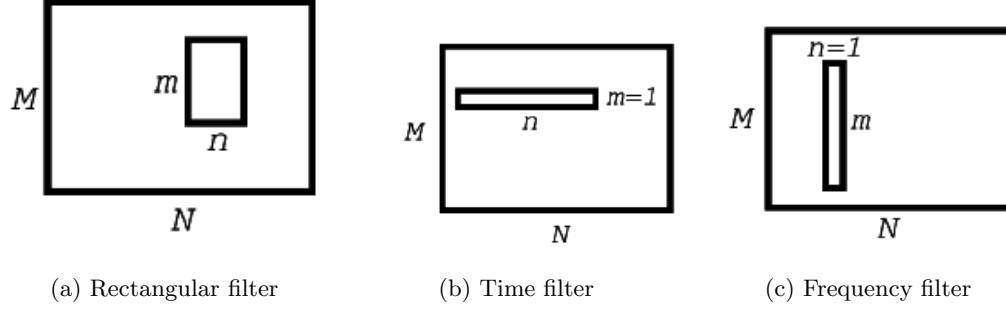


Figure 3.2: Different Filter sizes[26]

*Time filters* (fig 3.2b) can learn temporal cues like *onset* and *beats per minute*, while *frequency filters* (fig 3.2c) can differentiate timbre and note. *Rectangular filters* can learn short time sub-bands (Bass, kick, drums)[26]. It was shown in their experiments that *rectangular filters* or combination of time and frequency filters performed better than using just time or frequency filter. These experiments were done for a genre classification task.

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

The proposed model in this research achieves close to state of art performance on MTT dataset (**0.894 AUC**). The audio samples were down-sampled to 12 KHz and supervision was pushed until mel spectrogram extraction (with 96 bins). They also compared the features - MFCCs, CNN over STFT and CNN over Mel-log power spectrogram and report that the latter performs significantly better.

Features	AUC
STFT $\rightarrow$ CNN	0.846
STFT $\rightarrow$ MEL $\rightarrow$ CNN	<b>0.894</b>
STFT $\rightarrow$ MEL $\rightarrow$ MFCC	0.862

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 the data is *learned* to be scaled and shifted.

---

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

---

**Input :**  $\mathbf{X} \in \mathbb{R}^{B \times S \times Q}$ ,  $\triangleright B$  is batch size  
**Output :**  $\hat{\mathbf{X}} \in \mathbb{R}^{B \times S \times Q}$   
**Parameters to learn :**  $\gamma$  (Scale),  $\beta$  (Shift)  
1: Compute the frequency means  $\boldsymbol{\mu}$  and variance  $\boldsymbol{\sigma}^2$  from  $\mathbf{X}$   $\triangleright \boldsymbol{\mu}, \boldsymbol{\sigma}^2 \in \mathbb{R}^S$   
2: **for**  $i \in \{1, \dots, B\}$  **do**  
3:   **for**  $j \in \{1, \dots, Q\}$  **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 proposed CNN feature extractor with mel-log power spectrogram as input is shown in algorithm 20. There are five layers of convolutions with 2D filters (rectangular filter refereed in [26])  $\mathbf{W} \in \mathbb{R}^{K \times G \times F}$ .  $K$  is the number of filters, while  $G$  and  $F$  are sizes of the 2D filter. The resulting reduction is a 3D tensor. The convolution operation with 2D filter is shown in appendix A.2.2. The non-linearities include *spatial batch normalization* (*Spatial\_Bn*), followed by transformation with *exponential linear units* (*Elu*, see Sec. 2.5.2) and 2D max pooling (*MaxPool*). *Spatial\_Bn* is similar to the normalization algorithm mentioned above, except that the normalization is done along the 1st axis of tensors  $\mathbf{Y}$ . *MaxPool* <sub>$i,j$</sub>  is a dimensionality reduction done by pooling  $(i, j)$  elements along  $S$  and  $Q$  directions respectively.

---

**Algorithm 20**  $\mathbf{f} = \text{CHOI\_CNN}(\mathbf{X})$ 

---

**Input :**  $\mathbf{X} \in \mathbb{R}^{1 \times 96 \times 1366}$   $\triangleright \mathbf{X}$  is log-mel-power spectrogram  
**Output :**  $\mathbf{f} \in \mathbb{R}^{1024}$   
1:  $\mathbf{X} \leftarrow \text{BatchNorm}(\mathbf{X})$   
2:  $\mathbf{Y}_1 = \mathbf{X} \star \mathbf{W}_{C1}^{(1,1)}$   $\triangleright \mathbf{W}_{C1} \in \mathbb{R}^{32 \times 3 \times 3}, \mathbf{Y}_1 \in \mathbb{R}^{32 \times S1 \times Q1}$   
3:  $\mathbf{Y}_1 \leftarrow \text{MaxPool}_{(2,4)}(\text{Elu}(\text{Spatial\_Bn}(\mathbf{Y}_1)))$   $\triangleright \mathbf{Y}_1 \in \mathbb{R}^{32 \times T1 \times W1}$   
4:  $\mathbf{Y}_2 = \mathbf{Y}_1 \star \mathbf{W}_{C2}^{(1,1)}$   $\triangleright \mathbf{W}_{C2} \in \mathbb{R}^{128 \times 3 \times 3}, \mathbf{Y}_2 \in \mathbb{R}^{128 \times S2 \times Q2}$   
5:  $\mathbf{Y}_2 \leftarrow \text{MaxPool}_{(2,4)}(\text{Elu}(\text{Spatial\_Bn}(\mathbf{Y}_2)))$   $\triangleright \mathbf{Y}_2 \in \mathbb{R}^{128 \times T2 \times W2}$   
6:  $\mathbf{Y}_3 = \mathbf{Y}_2 \star \mathbf{W}_{C3}^{(1,1)}$   $\triangleright \mathbf{W}_{C3} \in \mathbb{R}^{128 \times 3 \times 3}, \mathbf{Y}_3 \in \mathbb{R}^{128 \times S3 \times Q3}$   
7:  $\mathbf{Y}_3 \leftarrow \text{MaxPool}_{(2,4)}(\text{Elu}(\text{Spatial\_Bn}(\mathbf{Y}_3)))$   $\triangleright \mathbf{Y}_3 \in \mathbb{R}^{128 \times T3 \times W3}$   
8:  $\mathbf{Y}_4 = \mathbf{Y}_3 \star \mathbf{W}_{C4}^{(1,1)}$   $\triangleright \mathbf{W}_{C4} \in \mathbb{R}^{192 \times 3 \times 3}, \mathbf{Y}_4 \in \mathbb{R}^{192 \times S4 \times Q4}$   
9:  $\mathbf{Y}_4 \leftarrow \text{MaxPool}_{(2,4)}(\text{Elu}(\text{Spatial\_Bn}(\mathbf{Y}_4)))$   $\triangleright \mathbf{Y}_4 \in \mathbb{R}^{192 \times T4 \times W4}$   
10:  $\mathbf{Y}_5 = \mathbf{Y}_4 \star \mathbf{W}_{C5}^{(1,1)}$   $\triangleright \mathbf{W}_{C5} \in \mathbb{R}^{256 \times 3 \times 3}, \mathbf{Y}_5 \in \mathbb{R}^{256 \times S5 \times Q5}$   
11:  $\mathbf{Y}_5 \leftarrow \text{Elu}(\text{Spatial\_Bn}(\mathbf{Y}_5))$   
12:  $\mathbf{f} = \text{Flatten}(\mathbf{Y}_5)$   $\triangleright \mathbf{f} \in \mathbb{R}^{1024}$

