MUSIC GENRE CLASSIFICATION USING DEEP LEARNING

MAJOR PROJECT REPORT

Submitted in partial fulfilment of the requirements for the award of the degree of

[redacted]

in

COMPUTER SCIENCE & ENGINEERING

by

[redacted] [redacted] [redacted] [redacted]

[redacted] [redacted] [redacted] [redacted]

Guided by

[redacted]

Assistant Professor, CSE Department

[redacted]

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

[redacted]

[redacted]

[redacted]

JUNE 2018

CANDIDATE'S DECLARATION

It is hereby certified that the work which is being presented in the B. Tech Major Project Report entitled **"MUSIC GENRE CLASSIFICATION USING DEEP LEARNING"** in partial fulfilment of the requirements for the award of the degree of [redacted] and submitted in the **Department of Computer Science & Engineering** of [redacted] [redacted] is an authentic record of our own work carried out during a period from **January 2017 to April 2017** under the guidance of [redacted]**, Department Of Computer Science and Engineering.**

The matter presented in the [redacted] has not been submitted by us for the award of any other degree of this or any other Institute.

[redacted] [redacted] [redacted] [redacted]

[redacted] [redacted] [redacted] [redacted]

This is to certify that the above statement made by the candidate(s) is correct to the best of my knowledge. They are permitted to appear in the External Minor Project Examination.

[redacted]
Assistant Professor, CSE

[redacted]
Head of Department, CSE

The [redacted] Viva-Voce Examination of [redacted] has been held on ……………………………….

[redacted]
Project Coordinator

(Signature of External Examiner)

ABSTRACT

This project was primarily aimed to create an automated system for classification model for music genres. Jim Samson in his article "Genre" defines music genre as "music genre is a conventional category that identifies pieces of music as belonging to a shared tradition or set of conventions." The term "genre" is a subject to interpretation and it is often the case that genres may very fuzzy in their definition. Further, genres do not always have sound music theoretic foundations, e.g. - Indian genres are geographically defined, Baroque is classical music genre based on time period. Despite the lack of a standard criteria for defining genres, the classification of music based on genres is one of the broadest and most widely used. Genre usually assumes high weight in music recommender systems. Genre classification, till now, had been done manually by appending it to metadata of audio files or including it in album info. This project however aims at content-based classification, focusing on information within the audio rather than extraneously appended information.

In our project we make use of various Deep Learning Models like Artificial Neural Networks (ANN) and Convolutional Neural Networks (CNN) to classify the genre of the music and then compare our results with the traditional Machine Learning Techniques like KNN, K-Means Clustering, Decision Trees and Random Forest.

ACKNOWLEDGEMENT

We express our deepest gratitude to [redacted]**, Asst. Professor, Department of Computer Science & Engineering** for her valuable guidance and suggestion throughout our project work. We are thankful to [redacted]**,** Project Coordinator for his valuable guidance.

We would like to extend our sincere thanks to [redacted]**, Head of the Department,** for his time to time suggestions to complete our project work. We are also thankful to [redacted] [redacted] **Director** for providing us the facilities to carry out our project work.

[redacted] [redacted] [redacted] [redacted]

[redacted] [redacted] [redacted] [redacted]

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATION

Abbreviation	**Full form**
SVM	Support Vector Machine
kNN	k Nearest Neighbors
MFCC	Mel-frequency cepstral coefficients

1. Introduction

1.1 What is Machine Learning?

Machine learning is when a computer uses an algorithm to understand the data it has been given and recognizes patterns. We call the process of learning “training,” and the output that this process produces is called a “model.” A model can be provided new data and reason against it based on what the model has previously learned. Machine learning models determine a set of rules using vast amounts of computing power that a human brain would be incapable of processing. The more data a machine learning model is fed, the more complex the rules and the more accurate the predictions. Whereas a statistical model is likely to have an inherent logic that can be understood by most people, the rules created by machine learning are often beyond human comprehension because our brains are incapable of digesting and analyzing enormous data sets.
The process of learning begins with observations or data, such as examples, direct experience, or instruction, to look for patterns in data and make better decisions in the future based on the examples that we provide. The primary aim is to allow the computers learn automatically without human intervention or assistance and adjust actions accordingly.[1]

1.1.1 Some Machine Learning Methods

Machine learning algorithms are often categorized as supervised or unsupervised.

● **Supervised machine learning** algorithms can apply what has been learned in the past to new data using labeled examples to predict future events. Starting from the analysis of a known training dataset, the learning algorithm produces an inferred function to make predictions about the output values. The system can provide targets for any new input after sufficient training. The learning algorithm can also compare its output with the correct, intended output and find errors to modify the model accordingly.
● In contrast, **unsupervised machine learning** algorithms are used when the information used to train is neither classified nor labeled. Unsupervised learning studies how systems can infer a function to describe a hidden structure from unlabeled data. The system doesn’t figure out the right output, but it explores the data and can draw inferences from datasets to describe hidden structures from unlabeled data.
● **Semi-supervised machine** learning algorithms fall somewhere in between supervised and unsupervised learning, since they use both labeled and unlabeled data for training – typically a small amount of labeled data and a large amount of unlabeled data. The systems that use this method can considerably improve learning accuracy. Usually, semi-supervised learning is chosen when the acquired labeled data requires skilled and relevant resources to train it / learn from it. Otherwise, acquiring unlabeled data generally doesn’t require additional resources.

- **Reinforcement machine learning** algorithms is a learning method that interacts with its environment by producing actions and discovers errors or rewards. Trial and error search and delayed reward are the most relevant characteristics of reinforcement learning. This method allows machines and software agents to automatically determine the ideal behavior within a specific context to maximize its performance. Simple reward feedback is required for the agent to learn which action is best; this is known as the reinforcement signal.
- Machine learning enables analysis of massive quantities of data. While it generally delivers faster, more accurate results to identify profitable opportunities or dangerous risks, it may also require additional time and resources to train it properly. Combining machine learning with AI and cognitive technologies can make it even more effective in processing large volumes of information. [1]

1.2 Music Genre

A **music genre** is a conventional category that identifies some pieces of music as belonging to a shared tradition or set of conventions. It is to be distinguished from *musical form* and *musical style*, although in practice these terms are sometimes used interchangeably. Recently, academics have argued that categorizing music by genre is inaccurate and outdated. Musical genres are categorical labels created by humans to characterize pieces of music. A musical genre is characterized by the common characteristics shared by its members. Genre hierarchies are commonly used to structure the large collections of music available on the Web.

1.2.1 The Importance of Music Genre Classification

The topic of the importance of music genres comes up often in country music, especially over recent years (and recent days) as we've seen the encroachment of other genres into country music like never before. On paper, there's nothing wrong with combining two or more genres of music to create something unique. The problem is that often when this enterprise is undertaken, it's not creating something unique, it's meant to mimic something that is patently similar to everything else being released in popular music, making it even more uninteresting and non-creative to try and appeal to as many people as possible, regardless of what the artist may attempt to sell you. Folks will say combining influences is necessary for music to "evolve," but that evolution regularly sounds like devolution with rehashed melodies and structures. Even more dangerous is when artists or the media wrongly classify a song in a genre when it is clearly the product of another genre, even though many times these are the same people who will preach that genres don't matter. Breaking down genres is not enhancing the diversity of popular music, it is the *death* of diversity in popular music.

Fig-1 Different Music Genres

1.2.2 Applications of Music Genre Classification

1.2.2.1 Playlist Generation:

Instead of browsing a music collection by accessing each song individually, it is often interesting to generate playlists of songs, i.e. an ordered list of songs that can be listened to in a sequence. Using our application, the users can generate their playlists automatically according to which genre they like.

