Learning Audio Features For Genre Classification with Deep Belief Networks

by

Satya Venkata Mohan Noolu
Bachelor of Technology, Jawaharlal Nehru Technological University, 2013

A Project Submitted in Partial Fulfillment
of the Requirements for the Degree of

MASTER OF SCIENCE
in the Department of Computer Science

Learning Audio Features For Genre Classification with Deep Belief Networks

by

Satya Venkata Mohan Noolu
Bachelor of Technology, Jawaharlal Nehru Technological University, 2013

Supervisory Committee

Dr. George Tzanetakis, Supervisor
Department of Computer Science

Dr. Alex Thomo, Committee member
Department of Computer Science

Abstract

Supervisory Committee

Dr. George Tzanetakis, Supervisor
Department of Computer Science

Dr. Alex Thomo, Committee member
Department of Computer Science

Feature extraction is a crucial part of many music information retrieval(MIR) tasks. In this project, I present an end to end deep neural network that extract the features for a given audio sample, performs genre classification. The feature extraction is based on Discrete Fourier Transforms (DFTs) of the audio. The extracted features are used to train over a Deep Belief Network (DBN). A DBN is built out of multiple layers of Randomized Boltzmann Machines (RBMs), which makes the system a fully connected neural network. The same network is used for testing with a softmax layer at the end which serves as the classifier. This entire task of genre classification has been done with the Tzanetakis dataset and yielded a test accuracy of 74.6%.

Table of Contents

ABSTRACT III

LIST OF TABLES VI

LIST OF FIGURES VII

ACKNOWLEDGMENTS VIII

DEDICATION IX

CHAPTER 1 : INTRODUCTION 1

1.1 INTRODUCTION 1

1.2 LEARNING AUDIO FEATURES FROM DBN 1

1.3 RESTRICTED BOLTZMANN MACHINES (RBMS) 2

1.4 CONCLUSION 5

CHAPTER 2 : RELATED WORK 6

2.1 TEMPORAL POOLING AND MULTISCALE LEARNING FOR AUTOMATIC ANNOTATION 6

2.1.1 Audio Preprocessing *6*

2.1.2 Feature Pooling *7*

2.1.3 Pooled Features Classifier(PFC) and Multi-Time-Scale Learning Model (MTSL) *7*

2.1.4 Discussion *8*

2.2 BINARY CODING OF SPEECH SPECTROGRAMS USING A DEEP AUTO ENCODER 8

2.2.1 Constructing and learning an auto-encoder *9*

2.2.2 Experiments and inference *10*

CHAPTER 3 : MODEL 11

3.1 PROBLEM TO BE SOLVED 11

3.2 DATASET 11

3.3 FEATURE SELECTION 11

3.3.1 mfcc *12*

3.3.2 Spectral-centroid *12*

3.4 FEATURE ENGINEERING 12

3.5 TRAINING THE DBN 13

3.6 TESTING THE DBN 14

3.7 PERFORMANCE EVALUATION OF MFCC FEATURES 14

3.7.1 Training *14*

3.7.2 Testing *15*

3.8 RELATIONSHIP BETWEEN THEANO AND KERAS 15

CHAPTER 4 : GOOGLE CLOUD PLATFORM 16

4.1 REAL TIME PERFORMANCE 17

4.2 How to create a project and virtual machine in Google Cloud Platform? 17
4.3 How to connect to your instance through the GCP console? 19
4.4 How to connect to your instance through the local command terminal? 20
4.5 Useful terminal commands: 22

CHAPTER 5 : IMPLEMENTATION 24

5.1 Technology Stack 24
5.1.1 Technologies 24
5.1.2 Libraries 24
5.2 Model Structure 25
5.2.1 Checkpoint 25
5.2.2 src 26

BIBLIOGRAPHY 27

List of Tables

Table 3.1: Hyper-parameters and training statistics of DBN 13
Table 3.2: Output accuracies 14
Table 3.3: Training performance evaluation 15
Table 3.4: Testing performance evaluation 15
Table 4.1: Machine comparisons 17
Table 5.1: Technologies used 24
Table 5.2: Libraries used 25

List of Figures

Figure 1.1: Schematic representation of a DBN. Source [2] 2
Figure 1.2: Structure of a RBM 3
Figure 1.3: RBM input path 4
Figure 1.4: RBM with multiple inputs 4
Figure 2.1: Comparison of the PFC and the MTSL model. Source [3]. 8
Figure 2.2: Schema of a deep auto encoder. (Source: Google) 9
Figure 4.1: GCP Console 18
Figure 4.2: Create an instance 18
Figure 4.3: Instance configuration 19
Figure 4.4: Instance initialization 20
Figure 4.5: Account selection 20
Figure 4.6: Project selection 21
Figure 4.7: Region selection 21
Figure 4.8: ssh into instance 22
Figure 5.1: Project structure 25

Acknowledgments

I would like to thank **Dr. George Tzanetakis** for believing in me and giving me a chance to work with him.

I would like to thank my dear friend, brother, **Sri Raghu Malireddi** for his continuous support and mentorship.

Dedication

I dedicate this thesis to my late grandfather, whom I deeply miss. He would be the proudest person to see me what I am today.

For my parents, whose struggle, pain and love made me stand where I am today. My brother, Pavan has never left my side and is very special.

Sachin Tendulkar, there are a million things that made you my inspiration. But your perseverance, humility, dedication are the traits that inspire me the most. I've learnt from you, that **"Get hit, fall down, keep going"** is what makes a person successful.

Last but not the least, my best friends turned brothers, Saaketh, Sri Harsha and Venkat, always made me feel special. The confidence you three instilled in me is one of its kind. I've thoroughly enjoyed our association over the years and hope you know how important you three are to me. You three will always stay close to my heart and I love you with all that I have. Long live our friendship.

Chapter 1 : Introduction

1.1 Introduction

Deep learning, these days, is a successful approach to outperform past work and achieve state-of-the-art results across many areas. Recently these techniques are being adopted in the field of Music Information Retrieval (MIR) to accomplish tasks in audio feature extraction, music generation, audio annotation, genre classification etc. Feature learning consists in exploiting the structure and pattern of the data distribution to construct a new representation of the input. Efficient feature extraction and learning allows one to build a system relying less on prior knowledge and more on data, which grants more flexibility to adapt to a given task. In the next few sections of this chapter, let's discuss about the past work related to feature extraction, classification, auto tagging and different deep learning techniques used to solve the above mentioned tasks to obtain state-of-art results.

1.2 Learning audio features from DBN

Audio feature extraction plays a key role in almost all of the music information retrieval (MIR) tasks. Out of many useful musical features like statistical, spectral, timbral, temporal, etc. it is not evident which feature is appropriate for a given task. This problem has given rise to the need of a system which extracts relevant audio features automatically, given the task. In [2], it is reported that the extracted features performed favourably against the MFCC features. Apparently all audio based music genre classification models use different types of acoustic features [2, 5]. For genre classification, the best reported accuracy on the Tzanetakis dataset [5] was achieved by a system that used auditory cortical representations of music recordings and sparse representation-based classifiers. But it is difficult to know whether the acoustic features or the machine learning techniques that are responsible for the success. Auto tagging is another MIR task which is closely related to the genre classification. From [6], it is clear that timbral and temporal features are often used to solve this challenge, just as genre classification. To validate the developed model,

the learned features are applied on the Tzanetakis dataset for the task of genre classification and Majorminer dataset for autotagging.