---

This feature extractor also requires a fixed sized mel-log power spectrogram from an input of length 29.1s. The resulting features then pass through a single layer perceptron of size equalling number of tags and sigmoid activation. The authors have then trained this model on MSD dataset and made the weights publicly available.

## 3.2 Model Selection

The performance of content based music tagging task will be better if the features extracted from the signal does not lose the information required for discriminating the semantics. CNNs and RNNs together can be used to impose supervision into the feature extraction pipeline to obtain features that could be optimal for the task. But the challenge is that, deep neural networks require large amount of data to realize operators that can out-perform solutions by engineered and unsupervised methods (eg. [14][17]). Given that our target dataset is small, we cannot use it directly to train the deep neural network. Therefore, the model is first trained with MTT dataset and the resulting converged weights can be used as initialization while training with the target dataset[20]. In this thesis, transfer learning is analysed by controlling the depth of supervision in the target dataset.

Recall that the feature extraction pipeline was divided in to three stages : Representation, Reduction and Temporal approximation. With this formalism, the depth of supervision can be controlled by training on our target dataset at following supervision levels :

- Supervised learning of Representation, Reduction, Temporal approximation and Classification operators (end to end learning)
- Supervised learning of Reduction, Temporal approximation and Classification operators
- Supervised learning of Temporal approximation and Classification operators
- Supervised learning only for Classification operators

The classifier always requires supervised training because of the nature of our task. Multi-layer perceptron is fixed as the classifier for all our experiments. Training through representation was shown sub-optimal in [18] and [22]. Hence we do not analyse supervision pushed to this level.

### 3.2.1 Supervised learning of Reduction, Temporal approximation and Classification operators

The representation operators are fixed and the operators for dimensionality reduction, temporal approximation and classification are solved for optimality. That is, the gradient of the error is back-propagated upto reduction operations. Deeper the network, more data is required for convergence. This model is first trained on MTT dataset and then further fine-tuned on the target dataset.

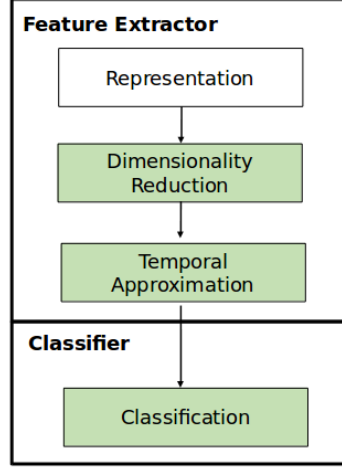


Figure 3.3: Error Back-propagation upto reduction operations

**Representation :** In [22] and [18], CNN with log mel-power spectrogram as input was shown to be optimal for the music tagging task. Hence we compute the log-mel power spectrogram for all the analyses.

**Reduction :** The log mel power spectrogram is convolved with learnable filters. In [26], 2D convolutions were shown to be more motivating than 1D. In [22], a 5 layers of 2D convolutions were used for reduction, which achieves close to state of art performance on MTT dataset. Therefore, we choose their CNN model (algorithm 20) for dimensionality reduction.

**Temporal approximation :** Supervision can be introduced at this stage with a *sequence to one* Recurrent Neural Network (see chapter 2 2.3.2). Long Short-term memory(LSTM) and Gated recurrent units(GRU) are the choices at our disposal. In this thesis, we investigate only LSTMs.

### 3.2.2 Supervised learning of Temporal approximation and Classification operators

The representation and reduction operators are not supervised on our target dataset. The representation and temporal approximation models are same as described in section 3.2.1. This model is also first trained on MTT dataset and then further fine-tuned on the target dataset.



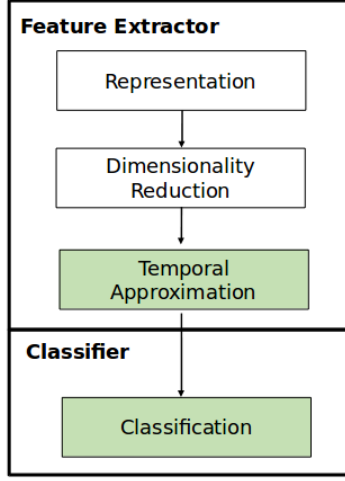


Figure 3.4: Error Back-propagation upto temporal approximation operations

**Reduction:** The CNN trained on large dataset is used as a black box feature extractor for the target classification task. That is, the weights converged on large dataset are not further modified by fine tuning on target dataset. In addition to black box CNN, we also investigate MFCCs because it is still not clear if CNNs can out perform MFCCs for a small dataset.

### 3.2.3 Supervised learning only for Classification operators

Supervision is not imposed on the feature extraction pipeline. The representation and reduction models are same as described in section [3.2.2](#)

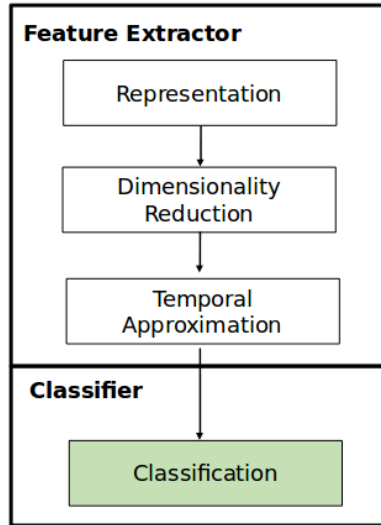


Figure 3.5: Error Back-propagation only through classifier

**Temporal Approximation:** Bag of frames features (see Sec. 2.3.1 ) obtained with unsupervised learning using K-means clustering algorithm was shown to be optimal for music tagging task in [17]. Comparing Bag of frames features with RNN tells if deep learning with transfer learning can boost performance when training dataset is small.