1.2.2.2 Music Recommendation System:

Rapid development of mobile devices and internet has made possible for us to access different music resources freely. The number of songs available exceeds the listening capacity of single individual. People sometimes feel difficult to choose from millions of songs. Moreover, music service providers need an efficient way to manage songs and help their customers to discover music by giving quality recommendation. Thus, there is a strong need of a good recommendation system. In this project, we have designed, implemented and analyzed a song recommendation system. We used Song Dataset provided by GTZAN to find correlations between users and songs and to learn from the previous listening history of users to provide recommendations for songs which users would prefer to listen most in future. Due to memory and processing power limitations, we could only experiment with a fraction of whole available dataset. We have implemented various algorithms such as popularity-based model, memory based collaborative filtering, SVD (Singular Value decomposition) based on latent factors and content-based model using k-NN.

1.3 Artificial Neural Network

Artificial neural networks (**ANNs**) or **connectionist systems** are computing systems inspired by the biological neural networks that constitute animal brains. Such systems learn (progressively improve their ability) to do tasks by considering examples, generally without task-specific programming. For example, in image recognition, they might learn to identify images that contain cats by analyzing example images that have been manually labeled as "cat" or "no cat" and using the analytic results to identify cats in other images. They have found most use in applications difficult to express with a traditional computer algorithm using rule-based programming.

An ANN is based on a collection of connected units called artificial neurons, (analogous to axons in a biological brain). Each connection (synapse) between neurons can transmit a signal to another neuron. The receiving (postsynaptic) neuron can process the signal(s) and then signal downstream neurons connected to it. Neurons may have state, generally represented by real numbers, typically between 0 and 1. Neurons and synapses may also have a weight that varies as learning proceeds, which can increase or decrease the strength of the signal that it sends downstream. Typically, neurons are organized in layers. Different layers may perform different kinds of transformations on their inputs. Signals travel from the first (input), to the last (output) layer, possibly after traversing the layers multiple times.

The original goal of the neural network approach was to solve problems in the same way that a human brain would. Over time, attention focused on matching specific mental abilities, leading to deviations from biology such as backpropagation, or passing information in the reverse direction and adjusting the network to reflect that information. Neural networks have been used on a variety of tasks, including computer vision, speech recognition, machine translation, social network filtering, playing board and video games and medical diagnosis.[2]

1.4 Deep Neural Network

A deep neural network (DNN) is an artificial neural network (ANN) with multiple hidden layers between the input and output layers. DNNs can model complex non-linear relationships. DNN architectures generate compositional models where the object is expressed as a layered composition of primitives. The extra layers enable composition of features from lower layers, potentially modeling complex data with fewer units than a similarly performing shallow network. Deep architectures include many variants of a few basic approaches. Each architecture has found success in specific domains. It is not always possible to compare the performance of multiple architectures, unless they have been evaluated on the same data sets.

DNNs are typically feedforward networks in which data flows from the input layer to the output layer without looping back. Recurrent neural networks (RNNs), in which data can flow in any direction, are used for applications such as modeling. Long short-term memory is particularly effective for this use. Convolutional deep neural networks (CNNs) are used in computer vision. CNNs also have been applied to acoustic modeling for automatic speech recognition (ASR).[3]

1.4.1 Challenges

As with ANNs, many issues can arise with naively trained DNNs. Two common issues are overfitting and computation time. DNNs are prone to overfitting because of the added layers of abstraction, which allow them to model rare dependencies in the training data. Alternatively, dropout regularization randomly omits units from the hidden layers during training. This helps to exclude rare dependencies. Finally, data can be augmented via methods such as cropping and rotating such that smaller training sets can be increased in size to reduce the chances of overfitting. DNNs must consider many training parameters, such as the size (number of layers and number of units per layer), the learning rate, and initial weights. Sweeping through the parameter space for optional parameters may not be feasible due to the cost in time and computational resources. Various tricks, such as batching (computing the gradient on several training examples at once rather than individual examples) speed up computation.

Large processing capabilities of many-core architectures (such as, GPUs or the Intel Xeon Phi) have produced significant speedups in training, because of the suitability of such processing architectures for the matrix and vector computations. Alternatively, engineers may look for other types of neural networks with more straightforward and convergent training algorithms. CMAC (cerebellar model articulation controller) is one such kind of neural network. It doesn't require learning rates or randomized initial weights for CMAC. The training process can be guaranteed to converge in one step with a new batch of data, and the computational complexity of the training algorithm is linear with respect to the number of neurons involved.[3]

1.5 Deep Learning

Deep learning (also known as deep structured learning or hierarchical learning) is part of a broader family of machine learning methods based on learning data representations, as opposed to task-specific algorithms. Learning can be supervised, semi-supervised or unsupervised.

Deep learning models are loosely related to information processing and communication patterns in a biological nervous system, such as neural coding that attempts to define a relationship between various stimuli and associated neuronal responses in the brain.

Deep learning architectures such as deep neural networks, deep belief networks and recurrent neural networks have been applied to fields including computer vision, speech recognition, natural language processing, audio recognition, social network filtering, machine translation, bioinformatics and drug design, where they have produced results comparable to and in some cases superior to human experts.[4]

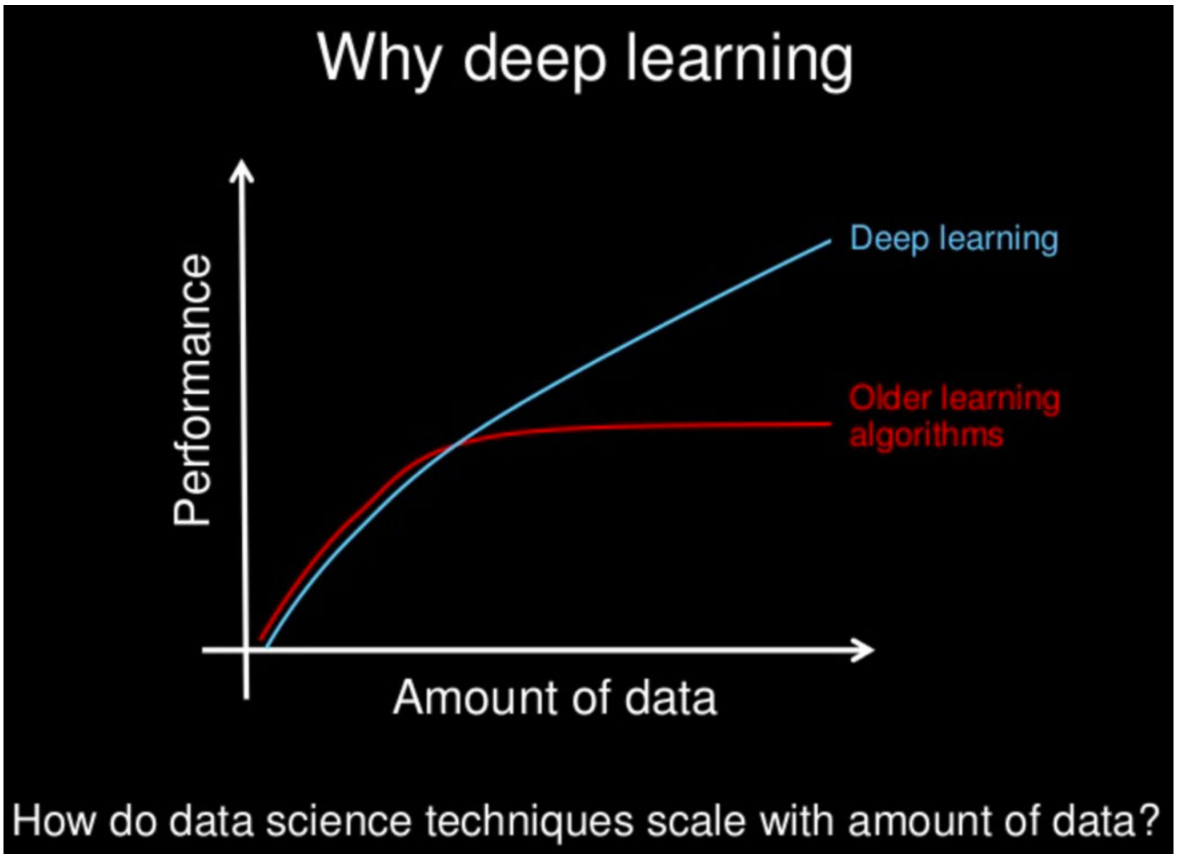


Fig-2 Why deep learning

1.5.1 Applications of deep learning

1.5.1.1 Automatic Speech Recognition

Large-scale automatic speech recognition is the first and most convincing successful case of deep learning. LSTM RNNs can learn "Very Deep Learning" tasks that involve multi-second intervals containing speech events separated by thousands of discrete time steps, where one-time step corresponds to about 10 ms. LSTM with forget gates is competitive with traditional speech recognizers on certain tasks.