In this approach a Deep Belief Network (DBN) [7] is used to learn feature representation. DBNs have already proved their worth in the field of MIR. The DBN is a neural network constructed from multiple layers of Restricted Boltzmann Machines (RBMs) [2]. In this experiment, the DBN training is a two-step process; first pre-trained with a training set in an unsupervised manner and then supervised fine-tuning using the same training set. Once trained, the activations of the DBN are used as the learned representations of the input audio. Then a non-linear Support Vector Machine (SVM) is used for both the tasks of genre classification and autotagging.

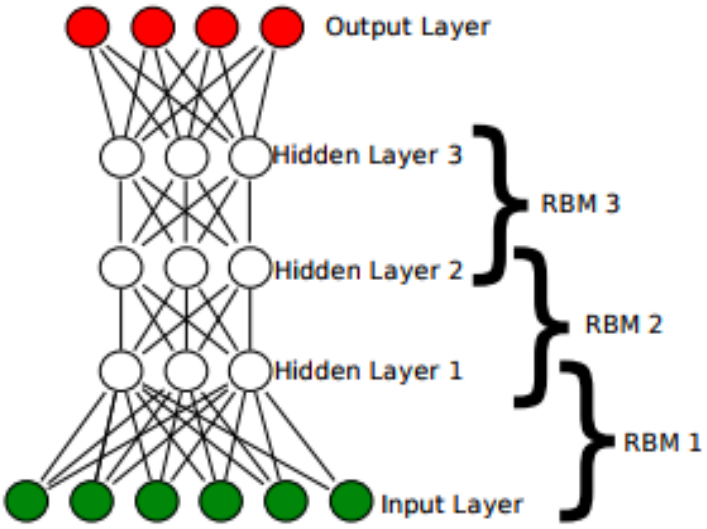

Figure 1.1: Schematic representation of a DBN. Source [2]

1.3 Restricted Boltzmann Machines (RBMs)

Restricted Boltzmann Machines (RBMs) are neural networks that belong to so called Energy Based Models. It is an algorithm that can be used for dimensionality reduction, classification, regression, feature learning and topic modeling. RBMs are shallow, two layer neural networks that constitute the building blocks of deep-belief networks.

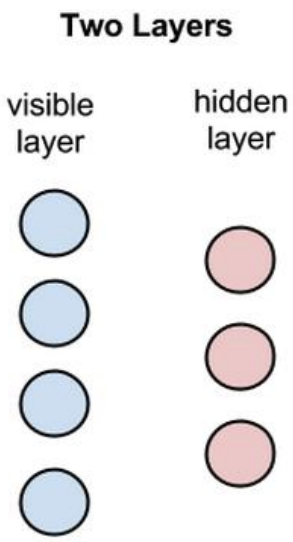

Figure 1.2: Structure of a RBM

Each circle in the above picture acts as a neuron and is called a node. And nodes are the units where the calculations take place. Note that there is no intra layer communication, which means the nodes are connected to each other across layers, but no two nodes of the same layer are connected. Each node processes the received input and makes stochastic decisions about whether to transmit that input or not. Each visible node takes a low-level feature from an item in the dataset to be learned. For example, from a dataset of audio files, each visible node would receive one frequency value for each frame in one audio file. Now that frequency value, X, went through node 1 of the hidden layer, X is multiplied by a weight and added to a so-called bias. The result of these two operations is passed to the activation function, which produces the node's output, given input X.

activation f ((weight w * input X) + bias b) = output a

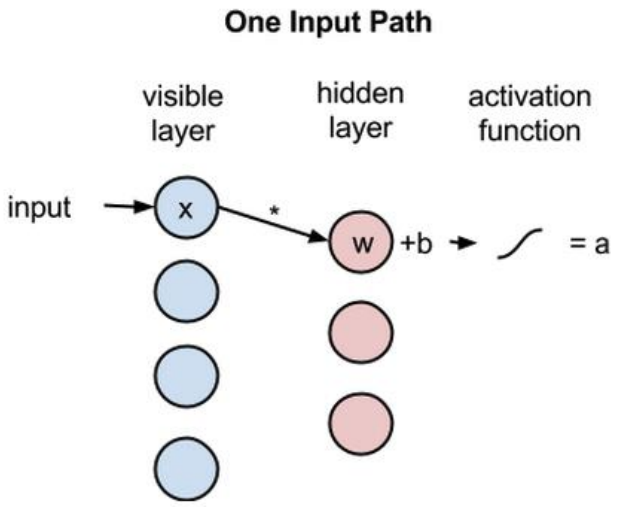

Figure 1.3: RBM input path

Now let's look at multiple inputs to multiple hidden layers. In the below network, at each hidden node, each input X is multiplied by its respective weight w. That is, a single input X would have three weights here, making 12 weights altogether (4 input nodes * 3 hidden nodes). Each hidden receives four inputs multiplied by their respective weights. The sum of those products is again added to the bias, and the result is passed through the activation algorithm producing one output for each hidden node.

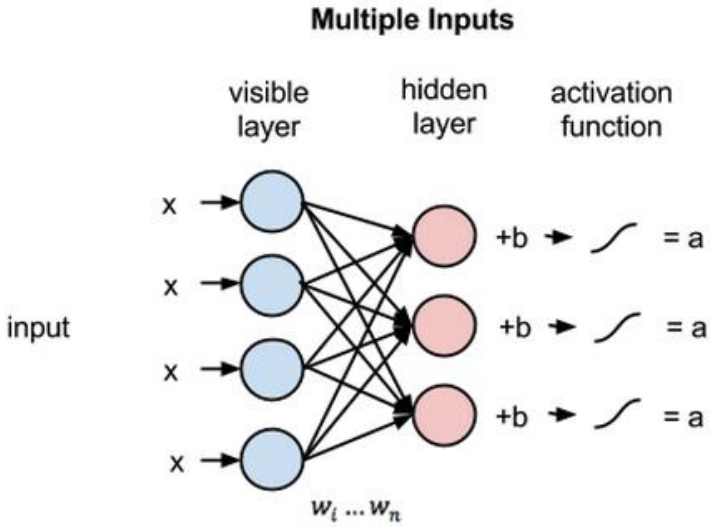

Figure 1.4: RBM with multiple inputs

If a neural network has a RBM internally, the output of the first hidden layer is passed as an input to the next hidden layer, and so on through as many layers as you like until they reach a final classifying layer.

1.4 Conclusion

The previous work has proposed a model where the relevant features are automatically extracted. And the activations from the DBN are used as the inputs for a non-linear Support Vector Machine (SVM). The work also suggested, it is almost impossible to train deep neural networks using gradient descent. However, deep neural networks (i.e. networks having multiple hidden layers) are no more difficult or impossible to train using stochastic gradient descent. And it is fairly possible to outperform the DBN system using an end to end deep neural network to solve the task of both genre classification and auto tagging. To speed up the inference process, I extracted the weights and biases form the Theano's RBM and used them in Keras based fully connected neural network. This end to end deep learning model would also provide the advantage of less computation time to train the network.