## Chapter 4

# Experiments and Results

Performance of features from Convolution Neural Network are analysed in this chapter. To analyse CNNs, the datasets, evaluation metrics and network settings have to be fixed. In section 4.1, the details of source and target dataset are presented. In section 4.2, the evaluation metrics that will be used are discussed. In section 4.3, experiments to analyse transfer learning on models selected in chapter 3, section 3.2 are performed.

### 4.1 Dataset

More specific to our task than representing audio is finding a proper dataset of labelled pairs. To analyse *transfer learning*, a large dataset is needed for the *source task*. 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(MTT) dataset consists of 25,856 clips of 29.1-s 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. Frequently matched tags are collected to build a labelled dataset. This dataset is the largest available that comes close to minimizing the *audio-semantic* noise. Only the top-50 tags from MTT are used for source task training. This is because, the CNN architecture in [22] for our analyses was also trained with top 50 tags (These tags were from MSD dataset. However, only the number '50' is important, which would otherwise change the size of final layer in neural network and might destroy the richness of pre-trained features). Since

our target task is to approximate features for songs of arbitrary length, clips from same songs in MTT dataset are merged as a single sample.

### 4.1.2 Dataset for target task

The annotations were gathered with one of the most straightforward approach - ask someone to listen to songs and tag them. This was done to have a clean mapping of perceptual properties to semantics. 842 songs approximately 5 - 8 min long , tagged by my supervisor Prof. Paolo Bientinesi in association with Prof. Marco Aluno (Professor of Composition and Theory at University EAFIT, Columbia) were used. Out of 842 songs, 100 are used for validation. The validation set is listed in appendix B.2. 63 tags listed in appendix B.1 were used for validation.

## 4.2 Evaluation

For multi-label classification with L labels, the performance of L binary classifiers are computed and averaged. Each label can belong to one of the class - *positive* (1) or *negative* (0). Area under Receiver operating characteristic Curve (AUC) is computed for each label. Mean of AUCs weighted by number of occurrence of each tag is reported. In this section, the need for this performance measure is discussed by comparing with other standard measures. To do this, some terminologies from information retrieval are recalled,

**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 ground truth 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, let us 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 does not tell anything about the percentage of false negatives. That is, even if the classifier *correctly* admits one '*electro*' as positive and 899 as negative (19 false negatives),

*precision* would be 1.0

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

**Recall :** This is also not comprehensive because it does not tell anything about false positives. 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 the threshold is changed, then the performance can also 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*.

**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 of a sample is positive, how often does the classifier admits this sample as positive'. *Fall-out* answers the question, 'when the ground truth of a sample is negative, how often does the classifier admits this sample 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 observation with positive ground truth higher than a randomly chosen observation with negative ground truth. 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.

### 4.3 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). The abstract algorithm for content based multi-label classifier is shown in algorithm 21. Function *R* is the representation operation,

$D$  represents dimensionality reduction operations (see Sec. 2.2),  $T$  is the temporal approximation (see Sec. 2.3) and  $C$  is the multi-label classifier (see Sec. 2.4).

---

**Algorithm 21**  $\text{pred} = \text{Model}(\mathbf{a})$

---

**Input :**  $\mathbf{a} \in \mathbb{R}^N$   
**Output :**  $\text{pred}$   $\triangleright$  indices of predicted labels  
1:  $\mathbf{X} = R(\mathbf{a})$   $\triangleright \mathbf{X} \in \mathbb{R}^{R \times P}$   
2:  $\mathbf{Y} = D(\mathbf{X})$   $\triangleright \mathbf{Y} \in \mathbb{R}^{T \times W}$   
3:  $\mathbf{f} = T(\mathbf{Y})$   $\triangleright \mathbf{f} \in \mathbb{R}^Z$   
4:  $\boldsymbol{\zeta} = C(\mathbf{f})$   $\triangleright \boldsymbol{\zeta} \in \mathbb{R}^{63}$   
5:  $\text{pred} = \{b(\zeta_i) | b(\zeta_i) = 1\}$   $\triangleright i \in \{1, 2, \dots, 63\}, b(\zeta_i) \in \{0, 1\}$

---

### 4.3.1 Representation Parameters :

As discussed in section 3.2 of previous chapter, supervision is not imposed to compute representation operators. Hence, the following representation parameters from [22] are fixed to compute *log mel power spectrogram* for all experiments.

Parameter	Setting
Sampling rate	12 KHz
STFT window function	Hamming window
Size of each segment in STFT	512 (42 ms)
Hop-Length	256
FFT Size	512
Mel bins	96

Table 4.1: Representation Parameters

The function  $R$  is thus replaced with log-mel power spectrogram computations. Step 1 - 4 in algorithm 22 represents the computation of log mel-power spectrogram (see Chapter 2, Sec. 2.1.3). The abstract algorithm for our experiments is shown below.

---

**Algorithm 22**  $\text{pred} = \text{Model}(\mathbf{a})$ 


---

**Input :**  $\mathbf{a} \in \mathbb{R}^{12000 \cdot t}$   
**Output :**  $\text{pred}$   $\triangleright$  indices of predicted labels

- 1:  $\mathbf{C} = \mathbf{a} \star \mathbf{W}_{STFT}^{(256)}$   $\triangleright \mathbf{C} \in \mathbb{R}^{512 \times P}, \mathbf{W}_{STFT} \in \mathbb{R}^{512 \times 512}$
- 2:  $\mathbf{C} \leftarrow \mathbf{C} \odot \mathbf{C}$
- 3:  $\mathbf{X} = \mathbf{C} \star \mathbf{W}_{MEL}^{(512)}$   $\triangleright \mathbf{X} \in \mathbb{R}^{96 \times P}, \mathbf{W}_{MEL} \in \mathbb{R}^{96 \times 512}$
- 4:  $\mathbf{X} \leftarrow \ln(\mathbf{X})$
- 5:  $\mathbf{Y} = D(\mathbf{X})$   $\triangleright \mathbf{Y} \in \mathbb{R}^{T \times W}$
- 6:  $\mathbf{f} = T(\mathbf{Y})$   $\triangleright \mathbf{f} \in \mathbb{R}^Z$
- 7:  $\zeta = C(\mathbf{f})$   $\triangleright \zeta \in \mathbb{R}^{63}$
- 8:  $\text{pred} = \{b(\zeta_i) | b(\zeta_i) = 1\}$   $\triangleright i \in \{1, 2, \dots, 63\}, b(\zeta_i) \in \{0, 1\}$

---

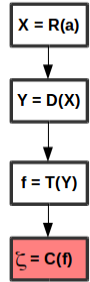
### 4.3.2 Perceptron Settings

Two layer perceptron (equation 2.14) with rectified linear units activation (*ReLU*) activation in the hidden layer is used for classification  $C$  (step 7 of algorithm 23). *ReLU* is used to handle the vanishing gradient problem. The operators  $\mathbf{W}_{L1}$  and  $\mathbf{W}_{L2}$  are solved for optimality. (see Sec. 2.5). To find the optimal settings for perceptron, functions  $D$  and  $T$  are fixed. Thus, log mel-power spectrogram is reduced to MFCC features and approximated to a fixed size representation with Bag of Frames features. Hence, the error is back-propagated only to the classifier  $C$  and the resulting learning problem is said to be shallow.