The initial success in speech recognition was based on small-scale recognition tasks based on TIMIT. The data set contains 630 speakers from eight major dialects of American English, where each speaker reads 10 sentences. Its small size lets many configurations be tried. More importantly, the TIMIT task concerns phone-sequence recognition, which, unlike word-sequence recognition, allows weak language models (without a strong grammar).[clarification needed]This lets the weaknesses in acoustic modeling aspects of speech recognition be more easily analyzed.

1.5.1.2 Image Recognition

A common evaluation set for image classification is the MNIST database data set. MNIST is composed of handwritten digits and includes 60,000 training examples and 10,000 test examples. As with TIMIT, its small size lets users test multiple configurations. A comprehensive list of results on this set is available. Deep learning-based image recognition has become "superhuman", producing more accurate results than human contestants. This first occurred in 2011. Deep learning-trained vehicles now interpret 360° camera views. Another example is Facial Dysmorphology Novel Analysis (FDNA) used to analyze cases of human malformation connected to a large database of genetic syndromes.

1.5.1.3 Natural Language Processing

Neural networks have been used for implementing language models since the early 2000s.LSTM helped to improve machine translation and language modelling. Other key techniques in this field are negative sampling and word embedding. Word embedding, such as word2vec, can be thought of as a representational layer in a deep learning architecture that transforms an atomic word into a positional representation of the word relative to other words in the dataset; the position is represented as a point in a vector space. Using word embedding as an RNN input layer allows the network to parse sentences and phrases using an effective compositional vector grammar. A compositional vector grammar can be thought of as probabilistic context free grammar (PCFG) implemented by an RNN. Recursive auto-encoders built atop word embeddings can assess sentence similarity and detect paraphrasing. Deep neural architectures provide the best results for constituency parsing, sentiment analysis, information retrieval, spoken language understanding, machine translation, contextual entity linking, writing style recognition and others.
Google Translate (GT) uses a large end-to-end long short-term memory network. GNMT uses an example-based machine translation method in which the system "learns from millions of examples." It translates "whole sentences at a time, rather than pieces. Google Translate supports over one hundred languages. The network encodes the "semantics of the sentence rather than simply memorizing phrase-to-phrase translations". GT uses English as an intermediate between most language pairs.

1.5.1.4 Recommendation Systems

Recommendation systems have used deep learning to extract meaningful features for a latent factor model for content-based music recommendations. Multiview deep learning has been applied for learning user preferences from multiple domains. The model uses a hybrid collaborative and content-based approach and enhances recommendations in multiple tasks.

1.5.1.5 Bioinformatics

An auto encoder ANN was used in bioinformatics, to predict gene ontology annotations and gene-function relationships. In medical informatics, deep learning was used to predict sleep quality based on data from wearables and predictions of health complications from electronic health record data. Deep learning has also showed efficacy in healthcare.[5]

1.6 Objective

This project was primarily aimed to create an automated system for classification model for music genres. The first step included finding good features that demarcated genre boundaries clearly. A total of five features, namely MFCC vector, chroma frequencies, spectral roll-off, spectral centroid, zero-crossing rate were used for obtaining feature
vectors for the classifiers from the GTZAN genre dataset . Many different classifiers were trained and used to classify, each yielding varying degrees of accuracy in prediction. At the end the accuracy of machine learning algorithms is compared with the Deep Learning Algorithm.

2. Terminologies

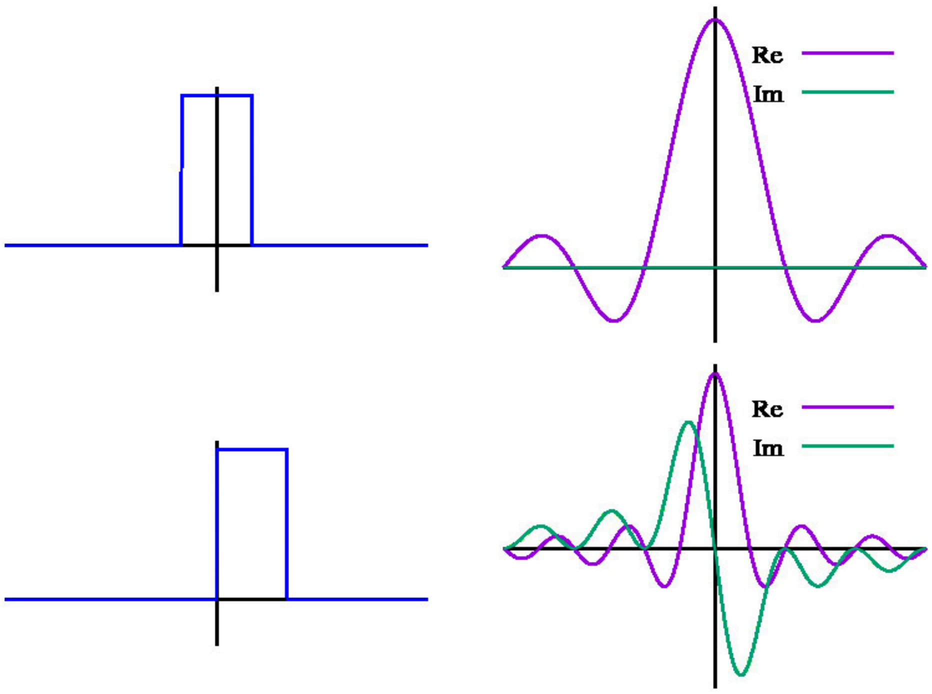
2.1 MFCC

In sound processing, the **mel-frequency cepstrum** (**MFC**) is a representation of the short-term power spectrum of a sound, based on a linear cosine transform of a log power spectrum on a nonlinear mel scale of frequency.

Mel-frequency cepstral coefficients (**MFCCs**) are coefficients that collectively make up an MFC. They are derived from a type of cepstral representation of the audio clip (a nonlinear "spectrum-of-a-spectrum"). The difference between the cepstrum and the mel-frequency cepstrum is that in the MFC, the frequency bands are equally spaced on the mel scale, which approximates the human auditory systems' response more closely than the linearly-spaced frequency bands used in the normal cepstrum. This frequency warping can allow for better representation of sound, for example, in audio compression.[23]

MFCCs are commonly derived as follows:

- Take the Fourier transform of (a windowed excerpt of) a signal. The Fourier transform of integrable function $f : \mathbb{R} \mapsto \mathbb{C}$ is $\hat{f}(\xi) = \int_{-\infty}^{\infty} f(x)\, e^{-2\pi i x \xi}\, dx,$