Chapter 2 : Related Work

2.1 Temporal pooling and multiscale learning for automatic annotation

Quality of the audio features is the key for any audio related task. Although MFCCs have shown their worth in speech analysis, [3] concludes that there are better features for music audio analysis. Recent work shows that features like mel-scaled energy bands, octave-based spectral contrast features ha ve outperformed MFCCs. Thus, finding optimal features for audio classification, auto tagging remained an open problem.

Automatic annotation is about assigning relevant word descriptors or tags to a given music audio. These tags could be anything like the genre, artist, album, instrumentation, etc. Finding an audio clip that is as relevant as possible to the given tag or a set of tags is called ranking. This information is highly useful for tasks like music recommendation, playlist generation or creation and quantify the music similarity. [3] talks about a system that extracts the relevant audio features automatically. Then, the impact of selection and the mixing of pooling functions is studied for temporal summarization. Once the features are preprocessed, they are summarized over time and used for training a classifier which gives us the tag affinities. The objective of summarizing is to transform a joint feature representation into a useful one that preserves important information while discarding the noise and redundancy. It is shown that combining several pooling functions improved the performance of the system. Finally, the feature learning, time summarization and classification steps are integrated in one deep learning system. Putting up all these together a state-of-art performance has been achieved on the Magnatagatune dataset.

2.1.1 Audio Preprocessing

To preprocess the audio, a three step process is followed. Firstly, to transform the audio in the spectral domain, DFTs over windows of 1024samples on the given audio at 22.1KHz is computed with a frame of 512 samples. Then, to obtain a set of spectral energy

bands, spectral amplitudes are computed through a set of 128 mel-scaled triangular filters. Finally, PCA is applied on these random subsamples of the training dataset. These preprocessed audio features are referred as Principal Mel-Spectrum Components (PMSC).

2.1.2 Feature Pooling

Feature pooling is one of the most straightforward ways to summarize features. It means extracting simple statistics such as mean or maximum of the features over a given time length. Time and again it is proved that selection of features is the key aspect of any MIR related task. Similarly, selection of the pooling function is also an important aspect for feature pooling. Not only the pooling function, but the choice of the temporal scale at which the pooling is applied also has a great impact on the system's performance. Longer time scales may discard too much information and shorter time scales may make the representation less compact. Hence an onset detection is used to set an optimized window length.

2.1.3 Pooled Features Classifier(PFC) and Multi-Time-Scale Learning Model (MTSL)

PFC and MTSL are the two models designed to carry out the experiments in this work. PFC is a conventional system that applies feature extraction, pooling and classification in three separate steps. MTSL is for learning both before and after the temporal pooling.

In detail, PFC applies the chosen section of pooling functions to the PMSC features and sends the pooled features to the classifier. The classifier is a single hidden layer, with 1000 units, neural network which is also known as a multi-layer perceptron (MLP). Each pooling window is considered as a training example and the predictions of the classifier are averaged over all the windows of the given audio to obtain the final classification.

On the other hand, MTSL is structurally similar to the PFC, except for the fact that a hidden layer is added between the input features and the pooling function. This makes the pooling applied on the activations of the new hidden layer. While the first layer is

learning on frames at a time scale of 46ms, the second layer works at the scale of pooling window. In other words, this model is hence called as a multi-time-scale learning as it learns on different time scales.

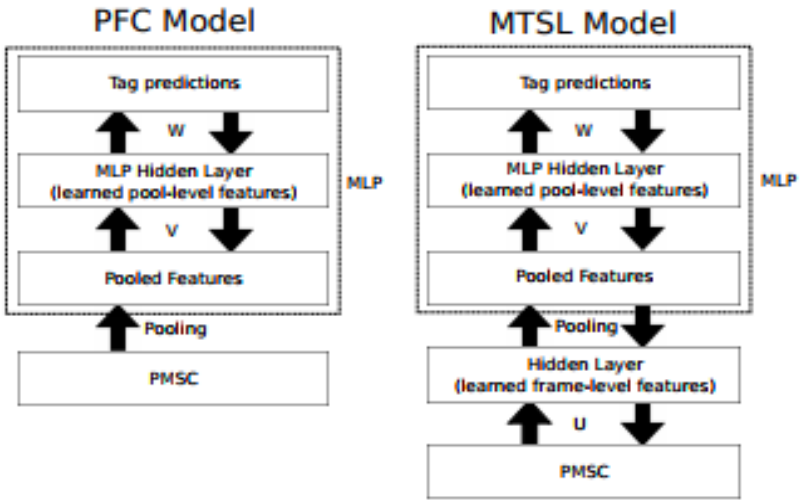

Figure 2.1: Comparison of the PFC and the MTSL model. Source [3].

2.1.4 Discussion

Results prove that Mel-spectrum features perform better than MFCCs and adding the PCA whitening further enhance and polish the performance as well as reducing the training time. It is also seen that combining pooling functions gives better classification rate. Although the MTSL model helped to achieve the state-of-the art performance, a deeper model with more time scales and smaller pooling windows might give a scope of much more improved results.

2.2 Binary coding of speech spectrograms using a deep auto encoder

Binary codes make the computation faster and are also easy to store since they are compact. [1] focuses on encoding of speech spectrograms using a deep generative model, which uses a log power spectrum instead of cepstra to capture greater fine details of the speech signal. With recent advancement in multilayer neural networks, machine learning problems like information retrieval, encoding, classification etc. have become less challenging. The work reported in [1] highlights the use of deep auto-encoders for

dimensionality reduction and further discovery of efficient binary codes in information retrieval. These auto-encoder generated codes are then compared against the discrete speech codes developed by using the traditional vector quantization (VQ) techniques. The potential benefits and results of using discrete representations of the audio are being conveyed as a conclusion to this work.

2.2.1 Constructing and learning an auto-encoder

A deep auto encoder is comprised of two symmetrical deep belief networks (DBN) that typically have a few allow layers, with first half representing the encoding half of the net and the last half that makes up the decoding half of the net. The layers are restricted Boltzmann machines (RBM), which are the building blocks of a DBN. Below is the schema of a deep auto-encoder's structure.

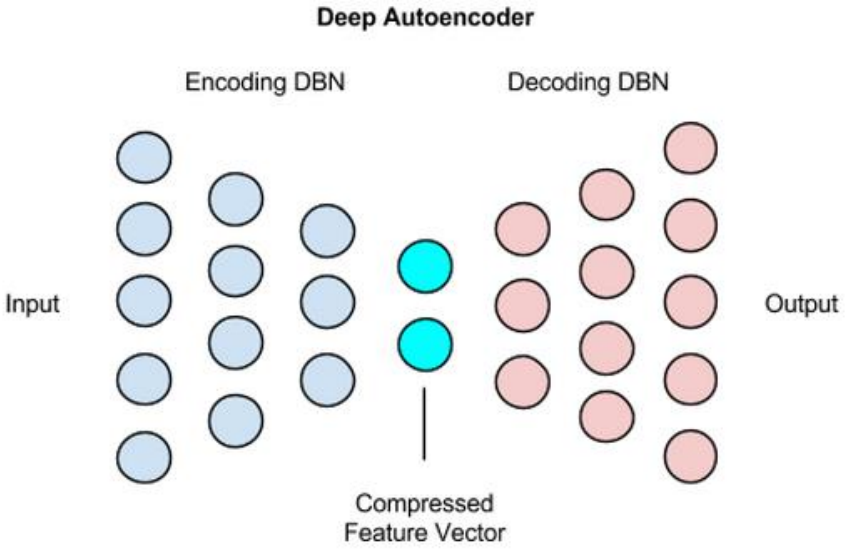

Figure 2.2: Schema of a deep auto encoder. (Source: Google)

The experiment is broken down into two parts. First task is to learn a Gaussian-binary restricted Boltzmann machine (RBM) which consists of one visible of linear variables with Gaussian noise one hidden layer of 500 to 3000 binary latent variables. The connection weights can be learned efficiently using the contrastive divergence approximation to log likelihood gradient. In the second task, the activation probabilities of

the Gaussian-binary RBM are treated as the training data for the binary-binary RBM. The Gaussian-binary and binary-binary RBMs are then composed to form a deep belief network (DBN). The resultant binary codes from the hidden units of the second RBM achieved lower distortion and the distortion can further be lowered with another step of fine tuning.