---

**Algorithm 23**  $\text{pred} = \text{Model}(\mathbf{a})$ 


---



```

graph TD
    A["X = R(a)"] --> B["Y = D(X)"]
    B --> C["f = T(Y)"]
    C --> D["zeta = C(f)"]
    style D fill:#f00,stroke:#f00
    
```

**Input :**  $\mathbf{a} \in \mathbb{R}^{12000 \cdot t}$   
**Output :**  $\text{pred}$   $\triangleright$  indices of predicted labels

- 1:  $\mathbf{C} = \mathbf{a} \star \mathbf{W}_{STFT}^{(256)}$   $\triangleright \mathbf{C} \in \mathbb{R}^{512 \times P}, \mathbf{W}_{STFT} \in \mathbb{R}^{512 \times 512}$
- 2:  $\mathbf{C} \leftarrow \mathbf{C} \odot \mathbf{C}$
- 3:  $\mathbf{X} = \mathbf{C} \star \mathbf{W}_{MEL}^{(512)}$   $\triangleright \mathbf{X} \in \mathbb{R}^{96 \times P}, \mathbf{W}_{MEL} \in \mathbb{R}^{96 \times 512}$
- 4:  $\mathbf{X} \leftarrow \ln(\mathbf{X})$
- 5:  $\mathbf{Y} = \mathbf{X} \star \mathbf{W}_{MFCC}^{(96)}$   $\triangleright \mathbf{Y} \in \mathbb{R}^{90 \times P}, \mathbf{W}_{MFCC} \in \mathbb{R}^{90 \times 96}$
- 6:  $\mathbf{f} = \text{BagOfFrames}(\mathbf{Y})$   $\triangleright \mathbf{f} \in \mathbb{R}^Z$
- 7:  $\zeta = \sigma(\mathbf{W}_{L2} \text{ReLU}(\mathbf{W}_{L1} \mathbf{f}))$   $\triangleright \zeta \in \mathbb{R}^{63}, \mathbf{W}_{L2} \in \mathbb{R}^{63 \times H}, \mathbf{W}_{L1} \in \mathbb{R}^{H \times Z}$
- 8:  $\text{pred} = \{b(\zeta_i) | b(\zeta_i) = 1\}$   $\triangleright i \in \{1, 2, \dots, 63\}, b(\zeta_i) \in \{0, 1\}$

---

The network is trained with ADAM optimizer. The training settings that are used in this thesis were explained in Chapter 2, section 2.5. All ADAM parameters except the learning rate  $\eta$  are used as proposed in [27]. The learning rate can be different for different sections of the network and hence we differentiate the learning rate of the classifier with subscript  $c$  ( $\eta_c$ ). The learning rate is reduced by factor  $\gamma_c$  every  $\text{step}_c$  iterations. This is done because in the beginning, large learning rate is needed to approach the solution, but as the training approaches actual solution, learning rate has to be reduced. Otherwise, the solution might be sub optimal. However, the learning rate is not reduced below  $10^{-6}$ . Even though the classifier is shallow, the network is first trained on the MTT dataset (source task) until convergence. The resulting weights are used as initialization for training



on target dataset. Since our aim is to analyse only the CNNs, the training hyper-parameters values for the classifier are fixed with values shown in table ??.

Hyper-parameter	Source task	Target task
$\eta_c$	$1^{-2}$	$1^{-3}$
$\gamma_c$	0.1	0.1
$step_c$	10000	3000

Table 4.2: Training Hyper-Parameters for Classifier

To set up a two layer neural network, the size of the classifier input (or feature)  $Z$ , the dimension of hidden layer  $H$  and the size of the output  $L$  should be known. The size of the output is equal to the number of labels in test. 63 labels are used for validation in target set. The experiments for different  $H$  and  $Z$  and their resulting WAUC scores are shown in table ??. (Here the features size ( $Z$ ) resulting from the Bag of frames is equal to the number of cluster centres to be chosen)

	$H = 512$	$H = 1024$	$H = 2048$
$Z = 512$	0.63	0.64	0.63
$Z = 1024$	0.64	<b>0.67</b>	0.62
$Z = 2048$	0.61	0.61	0.59

Table 4.3: Hyper parameter search : Classifier hidden dimension (H), Feature dimension (Z)

When the feature size is 512, it can be seen that the performance is capped around 0.64 for different hidden size. The performance improves with feature size 1024. However, the performance decreases when the dimension increases to 2048. This tells us that the classifier complexity with hidden size 512 is in-sufficient. But the model begins to over fit (learns the noisy data) when the size increases to 2048. Hence the feature dimension ( $Z$ ) and hidden dimension ( $H$ ) are fixed to **1024** for all the remaining analyses.

### 4.3.3 RNN Settings

The error from classifier  $C$  is back-propagated into the temporal approximation function  $T$  with a 2 layer LSTM. Therefore, the Bag of frames algorithm is replaced with LSTM. The operators of LSTM  $\Phi_{R1}$  and  $\Phi_{R2}$  are solved to sequentially combine frame-wise features into a fixed size projection. The LSTM operators

$$\Phi_{Rj} = \{\mathbf{W}_{(k,j)}, \mathbf{U}_{(k,j)}\} \quad k \in \{i, o, g, c\}, j \in \{1, 2\}$$

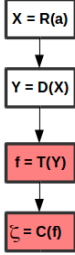
were described in Chapter 2, section 2.3.2. Thus the learning is pushed into the temporal approximation function. While combining MFCC features, the number of STFT frames can be large for longer songs and the information at the beginning of sequence can get lost. To get around this problem, the first layer of LSTM is a *sequence to one* network that combines frames up to 29.1s. The second layer of *sequence to one* LSTM combines features from every 29.1s frame to a fixed sized projection. This computation is illustrated in step 6 - 12 of algorithm 24. There are 1366 STFT

frames for 29.1s of 12 KHz sampled signal. Drop out with probability  $p$  is used as the transition operation (referred in chapter 2, algorithm 10). Drop out acts as a regularizer that prevents the network from over-fitting.