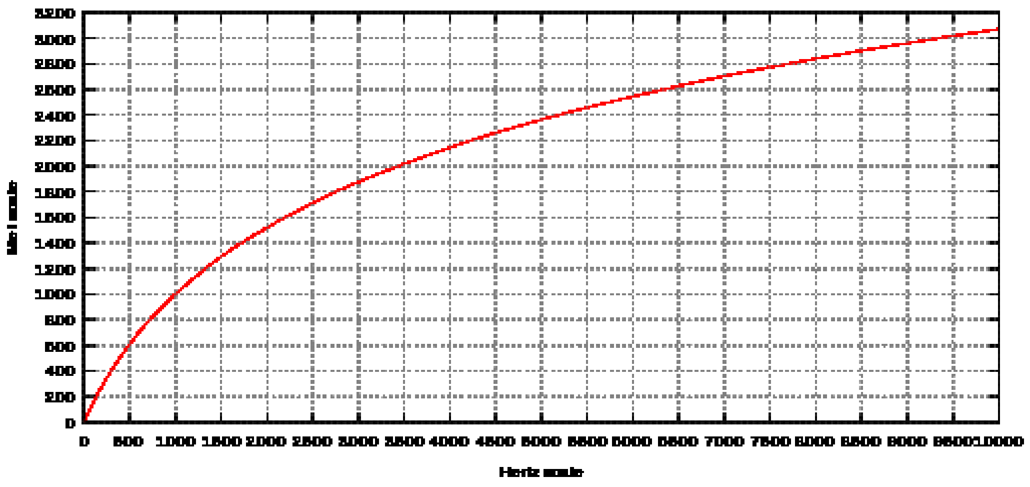


In the first row is the graph of the unit pulse function f (t) and its Fourier transform $\hat{f}$ (ω), a function of frequency ω. Translation (that is, delay) in the time domain is interpreted as complex phase shifts in the frequency domain. In the second row is shown g(t), a delayed unit pulse, beside the real and imaginary parts of the Fourier transform. The Fourier

Fig-3 Fourier Transform

- Map the powers of the spectrum obtained above onto the mel scale, using triangular overlapping windows. The **mel scale**, named by Stevens, Volkmann, and Newman in 1937, is a perceptual scale of pitches judged by listeners to be equal in distance from one another. The reference point between this scale and normal frequency measurement is defined by assigning a perceptual pitch of 1000 mels to a 1000 Hz tone, 40 dB above the listener's threshold. A popular formula to convert *f* hertz into *m* mels is

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



Plots of pitch mel scale versus Hertz scale

Fig-4 Mel vs Hertz

- Take the logs of the powers at each of the mel frequencies.
- Take the discrete cosine transform of the list of mel log powers, as if it were a signal. A **discrete cosine transform** (**DCT**) expresses a finite sequence of data points in terms of a sum of cosine functions oscillating at different frequencies.
- The MFCCs are the amplitudes of the resulting spectrum.

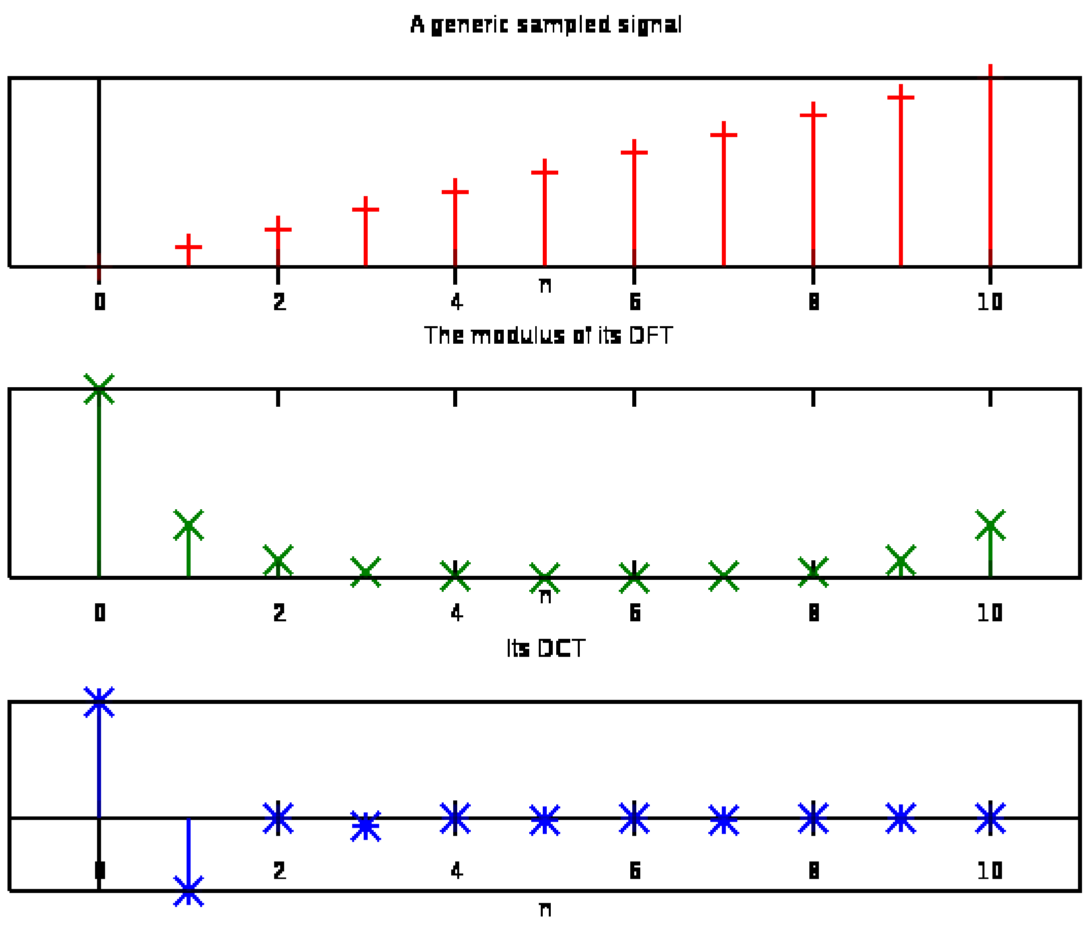


Fig- 5 DCT(Discrete Cosine Transform)

2.1.1 Chroma Frequencies

Chroma frequency vector discretized the spectrum into chromatic keys and represents the presence of each key. We take the histogram of present notes on a 12-note scale as a 12-length feature vector. The chroma frequency have a music theory interpretation. The histogram over the 12-note scale is sufficient to describe the chord played in that window. It provides a robust way to describe a similarity measure between music pieces.

2.1.2 Spectral Centroid

It describes where the ”center of mass” for sound is. It essentially is the weighted mean of the frequencies present in the sound. Consider two songs, one from blues and one from metal. A blues song is generally consistent throughout it length while a metal song usually has more frequencies accumulated towards the end part. So spectral centroid for blues song will lie somewhere near the middle of its spectrum while that for a metal song would usually be towards its end. Fig. 3. Formula to calculate the Spectral Centroid x(n) is the weighted frequency value of bin number n f(n) is the center frequency of that bin Zero Crossing Rate

It represents the number of times the waveform crosses 0. It usually has higher values for highly percussive sounds like those in metal and rock. Fig. 4. Formula to calculate the Zero Crossing Rate st is the signal of length t II{X} is the indicator function (=1 if X true, else =0) Spectral Roll-off It is a measure of the shape of the signal. It represents the frequency at which high frequencies decline to 0. To obtain it, we have to calculate the fraction of bins in the power spectrum where 85% of its power is at lower frequencies.

$$Centroid = \frac{\sum_{n=0}^{N-1} f(n)x(n)}{\sum_{n=0}^{N-1} x(n)}$$

Fig. 3. Formula to calculate the Spectral Centroid
x(n) is the weighted frequency value of bin number n
f(n) is the center frequency of that bin

2.1.3 Zero Crossing Rate

It represents the number of times the waveform crosses 0. It usually has higher values for highly percussive sounds like those in metal and rock.

$$zcr = \frac{1}{T-1}\sum_{t=1}^{T-1} \mathbb{I}\{s_t s_{t-1} < 0\}$$

Fig. 4. Formula to calculate the Zero Crossing Rate
st is the signal of length t
II{X} is the indicator function (=1 if X true, else =0)

2.1.4 Spectral Roll-off

It is a measure of the shape of the signal. It represents the frequency at which high frequencies decline to 0. To obtain it, we have to calculate the fraction of bins in the power spectrum where 85% of its power is at lower frequencies.[6]

2.2 CNN

A convolutional neural network (CNN, or ConvNet) is a class of deep, feed-forward artificial neural networks that has successfully been applied to analyzing visual imagery.
CNNs use a variation of multilayer perceptrons designed to require minimal preprocessing. They are also known as shift invariant or space invariant artificial neural networks (SIANN), based on their shared-weights architecture and translation invariance characteristics.
CNNs use relatively little pre-processing compared to other image classification algorithms. This means that the network learns the filters that in traditional algorithms were hand-engineered. This independence from prior knowledge and human effort in feature design is a major advantage.
A CNN consists of an input and an output layer, as well as multiple hidden layers. The hidden layers of a CNN typically consist of convolutional layers, pooling layers, fully connected layers and normalization layers.
Description of the process as a convolution in neural networks is by convention. Mathematically it is a cross-correlation rather than a convolution. This only has significance for the indices in the matrix, and thus which weights are placed at which index.[7][20]

2.2.1 Convolutional

Convolutional layers apply a convolution operation to the input, passing the result to the next layer. The convolution emulates the response of an individual neuron to visual stimuli. Each convolutional neuron processes data only for its receptive field.