Fine tuning can be performed by unrolling the DBN by its weight matrices, to create a deep five-layer network, which is an auto-encoder. The lower layers of the network use the matrices to encode the input whereas the upper layers use the matrices to decode the input. The auto-encoder then fine-tuned using the backpropagation of error derivatives.

Once the pre training and fine tuning process is completed, the spectrogram is encoded. First, N consecutive overlapping frames of 256-point log power spectra are normalized to zero mean and unit variance to provide the input to the auto-encoder. The first hidden layer then uses the logistic function to compute the real valued activations. These activations are fed to next layer, which is the coding layer, to compute codes. These binary codes are then used to reconstruct the original spectrogram. Finally, the overlap-and-add method is used to reconstruct the full length speech spectrogram, by applying the auto-encoder to every possible window of N consecutive frames.

2.2.2 Experiments and inference

The use of log power spectra has achieved good results over the TIMIT's dataset, which is comprised of both male and female voices. After examining the results measured by the standard log spectral distortion, the outcomes turned encouraging. Even without the fine tuning as an auto-encoder, the DBN architecture gives lower distortion than the traditional VQ coder. Also, with the use of fine tuning as an auto-encoder, the distortion is reduced to two folds compared with just pre training the DBN. It is concluded that the performance of the deep auto-encoder relies on the number of the hidden units. Also, when ran on a GPU, these networks tend to give improved results since the computation on CPU is partly slower.

Chapter 3 : Model

3.1 Problem to be solved

One of the things we, humans, are particularly good at is classifying songs. Within the first few seconds we can identify whether we're listening to rock, rap, pop, classical etc. However, a major chunk of us still live with uncategorized music libraries. And we cannot afford to manually label the songs, in our digital library, which will easily go up to hours or even days of work. In this work, we'll see how deep learning comes to the rescue in genre classification.

Classifying music files according to their genres is a challenging task in Music Information Retrieval (MIR). For any given machine learning task, the most important steps are feature extraction and selection. Feature extraction methods helps you in your mission to create an accurate predictive model. So, extracting relevant features is the first hurdle in achieving the task of genre classification. In the next few sections, the solution and implementation of the model is discussed.

3.2 Dataset

The "Tzanetakis dataset" was used from the well-known paper [5]. The dataset consists of 1000 30-second audio clips which are categorized in to 10 musical genres. Each genre holds 100 audio clips. All the clips are 22050Hz mono 16-bit audio files. This dataset has been used as a reference for the task of genre recognition ever since it was introduced.

3.3 Feature selection

Spectral features are the frequency based features that are obtained by converting the time based signal into the frequency domain using the Fourier Transform. These features can be used to identify the notes, pitch, rhythm and melody. In the further sections, let's discuss about the most infamous features in audio related tasks.

3.3.1 mfcc

Mel-Frequency Cepstral Coefficients (MFCCs) are the most widely used frame level features for audio related MIR tasks. They take advantage of source/filter deconvolution from the cepstral transform and perceptually realistic compression of spectra from the Mel pitch scale. But the first few MFCC values capture pitch-invariant timbral characteristics of the audio, they are commonly used in tasks where it is useful to generalize across pitch, such as multi-speaker speech recognition and musical timbre recognition [2].

3.3.2 Spectral-centroid

Spectral centroid is a measure of brightness of the sound. This measure is obtained by evaluating the centre of mass of the spectrum. The individual centroid of a spectral frame is defined as the average frequency weighted by amplitudes divided by the sum of the amplitudes. [16] reports, it is not sure of it the spectral centroid would be useful for classifying different genres of the music, which are mixed sounds. Also the spectral centroid features didn't seem to imply anything for genre classification and it only depends on what kinds of instruments are playing in time.

Having discussed the flaws of the other popular audio features, I present a system that extract features based on fairly low level representation i.e., the magnitude spectrum of the Discrete Fourier Transform of the audio samples. In the end I will show that these learned features will perform better than the MFCCs.

3.4 Feature engineering

A song is nothing but a very long series of data. The audio was divided into short frames of 46.44ms which is nothing but 1024 samples at 22050Hz sampling rate. For each of these frames the discrete Fourier transform (DFT) was calculated to convert audio data into frequency domain. This allows for a much simpler representation of the data. To avoid the redundancy in the DFT, only the absolute values are considered and used as the inputs with dimension 513. All the extracted features are saved as pickle files according to the respective genres. To learn the representations, the dataset was split into 50%, 20% and

30% for training, validation and testing respectively. For better reproducibility, a random seed was set for splitting the data.

3.5 Training the DBN

The DBN was first pre-trained with the training set. The DBN was built with 513 inputs, 3 hidden layers with each consisting of 50 units and 10 output units. After the fine tuning, the model was saved and then proceeded to fine-tuning using the same training set and using the validation set to do early stopping. For this training I have used Python's Theano library. During this process of training, to speed up the inference step, I extracted the weights and biases from Theano's RBM which were used for testing. In this entire process of training, I have tried wide range of hyper-parameters to find the best combination and chose the model with the best validation error. The chosen DBN model is below described in the table.

Number of hidden layers	3
Units per layer	50
Length of input vector	513
Fine-tuning learning rate	0.1
Pre-training learning rate	0.001
Pre-training epochs	10
Training epochs	474
Total training time (hours)	3

Table 3.1: Hyper-parameters and training statistics of DBN

3.6 Testing the DBN

For testing the DBN, I have used the weights and biases that were extracted from the Theano's DBN and passed them to the Keras based DBN. Once trained, we can use the activations of the DBN hidden units as a learned representation of the input audio. As the last step, a softmax classifier is used to calculate the classification accuracy. The softmax classifier is a generalization of binary form of logistic regression. This function assigns decimal probabilities to each genre and these decimal probabilities must add up to 1.0. And this function is implemented through the DBN layer just before the output layer. Also, the softmax layer must have the same number of nodes as the output layer. Below is the table with the summarized accuracies.

Training accuracy	0.97
Validation accuracy	0.735
Testing accuracy	0.746

Table 3.2: Output accuracies

3.7 Performance evaluation of MFCC features

Below table is the comparison of the performance of the mfcc features versus the learned low level DFT features. It can be seen that the learned frame level features compare favorably against MFCCs.