---

**Algorithm 24**  $\text{pred} = \text{Model}(\mathbf{a})$

---



**Input :**  $\mathbf{a} \in \mathbb{R}^{12000.t}$

**Output :**  $\text{pred}$   $\triangleright$  indices of predicted labels

1:  $\mathbf{C} = \mathbf{a} \star \mathbf{W}_{STFT}^{(256)}$   $\triangleright \mathbf{C} \in \mathbb{R}^{512 \times P}, \mathbf{W}_{STFT} \in \mathbb{R}^{512 \times 512}$

2:  $\mathbf{C} \leftarrow \mathbf{C} \odot \mathbf{C}$

3:  $\mathbf{X} = \mathbf{C} \star \mathbf{W}_{MEL}^{(512)}$   $\triangleright \mathbf{X} \in \mathbb{R}^{96 \times P}, \mathbf{W}_{MEL} \in \mathbb{R}^{96 \times 512}$

4:  $\mathbf{X} \leftarrow \ln(\mathbf{X})$

5:  $F = 1366$

6:  $W = \text{floor}(\frac{P}{F})$

7: **for**  $i \in \{0, \dots, W\}$  **do**

8:      $\mathbf{G} \leftarrow \mathbf{X}[:, (i+1)F]$

9:      $\mathbf{Y} \leftarrow \mathbf{G} \star \mathbf{W}_{MFCC}^{(96)}$   $\triangleright \mathbf{Y} \in \mathbb{R}^{90 \times F}, \mathbf{W}_{MFCC} \in \mathbb{R}^{90 \times 96}$

10:      $\mathbf{F}[i] = \text{Drop}_{(p)}(\text{Seq2One\_LSTM}(\mathbf{Y} | \Phi_{R2}))$   $\triangleright \mathbf{F} \in \mathbb{R}^{T \times W}$

11: **end for**

12:  $\mathbf{f} = \text{Drop}_{(p)}(\text{Seq2One\_LSTM}(\mathbf{F} | \Phi_{R1}))$   $\triangleright \mathbf{f} \in \mathbb{R}^{1024}$

13:  $\boldsymbol{\zeta} = \sigma(\mathbf{W}_{L2} \text{ReLU}(\mathbf{W}_{L1} \mathbf{f}))$   $\triangleright \boldsymbol{\zeta} \in \mathbb{R}^{63}, \mathbf{W}_{L2} \in \mathbb{R}^{63 \times 1024}, \mathbf{W}_{L1} \in \mathbb{R}^{1024 \times 1024}$

14:  $\text{pred} = \{b(\zeta_i) | b(\zeta_i) = 1\}$   $\triangleright i \in \{1, 2, \dots, 63\}, b(\zeta_i) \in \{0, 1\}$

---

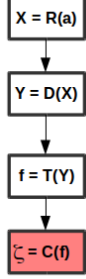
To set up a 2 layer LSTM, the size of hidden projection  $T$  and the drop-out probability  $p$  have to be known. Training settings for source and target tasks are same as that of the classifier. WAUC scores for different  $p$  and  $T$  are shown in table ??.

	$T = 512$	$T = 1024$	$T = 2048$
$p = 0$	0.66	0.61	0.59
$p = 0.3$	0.68	<b>0.74</b>	0.68
$p = 0.5$	0.67	0.71	0.64

Table 4.4: Hyper parameter search : RNN hidden dimension (T), Drop-out probability (p)

The optimum classifier settings from experiments in previous section is used ( $H = Z = 1024$ ). The hidden dimension ( $T$ ) 1024 is found to be optimal. It can be seen that the network over-fits without the drop-out ( $p = 0$ ). Therefore,  $p = 0.3$  and  $T = 1024$  is used fixed for remaining analyses.

### Black-box LSTM



The operators of LSTM trained on the source task can also be used as black-box feature extractor for the target task (that is, LSTM is not further trained with target data). This is done because fine-tuning on the target training set might degrade the operators sometimes. This could either be because of convergence of source task training towards general purpose features or because of insufficient training data in the target set. To check this, LSTM trained on the source dataset is used as a black-box feature extractor for the target task. But, for the optimal LSTM settings from experiments in previous section ( $p = 0.3$ ,  $T = 1024$ ), WAUC drops to **0.65**. This shows that the source task features cannot be used as general purpose features and still needs task-dependent training.

### 4.3.4 Analyses of CNN

With the perceptron and RNN settings from section 4.3.2 and 4.3.3, we are now ready to analyse CNNs. Instead of MFCC feature extraction operators ( $\mathbf{W}_{MFCC}$ ), hierarchical layers of learn-able operators  $\Phi_C$  are used (see Sec. 2.2.4). As discussed in chapter 3, section 3.2.1, 5 layers of 2D convolution neural network architecture from [22] is used (see algorithm 20). Thus,

$$\Phi_C = \{\mathbf{W}_{Cj}\} \quad j \in \{1, 2, 3, 4, 5\}$$

Features are extracted with CNN over 29.1s log mel-spectrogram. The resulting features from every 29.1s frames are sequentially combined to a fixed size representation with LSTM. (In algorithm 24, the input to first layer of RNN was features from every STFT frame. But now, the input is from every 29.1s frame, and this sequence is not as large as the former). Two layers of LSTM with *sequence to sequence* layer in between is used (see algorithm 10).

---

#### Algorithm 25 $\text{pred} = \text{Model}(\mathbf{a})$

---

```

graph TD
    A["X = R(a)"] --> B["Y = D(X)"]
    B --> C["f = T(Y)"]
    C --> D["ζ = C(f)"]
  
```

**Input :**  $\mathbf{a} \in \mathbb{R}^{12000.t}$

**Output :** **pred** ▷ indices of predicted labels

1:  $\mathbf{C} = \mathbf{a} \star \mathbf{W}_{STFT}^{(256)}$  ▷  $\mathbf{C} \in \mathbb{R}^{512 \times P}$ ,  $\mathbf{W}_{STFT} \in \mathbb{R}^{512 \times 512}$

2:  $\mathbf{C} \leftarrow \mathbf{C} \odot \mathbf{C}$

3:  $\mathbf{X} = \mathbf{C} \star \mathbf{W}_{MEL}^{(512)}$  ▷  $\mathbf{X} \in \mathbb{R}^{96 \times P}$ ,  $\mathbf{W}_{MEL} \in \mathbb{R}^{96 \times 512}$

4:  $\mathbf{X} \leftarrow \ln(\mathbf{X})$

5:  $F = 1366$

6:  $W = \text{floor}(\frac{P}{F})$

7: **for**  $i \in \{0, \dots, W\}$  **do**

8:      $\mathbf{G} \leftarrow \mathbf{X}[:, (i+1)F]$

9:      $\mathbf{Y}[i] = \text{CHOI\_CNN}(\mathbf{G} | \Phi_C)$  ▷  $\mathbf{Y} \in \mathbb{R}^{1024 \times W}$