2.2.2 Pooling

Convolutional networks may include local or global pooling layers, which combine the outputs of neuron clusters at one layer into a single neuron in the next layer. For example, max pooling uses the maximum value from each of a cluster of neurons at the prior layer. Another example is average pooling, which uses the average value from each of a cluster of neurons at the prior layer.

2.2.3 Weights

CNNs share weights in convolutional layers, which means that the same filter (weights bank) is used for each receptive field in the layer; this reduces memory footprint and improves performance.

In summary:

- A ConvNet architecture is in the simplest case a list of Layers that transform the image volume into an output volume (e.g. holding the class scores)
- There are a few distinct types of Layers (e.g. CONV/FC/RELU/POOL are by far the most popular)

- Each Layer accepts an input 3D volume and transforms it to an output 3D volume through a differentiable function
- Each Layer may or may not have parameters (e.g. CONV/FC do, RELU/POOL don't)
- Each Layer may or may not have additional hyperparameters (e.g. CONV/FC/POOL do, RELU doesn't)

2.2.4 Building blocks

2.2.4.1 Convolutional layer

The convolutional layer is the core building block of a CNN. The layer's parameters consist of a set of learnable filters (or kernels), which have a small receptive field, but extend through the full depth of the input volume. During the forward pass, each filter is convolved across the width and height of the input volume, computing the dot product between the entries of the filter and the input and producing a 2-dimensional activation map of that filter. As a result, the network learns filters that activate when it detects some specific type of feature at some spatial position in the input.
Stacking the activation maps for all filters along the depth dimension forms the full output volume of the convolution layer. Every entry in the output volume can thus also be interpreted as an output of a neuron that looks at a small region in the input and shares parameters with neurons in the same activation map.

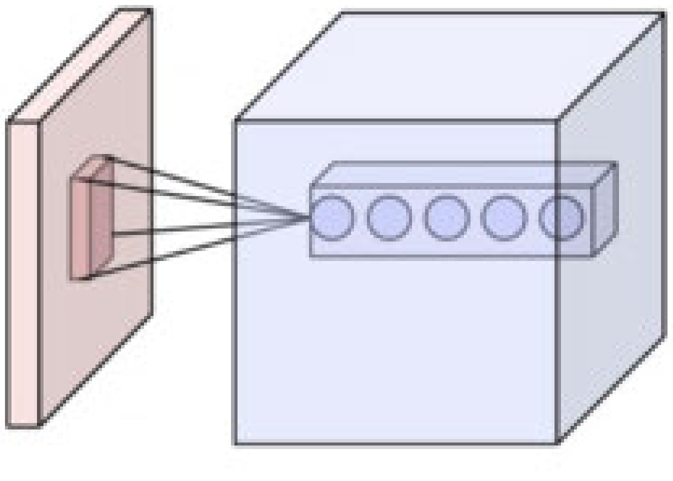


Fig-6 Neurons of a convolutional layer (blue), connected to their receptive field (red)

2.2.4.2 Local connectivity

When dealing with high-dimensional inputs such as images, it is impractical to connect neurons to all neurons in the previous volume because such a network architecture does not take the spatial structure of the data into account. Convolutional networks exploit spatially local correlation by enforcing a local connectivity pattern between neurons of adjacent layers: each neuron is connected to only a small region of the input volume. The extent of this connectivity is a hyperparameter called the receptive field of the neuron. The connections are local in space (along width and height), but always extend along the entire depth of the input volume. Such an architecture ensures that the learnt filters produce the strongest response to a spatially local input pattern.

2.2.4.3 Spatial arrangement

Three hyperparameters control the size of the output volume of the convolutional layer: the depth, stride and zero-padding.

● The depth of the output volume controls the number of neurons in a layer that connect to the same region of the input volume. These neurons learn to activate for different features in the input. For example, if the first convolutional layer takes the raw image as input, then different neurons along the depth dimension may activate in the presence of various oriented edges, or blobs of color.

● Stride controls how depth columns around the spatial dimensions (width and height) are allocated. When the stride is 1 then we move the filters one pixel at a time. This leads to heavily overlapping receptive fields between the columns, and also to large output volumes. When the stride is 2 (or rarely 3 or more) then the filters jump 2 pixels at a time as they slide around. The receptive fields overlap less, and the resulting output volume has smaller spatial dimensions.

● Sometimes it is convenient to pad the input with zeros on the border of the input volume. The size of this padding is a third hyperparameter. Padding provides control of the output volume spatial size. In particular, sometimes it is desirable to exactly preserve the spatial size of the input volume.

The spatial size of the output volume can be computed as a function of the input volume size W {\displaystyle W} W, the kernel field size of the Conv Layer neurons K {\displaystyle K} K, the stride with which they are applied S {\displaystyle S} S, and the amount of zero padding P {\displaystyle P} P used on the border. The formula for calculating how many neurons "fit" in a given volume is given by (W − K + 2 P) / S + 1 {\displaystyle (W-K+2P)/S+1} {\displaystyle (W-K+2P)/S+1}. If this number is not an integer, then the strides are set incorrectly, and the neurons cannot be tiled to fit across the input volume in a symmetric way. In general, setting zero padding to be P = (K − 1) / 2 {\displaystyle P=(K-1)/2} {\displaystyle P=(K-1)/2} when the stride is S = 1 {\displaystyle S=1} S=1 ensures that the input volume and output volume will have the same size spatially. Though it's generally not completely necessary to use up all of the neurons of the previous layer, for example, you may decide to use just a portion of padding.

2.2.4.4 Parameter sharing

Another important concept of CNNs is pooling, which is a form of non-linear down-sampling. There are several nonlinear functions to implement pooling among which *max pooling* is the most common. It partitions the input image into a set of non-overlapping rectangles and, for each such sub-region, outputs the maximum. The intuition is that the exact location of a feature is less important than its rough location relative to other features. The pooling layer serves to progressively reduce the spatial size of the representation, to reduce the number of parameters and amount of computation in the network, and hence to also control overfitting. It is common to periodically insert a pooling layer between successive convolutional layers in a CNN architecture. The pooling operation provides another form of translation invariance.

The pooling layer operates independently on every depth slice of the input and resizes it spatially. The most common form is a pooling layer with filters of size 2x2 applied with a stride of 2 down samples at every depth slice in the input by 2 along both width and height, discarding 75% of the activations. In this case, every max operation is over 4 numbers. The depth dimension remains unchanged.
In addition to max pooling, the pooling units can use other functions, such as average pooling or L2-norm pooling. Average pooling was often used historically but has recently fallen out of favor compared to max pooling, which works better in practice.
Due to the aggressive reduction in the size of the representation, the trend is towards using smaller filters or discarding the pooling layer altogether.

RoI pooling to size 2x2. In this example region proposal (an input parameter) has size 7x5.
Region of Interest pooling (also known as RoI pooling) is a variant of max pooling, in which output size is fixed and input rectangle is a parameter.
Pooling is an important component of convolutional neural networks for object detection based on Fast R-CNN architecture.