3.7.1 Training

Features	DFT	MFCC
Units per layer	50	50
Length of input vector	513	20
Fine-tuning learning rate	0.1	0.1

Pre-training learning rate	0.001	0.001
Pre-training epochs	10	10
Training epochs	474	474
Total training time (hours)	3	2.5

Table 3.3: Training performance evaluation

3.7.2 Testing

Features	DFT	MFCC
Training accuracy	0.97	0.81
Validation accuracy	0.735	0.629
Testing accuracy	0.746	0.603

Table 3.4: Testing performance evaluation

3.8 Relationship between Theano and Keras

Theano is a Python library mainly used for deep learning. In [2], the DBNs were implemented using Theano. Since [2] is the inspiration for this project, I have trained the RBMs in unsupervised and DBN in supervised fashion. And in order to speed up the inference step, I have used the saved weights and biases from DBN and passed them to Keras based DBN. Keras is a light weight machine learning library that uses TensorFlow in the back end.

Chapter 4 : Google Cloud Platform

Google compute engine lets users create and run virtual machines on Google's cloud infrastructure. Google compute offers scale, performance and value that allows you to easily launch large compute clusters on Google's infrastructure. There are no upfront investments and users can run thousands of virtual machines on a system that has been designed to be fast, and to offer strong consistency of performance. Compute engine is three things, virtual machines, networking and disks. Inside the projects we have all of these three components. Network is the same network that google uses to serve all of its products. And the project is a private project. Virtual machines are the core of the compute engine. Whereas persistent disk is a logical block storage device. Compute engine is an Infrastructure as a Service (IaaS) service. An IaaS is a form of cloud computing that provides virtualized computing resources over the internet.

The experiments mentioned in the previous sections were conducted in a GPU enabled virtual machine provided by Google Cloud Platform. I have chosen a Linux, Ubuntu machine with configuration of 4 virtual CPUs of 15 GB with a hard disk capacity of 64 GB. And the machine is also empowered with 1 Tesla K80 GPU. The GPU was used in the inference step by Keras library, which runs on top of TensorFlow library. Other libraries like NumPy, LibRosa, Pickle, Theano etc. are installed regularly in the virtual machine. Once the environment is set, you are good to go.

I have opted for Google Compute Engine over Google ML engine since Compute Engine is more cost efficient and the user has more control over the libraries and the package installation during the environment set up.

The entire computation has been tremendously hastened with GCP when compared to the performing these experiments on a local machine.

Component	Local Machine	GCP VM
CPU	1	4 VCPUs
Cores	Dual (2)	32
GPU	Intel	NVIDIA Tesla K-80
Training time	11 hours (approx.)	3 hours

Table 4.1: Machine comparisons

4.1 Real time performance

Although, the inspiration of this project, [2] hasn't talked about the machine configuration on which their experiments were run, they reported that the training time is approximately 104 hours. Whereas with the above configuration for my virtual machine instance, I could finish the entire training in just 3 hours approximately. The real time inference computation results on CPU took less than 10ms whereas on a GPU it is even better, which is less than 5ms. With this kind of performance, this model can be implemented as a mobile application, since most of the mobiles these days come with robust configuration.

4.2 How to create a project and virtual machine in google cloud platform?

This section explains how to create a GCP project, a Linux virtual machine instance in Google compute engine using the GCP console. Below are the steps.

- *Step1:* Login to the GCP's console and select Compute Engine from the navigation menu at the left side of the console.

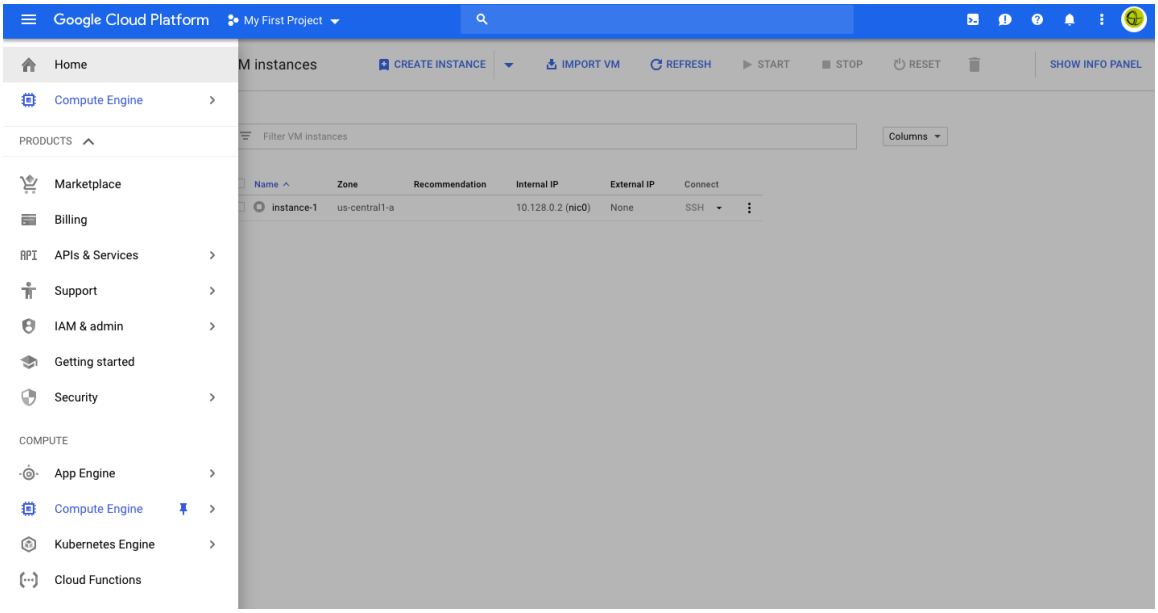

Figure 4.1: GCP Console

- *Step 2:* Under the compute engine's menu, select VM instances and click on create instance.