10: **end for**

11:  $\mathbf{F} = \text{Drop}_{(0.3)}(\text{Seq2Seq\_LSTM}(\mathbf{Y} | \Phi_{R2}))$  ▷  $\mathbf{F} \in \mathbb{R}^{1024 \times W}$

12:  $\mathbf{f} = \text{Drop}_{(0.3)}(\text{Seq2One\_LSTM}(\mathbf{F} | \Phi_{R1}))$  ▷  $\mathbf{f} \in \mathbb{R}^{1024}$

13:  $\zeta = \sigma(\mathbf{W}_{L2} \text{ReLU}(\mathbf{W}_{L1} \mathbf{f}))$  ▷  $\zeta \in \mathbb{R}^{63}$ ,  $\mathbf{W}_{L2} \in \mathbb{R}^{63 \times 1024}$ ,  $\mathbf{W}_{L1} \in \mathbb{R}^{1024 \times 1024}$

14: **pred** =  $\{b(\zeta_i) | b(\zeta_i) = 1\}$  ▷  $i \in \{1, 2, \dots, 63\}$ ,  $b(\zeta_i) \in \{0, 1\}$

---

## Source task training

The CNN architecture that we use has been already analysed in [22] for multi-label classification task with fixed sized input on MSD dataset. These weights are used as initialization for training our source task model that handles input of arbitrary size. The CNN hyper parameters (number of layers, transition functions, filter sizes and number of filters in each layer) proposed in [22] is used. But we search for optimum training parameters for CNN ( $\eta_c, step_c$ ). The reduction factor  $\gamma_c$  is fixed at 0.1.  $\eta_c$  and  $step_c$  can be different for different CNN layers. However, the CNN is split into two sections : layer 1,2 and layer 3,4,5. The training hyper-parameters are set separately for each section. Table ?? shows WAUC of experiments on source set where only the last three out of five layers of CNN is fine-tuned. That is, the learning rate for the first two layers is 0 (Freeze).

Layer = 1,2	Layer = 3,4,5	WAUC
Freeze	( $1^{-3}$ ; 30K)	0.79
Freeze	( $1^{-3}$ ; 50K)	<b>0.81</b>
Freeze	( $1^{-3}$ ; 70K)	0.74
Freeze	( $1^{-4}$ ; 30K)	0.79
Freeze	( $1^{-4}$ ; 50K)	<b>0.81</b>
Freeze	( $1^{-4}$ ; 70K)	0.77

Table 4.5: Source task experiments with 1st two layers freezed : ( $\eta_f, step_f$ )

It can be seen that learning rate  $1^{-3}$  and  $1^{-4}$  stepping down every 50K iterations have similar performance. In table ??, WAUC of experiments where the whole CNN is trained are reported. The learning rate and step down for the last three layers are chosen from the experiments in table ??.

Layer = 1,2	Layer = 3,4,5	WAUC
( $1^{-5}$ ; 50K)	( $1^{-3}$ ; 50K)	0.82
( $1^{-5}$ ; 50K)	( $1^{-4}$ ; 50K)	0.79
( $1^{-4}$ ; 50K)	( $1^{-3}$ ; 50K)	<b>0.84</b>
( $1^{-4}$ ; 50K)	( $1^{-4}$ ; 50K)	0.77
( $1^{-4}$ ; 70K)	( $1^{-3}$ ; 50K)	0.82
( $1^{-4}$ ; 70K)	( $1^{-4}$ ; 50K)	0.79
( $1^{-3}$ ; 50K)	( $1^{-3}$ ; 50K)	0.83
( $1^{-3}$ ; 50K)	( $1^{-4}$ ; 50K)	0.76
( $1^{-3}$ ; 70K)	( $1^{-3}$ ; 50K)	0.81
( $1^{-3}$ ; 70K)	( $1^{-4}$ ; 50K)	0.76

Table 4.6: Source task experiments with all layers trained : ( $\eta_f, step_f$ )

Training all the layers of CNN on the source dataset with learning rate  $1^{-4}$  in layers 1,2 and  $1^{-3}$  in layers 3,4,5 is shown to be optimal. It can also be inferred that the last layers of CNN tend to find task specific features not suited for general purpose classification.

### Target task training

After the training converges on the source dataset, the weights are initialized for training on target dataset. The training hyper-parameters can be different for the target task. The step down iteration is lower for the target task because of smaller training data.

Layer = 1,2	Layer = 3,4,5	WAUC
Freeze	$(1^{-3}; 5K)$	0.65
Freeze	$(1^{-3}; 10K)$	0.63
Freeze	$(1^{-3}; 20K)$	0.61
Freeze	$(1^{-4}; 5K)$	0.67
Freeze	$(1^{-4}; 10K)$	<b>0.68</b>
Freeze	$(1^{-4}; 20K)$	0.63

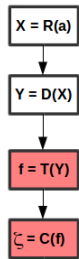
Table 4.7: Target task experiments with 1st two layers freezed :  $(\eta_f, step_f)$

Layer = 1,2	Layer = 3,4,5	WAUC
$(1^{-6}; NA)$	$(1^{-4}; 10K)$	0.64
$(1^{-5}; 10K)$	$(1^{-4}; 10K)$	<b>0.71</b>
$(1^{-5}; 20K)$	$(1^{-4}; 10K)$	0.69
$(1^{-4}; 10K)$	$(1^{-4}; 10K)$	0.68
$(1^{-4}; 20K)$	$(1^{-4}; 10K)$	0.67

Table 4.8: Target task experiments with all layers trained :  $(\eta_f, step_f)$

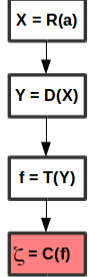
Fine-tuning all the layers of CNN on the target dataset with learning rate  $1^{-5}$  in layers 1,2 and  $1^{-4}$  in layers 3,4,5 is shown to be optimal. However, the performance is slightly lesser than what is achieved with MFCC features.

### Black-box CNN + Fine-tune LSTM



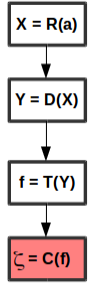
To see if the reduction in performance is because of fine-tuning on a small target dataset, CNN trained on source data is used as black-box feature extractor for target task (CNN is not trained on target data). This tells us if the features of CNN trained on source task are richer than MFCCs. The LSTM is however fine tuned with the settings discussed earlier. But WAUC drops to **0.65**, which shows that the CNN features from source task may not be rich enough. However, it is still not clear because we do not know if fine-tuning LSTM on target data is optimal for extracting information from CNN features.

### Black-box CNN + Black-box LSTM



Now both CNN and LSTM are used as black-box feature extractor. Thus, only the classifier is trained on target data. The WAUC further drops to **0.63**. It can be seen that training (source data) both CNN and LSTM in a single pipeline restricts the optimality of RNN (Recall that MFCC + LSTM performed better than MFCC + BoF). This could result in CNN features that are not as rich as MFCC.