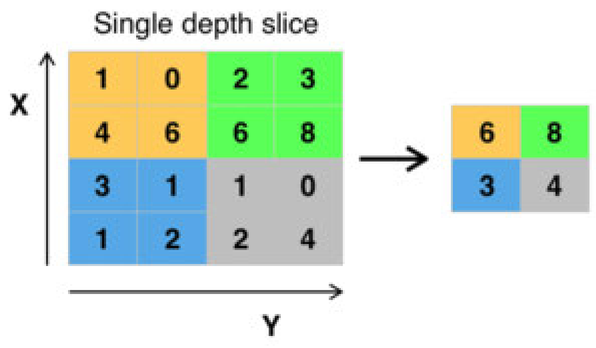


Fig- 7 Max pooling with a 2x2 filter and stride = 2

2.2.4.5 ReLU layer

ReLU is the abbreviation of Rectified Linear Units. This layer applies the non-saturating activation function f (x) = max (0 , x) {\displaystyle f(x)=\max(0,x)} {\displaystyle f(x)=\max(0,x)}. It increases the nonlinear properties of the decision function and of the overall network without affecting the receptive fields of the convolution layer.

Other functions are also used to increase nonlinearity, for example the saturating hyperbolic tangent f (x) = tanh ⁡ (x) {\displaystyle f(x)=\tanh(x)} {\displaystyle f(x)=\tanh(x)}, f (x) = | tanh ⁡ (x) | {\displaystyle f(x)=|\tanh(x)|} {\displaystyle f(x)=|\tanh(x)|}, and the sigmoid function f (x) = (1 + e − x) − 1 {\displaystyle f(x)=(1+e^{-x})^{-1}} f(x)=(1+e^{-x})^{-1}. ReLU is often preferred to other functions, because it trains the neural network several times faster[39] without a significant penalty to generalization accuracy.

2.2.4.6 Fully connected layer

Finally, after several convolutional and max pooling layers, the high-level reasoning in the neural network is done via fully connected layers. Neurons in a fully connected layer have connections to all activations in the previous layer, as seen in regular neural networks. Their activations can hence be computed with a matrix multiplication followed by a bias offset.

2.2.4.7 Loss layer

The loss layer specifies how training penalizes the deviation between the predicted and true labels and is normally the final layer. Various loss functions appropriate for different tasks may be used there. SoftMax loss is used for predicting a single class of K mutually exclusive classes. Sigmoid cross-entropy loss is used for predicting K independent probability values in [0 , 1] {\displaystyle [0,1]} [0,1]. Euclidean loss is used for regressing to real-valued labels (− ∞ , ∞) {\displaystyle (-\infty ,\infty)} (-\infty ,\infty).[7]

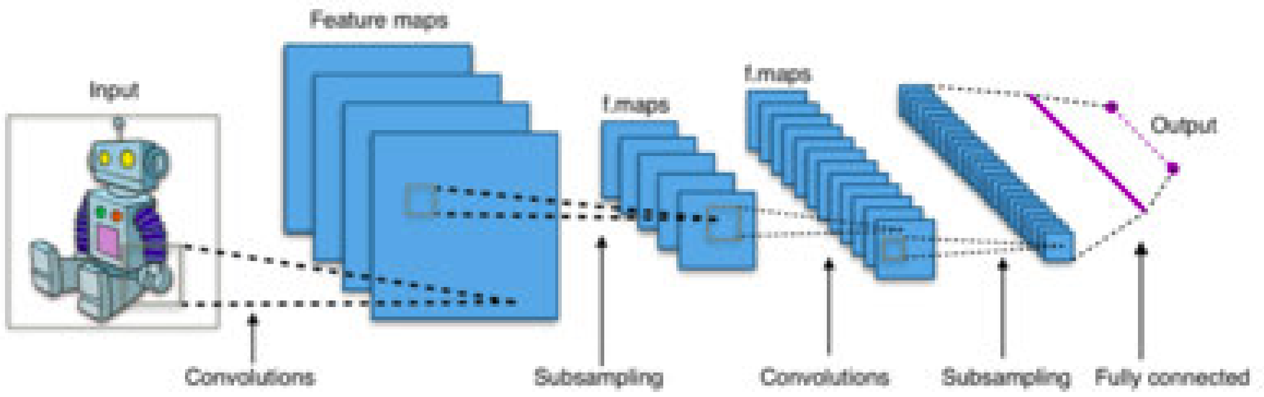


Fig-8 Typical CNN architecture

2.3 CRNN

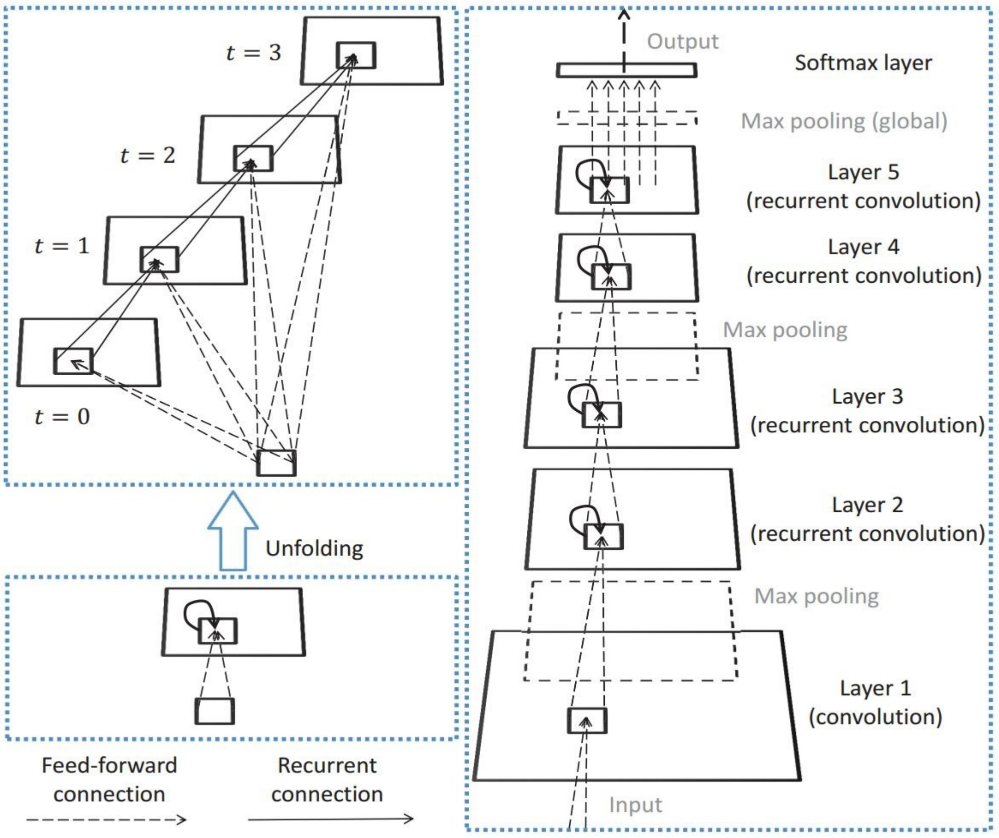
A CRNN (Convolution recurrent neural network) is a combination of Convolution Neural Network(CNN) and Recurrent Neural Network(RNN)[22].

A **recurrent neural network** (**RNN**) is a class of artificial neural network where connections between units form a directed graph along a sequence. This allows it to exhibit dynamic temporal behavior for a time sequence. Unlike feedforward neural networks, RNNs can use their internal state (memory) to process sequences of inputs.

In a mixed CNN and RNN architecture the positive features of an RNN are used to improve the CNN. As described by Liang and Hu in their paper architecture for object detection, which is the first use of term CRNN, they define the main idea as:

A prominent difference is that CNN is typically a feed-forward architecture while in the visual system recurrent connections are abundant. Inspired by this fact, we propose a recurrent CNN (RCNN) for object recognition by incorporating recurrent connections into each convolutional layer.