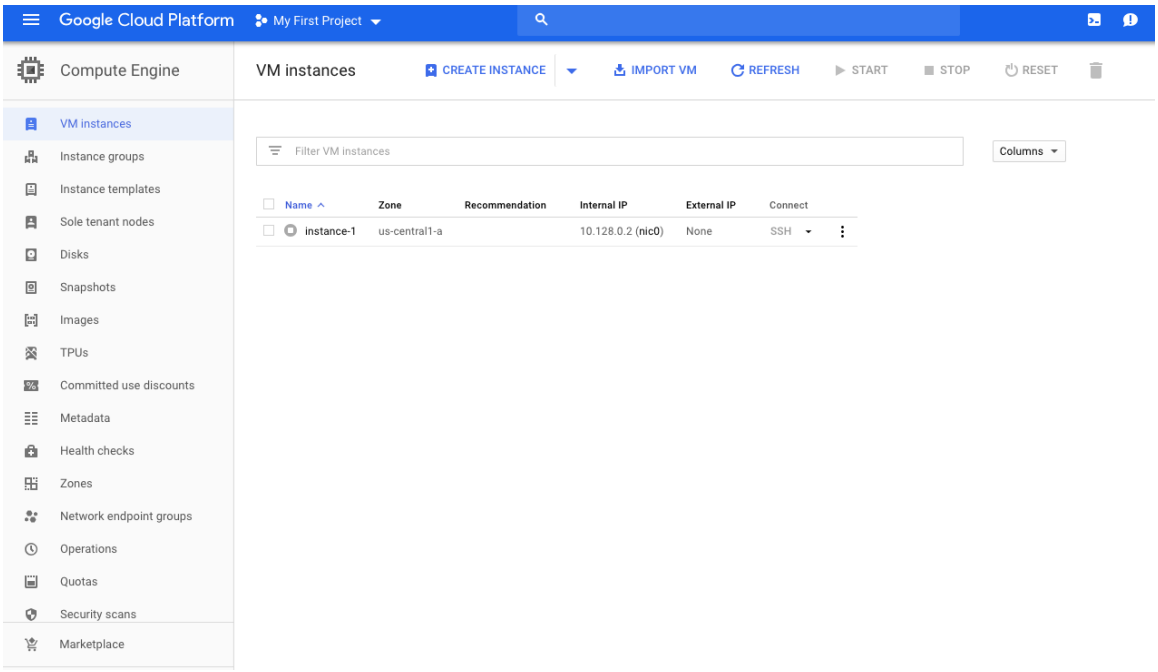

Figure 4.2: Create an instance

- *Step 3:* Choose the configuration for your VM instance and hit the create button at the bottom. If your machine needs a GPU, make sure you select the region and zone that offers GPU. Not all of them have GPUs available.

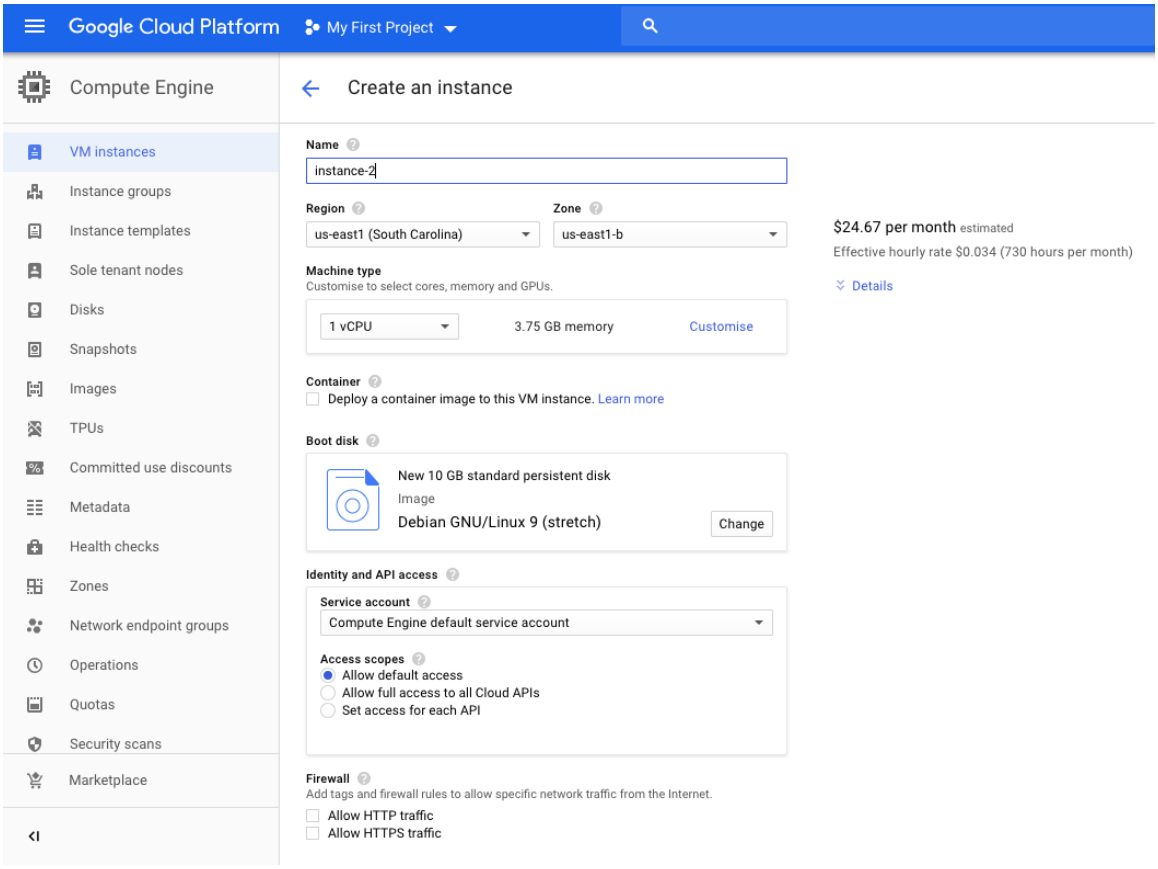

Figure 4.3: Instance configuration

Once the instance has been created, allow a short time for it to start the instance. Once ready, it'll be listed on the VM instance page with a green status icon.

4.3 How to connect to your instance through the GCP console?

- In the GCP console, go compute engine and then to the VM instances page.
- In the list of virtual machine instances, click on SSH in the row of instance. Make sure the instance is started.

4.4 How to connect to your instance through the local command terminal?

- *Step 1:* Download and install the Google cloud sdk on your local machine.
- *Step 2:* Initialize the configuration with command “gcloud init”

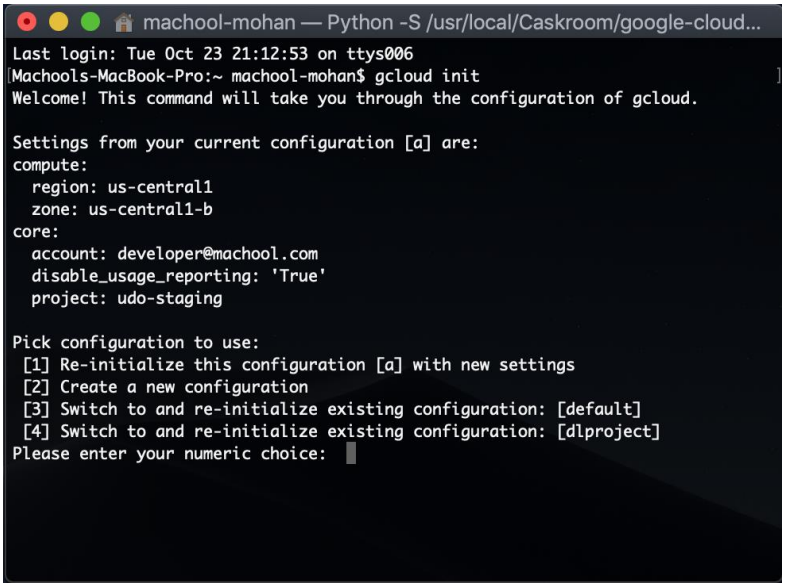

Figure 4.4: Instance initialization

- *Step 3:* Once you initialize the configuration, next step is to choose your account.