### Black-box CNN + Bag of Frames



Replacing black-box LSTM with Bag of frames algorithm for black-box CNN features, WAUC increases to **0.67**. Thus, CNN features obtained by training CNN + LSTM on the source task data still contain information for classification. But, CNN features are not as rich as MFCC for our target task.

## 4.4 Summary of results

Transfer learning with CNN and LSTM for music tagging task were analysed by stage-wise induction of supervised learning deeper into the model. Classification on MFCC features approximated with Bag of frames algorithm does not require supervised learning and hence used as baseline for comparison. LSTM were used to introduce supervised learning of temporal approximation function. CNN and LSTM were then trained in a single pipeline to introduce supervision into reduction operations. Performances of features resulting from LSTM and CNN with and without (black-box) fine-tuning on target data have been discussed.

### Analyses of LSTM

LSTM analyses are summarized in table ?? . LSTM trained on source task, but used as a black box extractor on target task performs below the BoF baseline. But when LSTM is fine-tuned on target data, performance improves significantly. Therefore, it is seen that LSTM is still not sufficient to realize general purpose approximation operators, but requires task dependent fine-tuning to out-perform BoF.

<b>Model</b>	<b>WAUC</b>
MFCC + BoF	0.67
MFCC + Black-box LSTM	0.65
MFCC + Fine-tune LSTM	<b>0.74</b>

Table 4.9: Summary of LSTM analyses

### Analyses of CNN

CNN analyses are summarized in table ?? . CNN and LSTM were trained in a single pipeline on the source dataset. Performance of black-box CNN with Bag of frames approximation is again better than black-box LSTM. But performance of black-box CNN with fine-tuned LSTM is lesser than it's MFCC counter-part and this shows that CNN features are not as rich as MFCC to serve as general purpose feature extractor. Fine-tuning CNN improves the performance, but it is still slightly lesser than MFCC. This shows that fine-tuning CNN on smaller dataset still cannot outperform MFCC for our target task.

<b>Model</b>	<b>WAUC</b>
Black-box CNN + BoF	0.67
Black-box CNN + Black-box LSTM	0.63
Black-box CNN + Fine-tune LSTM	0.65
Fine-tune CNN + Fine-tune LSTM	0.71

Table 4.10: Summary of CNN + LSTM analyses

## Chapter 5

# Conclusion

The performance of music tagging algorithm will be higher if the extracted features encode the acoustic cues necessary for discriminating the semantics. Supervised feature learning is used to obtain features that could be optimal for our context of music tagging. Convolution and recurrent neural networks were used to induct supervised learning into the feature extraction pipeline. The number of parameters to solve increases as the learning problem becomes deeper and hence more training data is required for convergence. Therefore, in this thesis, transfer learning with models trained on *MagnaTagaTune* dataset were analysed by comparing different levels of fine-tuning on the target dataset. It has been found that fine-tuning entire CNN + RNN on target data boosts the performance and there were significant performance difference with black-box CNN. It would be useful to bridge this gap to improve performance, because deep levels of training on our target data, which is small, struggles to out-perform MFCC features.

### Proposal for improvements

- Since MFCC performs better than fine-tuned CNNs, it would be worthy to analyse CNN architectures trained on the source data by initializing CNN filters with cosine functions to get richer features. (In chapter 2, section 2.2.4, an illustration of MFCC computation as convolution operations was shown)
- Singular value decomposition of the converged filters at each CNN layer can be performed to see if they are converging to any known mathematical function. If it does, then it would be useful to set those functions as initialization before training.





# 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  $\mathbf{I}_n = [\mathbf{e}_1, \mathbf{e}_2, \dots, \mathbf{e}_n]$ , where  $\mathbf{e}_i$  is a vector with 1 in  $i^{th}$  place and 0 elsewhere. Any vector  $\mathbf{x} = [x_1, x_2, \dots, x_n] \in \mathbb{R}^N$  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  $[y_1, y_2, \dots, y_m]$  in [basis](#)  $\mathbf{V} = [\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_m] \in \mathbb{R}^{N \times M}$ .

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

$\mathbf{V}$  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,

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

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

Where:

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

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

**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,

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

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

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

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

# Appendix B

## Experiments

### B.1 Validation Tags

The list of tags and it's number of occurrences in the validation set.

No.	Tag	Freq
1	down beat	3
2	blip	3
3	bass	39
4	deep	34
5	dark	4
6	kick	33
7	tight	9
8	bass lead	33
9	empty	7
10	tiro	5
11	female	51
12	house	59
13	happy	4
14	piano	11
15	male	19
16	spoken	7
17	aggressive	8
18	tztz	31
19	disco	8
20	chic	3
21	melody	11

No.	Tag	Freq
22	jazzy	5
23	drums	4
24	train	4
25	trip	2
26	soulful	27
27	unzug	12
28	song	32
29	steady	12
30	slow	8
31	loopy	4
32	volvo	11
33	open	7
34	strings	18
35	techno	5
36	tribal	3
37	electro	4
38	1 by 2	5
39	flute	4
40	gay	3
41	gentle	5
42	mellow	4

No.	Tag	Freq
43	soft	2
44	rough	2
45	energy	8
46	voices	3
47	get down	6
48	warm	4
49	elegant	4
50	funkyg	6
51	upbeat	6
52	hypnotic	5
53	choir	3
54	scary	3
55	horns	2
56	drama	8
57	space	6
58	ethereal	6
59	silly	5
60	effects	2
61	epic	4
62	keys	4
63	fast	4