The key module of this RCNN are the recurrent convolutional layers (RCL), which introduce recurrent connection into a convolution layer. With these connections the network can evolve over time though the input is static, and each unit is influenced by its neighboring units. The network is trained with BPTT and therefore with the same unfolding algorithm described in the first paragraph (Introduction). The unfolding facilitates the learning process and sharing weights reduce the parameters. In figure 9 on the left the unfolding for three-time steps of the recurrent connection is shown and on the right the architecture of the whole used RCNN. The network consists of first a convolutional layer to save computations, which is followed by a max pooling layer. On top of that two RCL, one max pooling and then again two RCL layer are used. Finally, one global max pooling and a SoftMax layer are used. The pooling operation have stride two and size three. The global max pooling layer outputs the maximum over every feature map, yielding to a feature vector that represents the image.[8]



Mixed CNN and RNN architecture. On the left the a RCL is unfolded for three time steps, leading to a feed-forward network with largest depth of four and smallest depth of one. On the right the RCNN used is shown with one convolutional layer, four RCL, three max pooling and one SoftMax layer.

Fig- 9 Mixed CNN and RNN Architecture

3. Architectures

3.1 Fully recurrent

Basic RNNs are a network of neuron-like nodes, each with a directed (one-way) connection to every other node.[citation needed] Each node (neuron) has a time-varying real-valued activation. Each connection (synapse) has a modifiable real-valued weight. Nodes are either input nodes (receiving data from outside the network), output nodes (yielding results), or hidden nodes (that modify the data on route from input to output).

For supervised learning in discrete time settings, sequences of real-valued input vectors arrive at the input nodes, one vector at a time. At any given time, step, each non-input unit computes its current activation (result) as a nonlinear function of the weighted sum of the activations of all units that connect to it. Supervisor-given target activations can be supplied for some output units at certain time steps. For example, if the input sequence is a speech signal corresponding to a spoken digit, the final target output at the end of the sequence may be a label classifying the digit.

In reinforcement learning settings, no teacher provides target signals. Instead a fitness function or reward function is occasionally used to evaluate the RNN's performance, which influences its input stream through output units connected to actuators that affect the environment. This might be used to play a game in which progress is measured with the number of points won.

Each sequence produces an error as the sum of the deviations of all target signals from the corresponding activations computed by the network. For a training set of numerous sequences, the total error is the sum of the errors of all individual sequences.

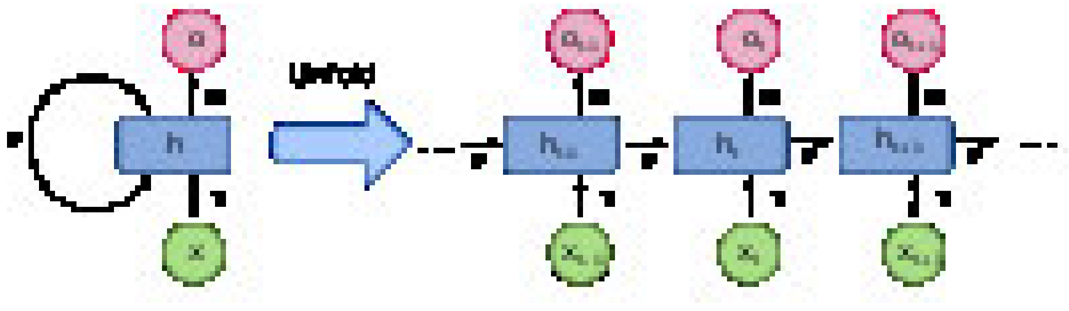


Fig-10 Unfolded basic recurrent neural network

3.2 Recursive

A recursive neural network is created by applying the same set of weights recursively over a differentiable graph-like structure by traversing the structure in topological order. Such networks are typically also trained by the reverse mode of automatic differentiation. They can process distributed representations of structure, such as logical terms. A special case of recursive neural networks is the RNN whose structure corresponds to a linear chain. Recursive neural networks have been applied to natural language processing. The Recursive Neural Tensor Network uses a tensor-based composition function for all nodes in the tree.

3.3 Bidirectional associative memory

Introduced by Kosko, a bidirectional associative memory (BAM) network is a variant of a Hopfield network that stores associative data as a vector. The bi-directionality comes from passing information through a matrix and its transpose. Typically, bipolar encoding is preferred to binary encoding of the associative pairs. Recently, stochastic BAM models using Markov stepping were optimized for increased network stability and relevance to real-world applications.
A BAM network has two layers, either of which can be driven as an input to recall an association and produce an output on the other layer.[8]

3.4 kNN algorithm

In pattern recognition, the ***k*-nearest neighbors algorithm** (***k*-NN**) is a non-parametric method used for classification and regression. In both cases, the input consists of the *k* closest training examples in the feature space. *k*-NN is a type of instance-based learning, or lazy learning, where the function is only approximated locally, and all computation is deferred until classification. [26]

Suppose we have pairs $(X,Y),(X_1,Y_1),\ldots,(X_n,Y_n)$ taking values in $\mathbb{R}^d \times \{1,2\}$ where *Y* is the class label of *X*, so that $X|Y=r \sim P_r$ for $r=1,2$ (and probability distributions P_r). Given some norm $\|\cdot\|$ on $\mathbb{R}^d$ and a point $x \in \mathbb{R}^d$, let $(X_{(1)},Y_{(1)}),\ldots,(X_{(n)},Y_{(n)})$ be a reordering of the training data such that $\|X_{(1)}-x\| \leq \cdots \leq \|X_{(n)}-x\|$.

The training examples are vectors in a multidimensional feature space, each with a class label. The training phase of the algorithm consists only of storing the feature vectors and class labels of the training samples.

In the classification phase, *k* is a user-defined constant, and an unlabeled vector (a query or test point) is classified by assigning the label which is most frequent among the *k* training samples nearest to that query point.

A commonly used distance metric for continuous variables is Euclidean distance. For discrete variables, such as for text classification, another metric can be used, such as the **overlap metric** (or Hamming distance). In the context of gene expression microarray data, for example, *k*-NN has also been employed with correlation coefficients such as Pearson and Spearman.[1] Often, the classification accuracy of *k*-NN can be improved significantly if the distance metric is learned with specialized algorithms such as Large Margin Nearest Neighbor or Neighborhood components analysis.[9]

3.4.1 kNN Algorithm Pseudocode:

```
1.  Calculate "d(x, x_i)" i =1, 2, ….., n; where d denotes the Euclidean Distance between the points.
2.  Arrange the calculated n Euclidean distances in non-decreasing order.
3.  Let k be a +ve integer, take the first k distances from this sorted list.
4.  Find those k-points corresponding to these k-distances.
5.  Let k_i denotes the number of points belonging to the i^th class among k points i.e. k ≥ 0
6.  If $k_i > k_j \ \forall \ i \neq j$ then put x in class i.
```