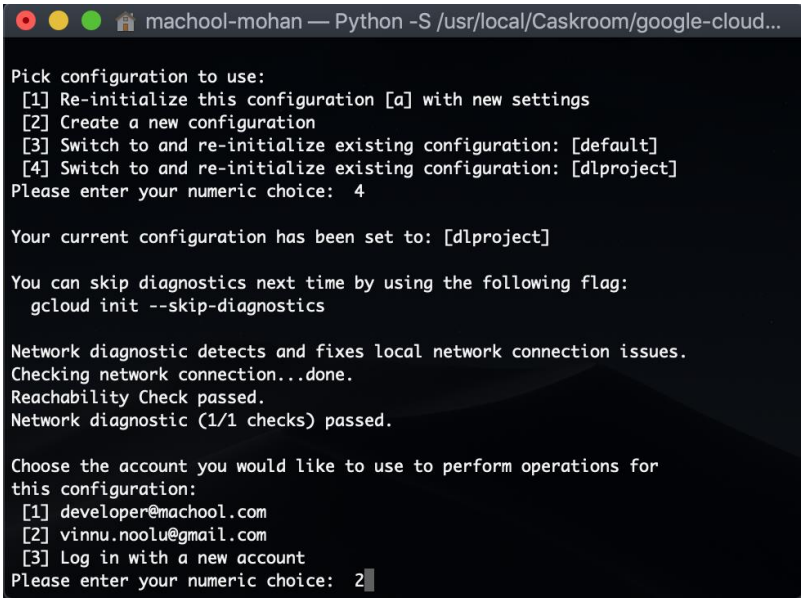

Figure 4.5: Account selection

- *Step 4:* Select your project

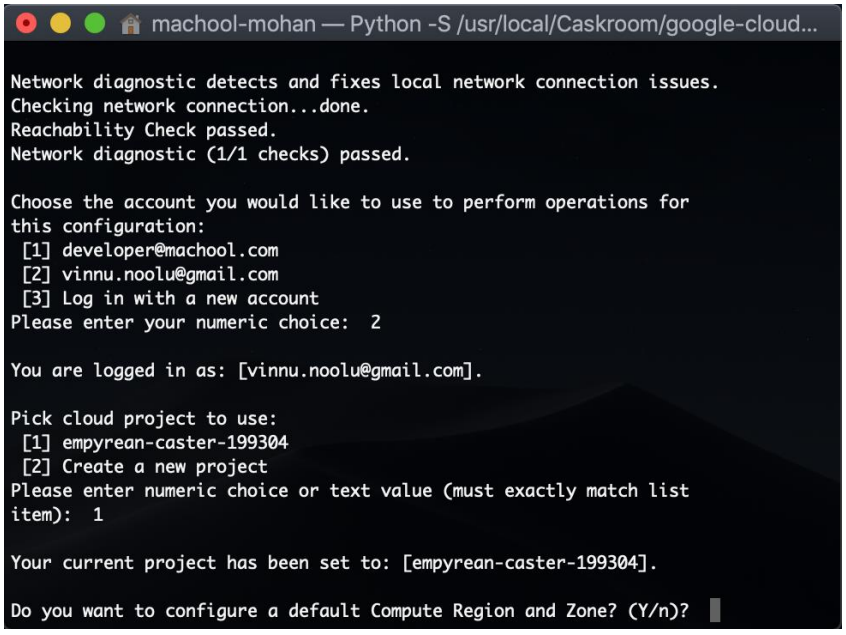

Figure 4.6: Project selection

- *Step 5:* Select the region where your VM is set up.

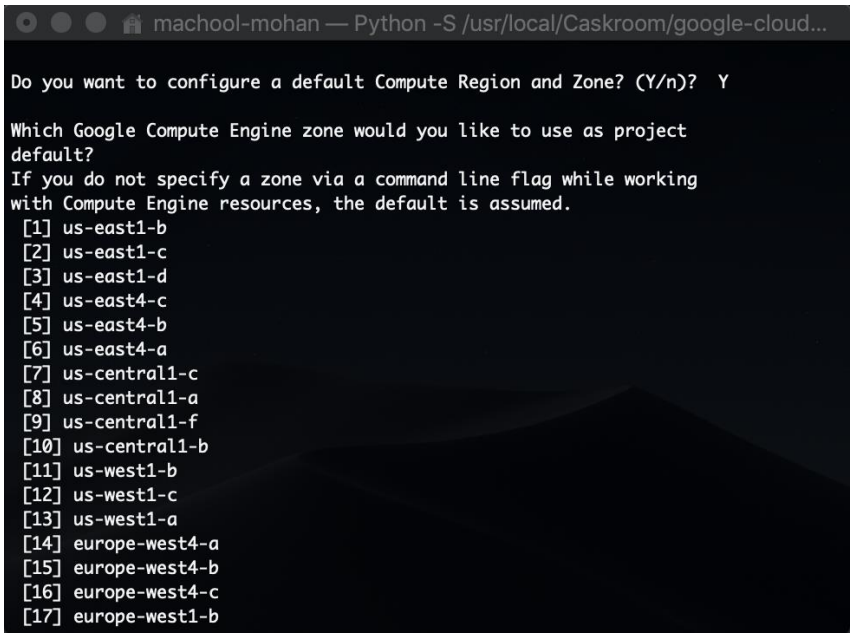

Figure 4.7: Region selection

- *Step 6:* Finally, ssh into your VM to get access to the files. Command is "gcloud compute ssh <name of the instance> --zone <region>"

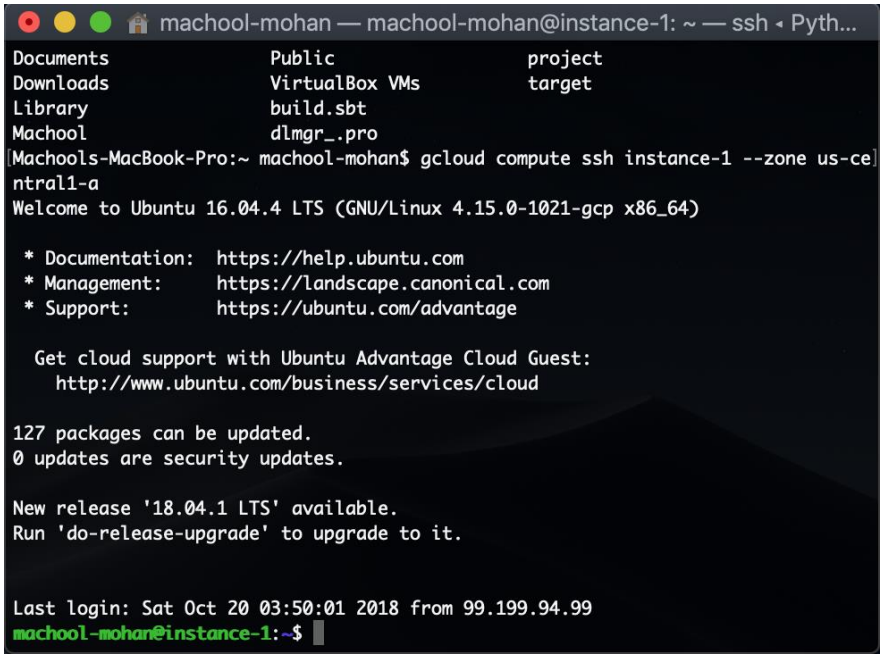

Figure 4.8: ssh into instance

4.5 Useful terminal commands:

1. Initialize an instance configuration.
   - gcloud init

2. SSH into the instance.
   - gcloud compute ssh <name of the instance> --zone <region>

3. Copy files from local to google cloud.
   - gcloud compute scp --recurse <file or directory>/ <instance name>:./<destination dir>

4. Copy files from google cloud to local.

- gcloud compute scp --recurse <instance name>:./<file or folder>/ ./<destination dir>

Chapter 5 : Implementation

5.1 Technology Stack

This section briefly describes the programming language and a set of libraries that are used for the implementation.