## B.2 Validation Set

1	v_Marie St James - Closer I Get
2	v_Degrees Of Motion - Do You Want It Right Now (Extended Club Mix)
3	v_Alison Limerick - Where Love Lives (Sauna Mix)
4	v_Soldiers Of Twilight - Believe (Extended)
5	v_Dj Spiller - Groovejet (If This Ain't Love) (Spiller's Radio Edit)
6	Soul Providers - Rise (Ricky Montanari & Stefano Greppi Dark Vocal Mix)
7	v_Filippo Naugthy Moscatello - Final Signal
8	v_Aaron-Carl - Dance Naked (Soulman's Eviction Loops)
9	v_Basement Jaxx - Rendez-Vu (All U Crazyes)
10	v_Deep Dish - My Only Sin
11	v_St Etienne - Only Love Can Break Your Heart (MAW Dub)
12	v_Mondo Grosso - Star Suite (Shelter Dub)
13	v_Brother Of Soul - Ife Bobowa
14	Malena Perez - Cool Lil Thing (Alix Alvarez dub)
15	v_Big Moses ft Kenny Bobian - Brighter Days (Original Mix)
16	v_The Product G&B ft Carlos Santana - Dirty Dancing (Robbie Rivera's Tribal Sessions Sub)
17	v_NSK Tune - Out Of My Mind (NR Main Dub)
18	v_Johnick - The Captain
19	v_Mongobonix - Mas-Pito
20	v_Basement Jaxx - Rendez-Vu (Radio Edit)
21	v_Mang'e Le Funk - I Still Want You
22	LUPO - Hell Or Heaven (Extended Mix)
23	v_Cevin Fisher - Music Saved My Life (Freezy Jam Mix)
24	v_Capricorn - Love In London (2001 Mix)
25	v_Didier Sinclair - Lovely Flight (Dj Chris Pi Airlines Mix)
26	v_Africanism - Bisou Sucre
27	v_Freestyle Orchestra - Twi-Lite (K-Dope Twi-Lite)
28	v_D.J Spen - Da Changez
29	v_Men From The Nile - Watch Them Come
30	v_Deepswing - In The Music (Original Version)
31	Simone - Hey Fellas (Morel's Gospel Mix)
32	v_Liquid Measures ft Jocelyn Brown - Take Me Up (Pasta Boys club remix)
33	v_Jaydee - Plastic Dreams (Morales Club Mix)
34	v_H20 ft Billie - Take Me Higher (Original Mix)
35	v_Clivilles & Cole - A Deeper Love (A Deeper Feeling Mix)
36	v_Confession - I Found My Love (Club Mix A)
37	v_Discorosso - Discorosso (Kama Kama Vrs.No.1)
38	v_Depeche Mode - I Feel Loved (Danny Tenaglia's Labor of Love Instrumental)
39	v_Djum Djum - Difference (Sagwa Mix)
40	v_Coco Steel & Lovebomb - Feel It (Vox Mix)

41	v_Coco Steel & Lovebomb - Feel It (Version Mix)
42	v_Robbie Rivera - Feel This (Robbie Rivera's Tribal Sessions Mix)
43	v_Paperclip People - Throw
44	v_MAW - Unreleased Project (Clouds)
45	v_D.J Spen - Back When
46	v_E-Smoove - Be With You (Vocal Vibes)
47	v_Soul Vision - Don't Stop (Edit)
48	v_Lil' Louis & The World - Club Lonely (DJ Pierre's Afro Club Mix)
49	v_Meli'sa Morgan - Still In Love With You (Still In Love Mix)
50	v_Rachid - Pride (Album Version)
51	v_Metro Area - Strut
52	v_Mekon ft Mark Almond - Please Stay (Bertrand Burgalat's Vocal Remix)
53	v_Massive Attack - Unfinished Sympathy (Paul Oakenfold Mix)
54	v_Interfront - Bases Mix R-5
55	v_Interfront - Mathausen
56	v_Interfront - Bonus Beats Strange
57	v_Future Sound Of London - Papua New Guinea (Qube Mix)
58	v_Junior Sanchez - N.A.S.T.Y
59	v_Full Intention pres Hustle Espanol - (Do The) Spanish Hustle (Dub Mix)
60	v_Brother Of Soul - Celebration Of Life
61	v_Gat Decor - Passion (D.Emerson Edit)
62	v_Paperclip People - Remake Uno
63	v_KenLou 3 - What A Sensation (Sensational Beats)
64	v_Chiapet - Tick Tock (Apocalypse Now Mix)
65	v_Aaron-Carl - Dance Naked (C&M Balearic Drama Remix)
66	v_Starchaser - Jambe Myth (Dsnny Jc vs Speed City Mix)
67	v_Lil' Devious - Come Home (Dave Clarke Remix)
68	v_Bob Marley vs Funkstar De Luxe - Sun Is Shining (ATB Club Mix)
69	v_Cricco Castelli - Life Has Changed
70	v_Sandy Rivera - Love For Free
71	Martha Wash - Leave a Light On
72	v_Lil' Louis & The World - Club Lonely (Bellbottoms & Platforms Mix)
73	v_Liquid Woman - Come And Go With Me (Anthem Radio Edit)
74	v_Age Of Love - The Age Of Love (Sign Of The Time Mix)
75	v_Interfront - Strange Instru
76	v_Sonique - Feels So Good (En-Motion Remix)
77	v_Robert Owens - I'll Be Your Friend (Dead Zone)
78	v_Crystal Waters - Gypsy Woman (Basement Boys 'Strip To The Bone' Mix)
79	v_Daft Punk - Around The World (Kenlou Mix)
90	v_Those Guys - Sierra Leone (Main)
81	v_Kerri Chandler - Atmosphere (Kerri's Foundation Dub)

82	v_Rui Da Silva - Touch Me (Peace Division Mix)
83	v_Cottage Alert - Lost Love (Original Demo Cut)
84	v_Black Masses - Wonderful Person (Wonderful Brazil Dub)
85	Basil - City Streets (Kerri Chandler Kaoz On City Streets)
86	v_Africanism - The Dragon
87	v_Black Science Orchestra - Keep On Keepin' On (Spen & Karisma's Deepah Dub)
88	v_The Rolling Stones - Saint Of Me (Deep Dish Grunge Garage Dub)
89	v_Chez Damier - Can You Feel It (Vocal Club Mix)
90	v_Armand van Helden - Entra Mi Casa (Original Long Version)
91	Jamie Lewis ft Michael Watford - For You (Moodbangers Mix)
92	KC Flightt vs Funky Junction - Voices (Pete Heller Main Mix)
93	v_Toni Braxton - Spanish Guitar (Joe Claussel Dub)
94	v_Backroom Congregation - Sunday Morning (WMC 98 Instrumental)
95	Una Mas - I Will Follow You (Full Intention Club Mix)
96	v_Modjo - No More Tears (Highpass vs Triple X Remix)
97	v_Massive Attack - Unfinished Sympathy
98	v_Bob Sinclair - New York City Music
99	Kimara Lovelace - I Luv You More (Sean McCabe Demo Mix)
100	v_The Black Science Orchestra - Sunshine (Sunset Vocal)







# 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](https://doi.org/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](https://doi.org/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](https://doi.org/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](http://dx.doi.org/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. Fernandez-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](http://dx.doi.org/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](http://dx.doi.org/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](http://dx.doi.org/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 Judgment in Music”. In: *Psychology of Aesthetics, Creativity, and the Arts* 10.2 (2016), pp. 157–170. DOI: [10.1037/aca0000034](http://dx.doi.org/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](https://arxiv.org/abs/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](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](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](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/eeel3135.s2003/lectures/lecture19/spectral\\_leakage.pdf](https://mil.ufl.edu/nechyba/www/eeel3135.s2003/lectures/lecture19/spectral_leakage.pdf). Online; accessed 20 March 2017.