5.1.1 Technologies

Technology	Description
Python 3	Python 3.0 a.k.a Python 3000 or Py3K is a new version of language that is incompatible with 2.x line of releases. The language is mostly the same, but many details, especially how built-in-objects like dictionaries and strings work, have changed considerably and a lot of deprecated features have been finally removed.
Keras	Keras is a high level neural networks API, written in Python and is capable of running on top of TensorFlow and Theano. It was developed with a focus on enabling fast implementation. It runs seamlessly on both CPU and GPU.

Table 5.1: Technologies used

5.1.2 Libraries

Library	Description
Theano	Theano is a Python library that allows you to define, optimize and evaluate mathematical expressions involving multi-dimensional arrays efficiently. It preforms data intensive operations much faster on a GPU than on a CPU.
NumPy	NumPy is a fundamental package for scientific computing with Python.
Pickle	The pickle module implements a fundamental but powerful algorithm for serializing and de-serializing a python object structure. Pickling is a process where a python object hierarchy is converted into a byte stream and un-pickling is the inverse operation.

LibROSA	LibROSA is a python package for music and audio analysis. It provides the building blocks necessary to create music information retrieval systems.

Table 5.2: Libraries used

5.2 Model Structure

I have segregated the classes as per their own functionalities into 8 python files, details of which are given in the following subsections.

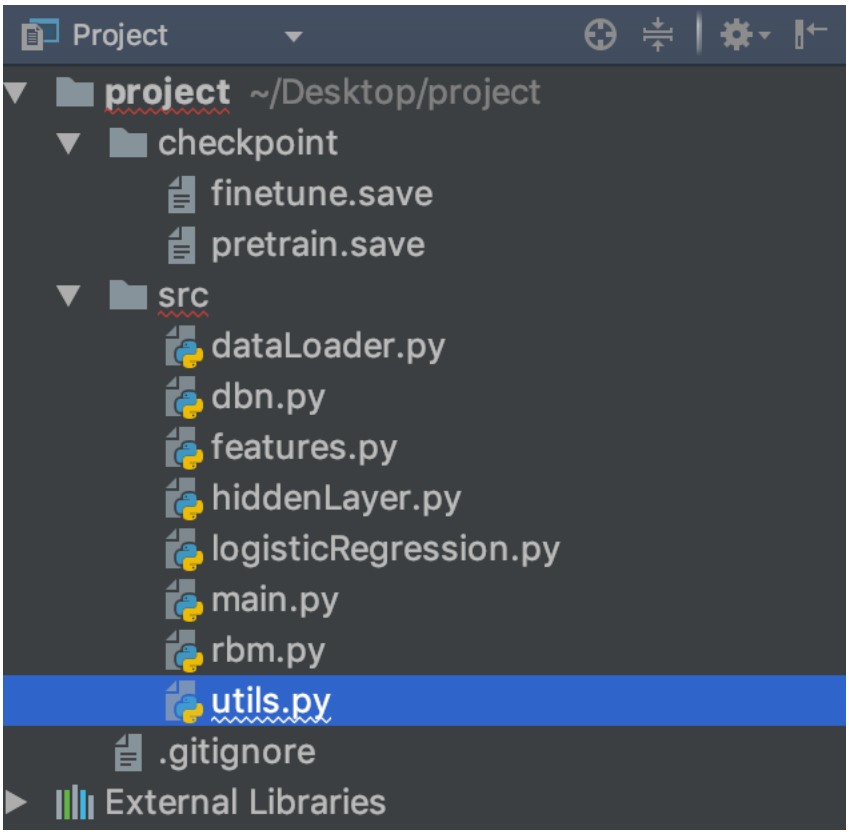

Figure 5.1: Project structure

5.2.1 Checkpoint

Pretrain.save:

This is a pickle file of the saved mode. The training was done by initializing the weights randomly. As soon as the training is started, the weights are changed in order to perform the task with less mistakes (i.e. optimization). Once we achieve satisfying training results,

the weights are saved. Pre-training gives the network a head start, as if it has seen the data before.

Finetune.save:

This pickle file is the saved model after fine tuning. Instead of repeating what was done in the first network and start from training with randomly initialized weights, I have used the saved weights from previous networks as the initial weights for the new model.

5.2.2 src

dataLoader:

This class is responsible for loading the audio data i.e. Tzanetakis dataset.

dbn:

This class is made to support a variable number of layers. A deep belief network is obtained by stacking several RBMs on top of each other. The hidden layer of RBM at layer 'I' becomes the input of the RBM at layer 'i+1'. The first layer RBM gets as input, the input of the network and the hidden layer of the last RBM represents the output. Number of input layers are 784, whereas output layers are just 10. Number of hidden layers are 500.

features:

This script pre-processes the Tzanetaki's dataset and generates the [X, y] labels required for any classification purposes.

main:

This file has the heart of the implementation. It has functions to test and train the dbn. Also hold the section of code where the pre-training and fine tuning happens.

rbm:

This rbm constructor defines the parameters of the model along with basic operations for inferring hidden from visible as well as for performing contrastive divergence.

Bibliography

[1] Deng, Li, et al. "Binary coding of speech spectrograms using a deep auto-encoder." Eleventh Annual Conference of the International Speech Communication Association. 2010.

[2] Hamel, Philippe, and Douglas Eck. "Learning Features from Music Audio with Deep Belief Networks." ISMIR. 2010.

[3] Hamel, Philippe, et al. "Temporal Pooling and Multiscale Learning for Automatic Annotation and Ranking of Music Audio." ISMIR. 2011.

[4] Bergstra, James, Michael I. Mandel, and Douglas Eck. "Scalable Genre and Tag Prediction with Spectral Covariance." ISMIR. 2010.

[5] Tzanetakis, George, and Perry Cook. "Musical genre classification of audio signals." IEEE Transactions on speech and audio processing 10.5 (2002): 293-302.

[6] Bertin-Mahieux, Thierry, Douglas Eck, and Michael Mandel. "Automatic tagging of audio: The state-of-the-art." Machine audition: Principles, algorithms and systems (2010): 334-352.

[7] Hinton, Geoffrey E., Simon Osindero, and Yee-Whye Teh. "A fast learning algorithm for deep belief nets." *Neural computation* 18.7 (2006): 1527-1554.

[8] Google cloud compute engine documentation.

[9] Pickle documentation. https://docs.python.org/2/library/pickle.html

[10] Theano documentation. http://deeplearning.net/software/theano/

[11] LibROSA documentation. https://librosa.github.io/librosa/

[12] NumPy documentation. http://www.numpy.org/

[13] Keras documentation. https://keras.io/

[14] Theano: A Python framework for fast computation of Mathematical expressions.

[15] A beginner's guide to RBMs. https://skymind.ai

[16] Spectral-Centroid guide: https://ccrma.stanford.edu/~unjung/AIR/areaExam