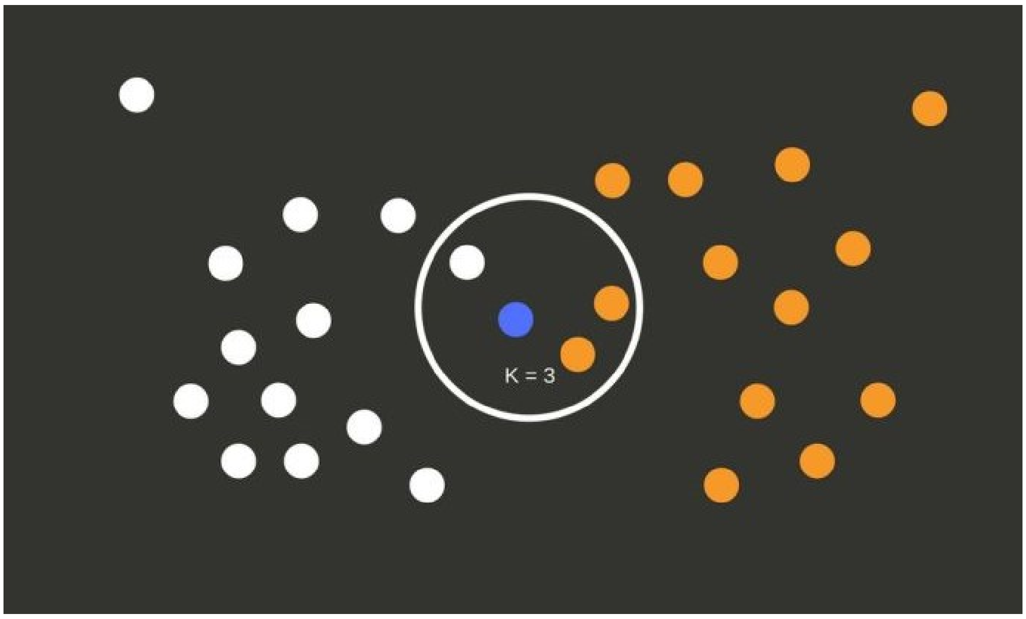
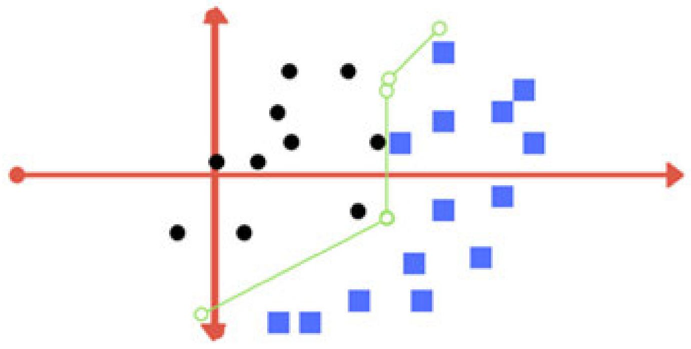


Fig- 11 KNN Algorithm

3.5 SVM

A Support Vector Machine (SVM) is a discriminative classifier formally defined by a separating hyperplane. In other words, given labeled training data (supervised learning), the algorithm outputs an optimal hyperplane which categorizes new examples. In two-dimensional space this hyperplane is a line dividing a plane in two parts where in each class lay in either side.[25]



Classification using SVM

Fig- 12 Classification using SVM

3.5.1 Kernel

The learning of the hyperplane in linear SVM is done by transforming the problem using some linear algebra. This is where the kernel plays role. For **linear kernel** the equation for prediction for a new input using the dot product between the input (x) and each support vector (xi) is calculated as follows:

f(x) = B(0) + sum(ai * (x,xi))

This is an equation that involves calculating the inner products of a new input vector (x) with all support vectors in training data. The coefficients B0 and ai (for each input) must be estimated from the training data by the learning algorithm.

The **polynomial kernel** can be written as *K(x,xi) = 1 + sum(x * xi)^d* and **exponential** as *K(x,xi) = exp(-gamma * sum((x—xi^2)).*

3.5.2 Regularization

The Regularization parameter (often termed as C parameter in python’s sklearn library) tells the SVM optimization how much you want to avoid misclassifying each training example.

For large values of C, the optimization will choose a smaller-margin hyperplane if that hyperplane does a better job of getting all the training points classified correctly. Conversely, a very small value of C will cause the optimizer to look for a larger-margin separating hyperplane, even if that hyperplane misclassified more points.

3.5.3 Gamma

The gamma parameter defines how far the influence of a single training example reaches, with low values meaning 'far' and high values meaning 'close'. In other words, with low gamma, points far away from plausible separation line are considered in calculation for the separation line. Whereas high gamma means the points close to plausible line are considered in calculation.

3.5.4 Margin

And finally, last but very important characteristic of SVM classifier. SVM to core tries to achieve a good margin. A margin is a separation of line to the closest class points. A ***good margin*** is one where this separation is larger for both the classes. Images below gives to visual example of good and bad margin. A good margin allows the points to be in their respective classes without crossing to other class.

3.5.5 Applications

SVMs can be used to solve various real-world problems:

- SVMs are helpful in text and hypertext categorization as their application can significantly reduce the need for labeled training instances in both the standard inductive and transudative settings.

- Classification of images can also be performed using SVMs. Experimental results show that SVMs achieve significantly higher search accuracy than traditional query refinement schemes after just three to four rounds of relevance feedback. This is also true of image segmentation systems, including those using a modified version SVM that uses the privileged approach as suggested by Vapnik.

- Handwritten characters can be recognized using SVM.

- The SVM algorithm has been widely applied in the biological and other sciences. They have been used to classify proteins with up to 90% of the compounds classified correctly. Permutation tests based on SVM weights have been suggested as a mechanism for interpretation of SVM models. Support vector machine weights have also been used to interpret SVM models in the past. Posthoc interpretation of support vector machine models in order to identify features used by the model to make predictions is a relatively new area of research with special significance in the biological sciences.[10]

3.6 k Means

k-means is one of the simplest unsupervised learning algorithms that solve the well-known clustering problem. The procedure follows a simple and easy way to classify a given data set through a certain number of clusters (assume k clusters) fixed apriori. The main idea is to define k centers, one for each cluster. These centers should be placed in a cunning way because of different location causes different result. So, the better choice is to place them as much as possible far away from each other. The next step is to take each point belonging to a given data set and associate it to the nearest center. When no point is pending, the first step is completed, and an early group age is done. At this point we need to re-calculate k new centroids as barycenter of the clusters resulting from the previous step. After we have these k new centroids, a new binding has to be done between the same data set points and the nearest new center. A loop has been generated. As a result of this loop we may notice that the k centers change their location step by step until no more changes are done or in other words centers do not move any more. Finally, this algorithm aims at minimizing an objective function know as squared error function given by:

$$J(V) = \sum_{i=1}^{c} \sum_{j=1}^{c_i} \left(\|x_i - v_j\| \right)^2$$

where,

'||xi - vj||' is the Euclidean distance between xi and vj.
'ci' is the number of data points in ith cluster.
'c' is the number of cluster centers.

3.6.1 Algorithmic steps for k-means clustering

Let X = {x1,x2,x3,……..,xn} be the set of data points and V = {v1,v2,…….,vc} be the set of centers.

1) Randomly select 'c' cluster centers.
2) Calculate the distance between each data point and cluster centers.
3) Assign the data point to the cluster center whose distance from the cluster center is minimum of all the cluster centers..
4) Recalculate the new cluster center using:
where, 'ci' represents the number of data points in ith cluster.
5) Recalculate the distance between each data point and new obtained cluster centers.
6) If no data point was reassigned then stop, otherwise repeat from step 3).[11]

3.6.2 Advantages

1) Fast, robust and easier to understand.
2) Relatively efficient: O(tknd), where n is # objects, k is # clusters, d is # dimension of each object, and t is # iterations. Normally, k, t, d << n.
3) Gives best result when data set are distinct or well separated from each other.

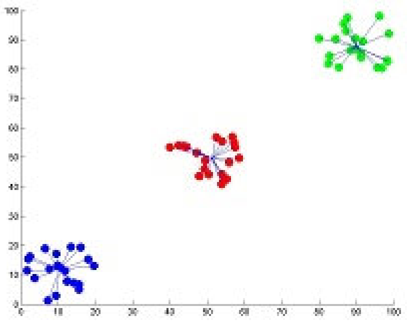


Fig 13: Showing the result of k-means for 'N' = 60 and 'c' = 3

3.6.3 Disadvantages

1) The learning algorithm requires apriori specification of the number of cluster centers.
2) The use of Exclusive Assignment - If there are two highly overlapping data then k-means will not be able to resolve that there are two clusters.
3) The learning algorithm is not invariant to non-linear transformations i.e. with different representation of data we get different results (data represented in form of cartesian co-ordinates and polar co-ordinates will give different results).
4) Euclidean distance measures can unequally weight underlying factors.
5) The learning algorithm provides the local optima of the squared error function.
6) Randomly choosing of the cluster center cannot lead us to the fruitful result.
7) Applicable only when mean is defined i.e. fails for categorical data.
8) Unable to handle noisy data and outliers.
9) Algorithm fails for non-linear data set.

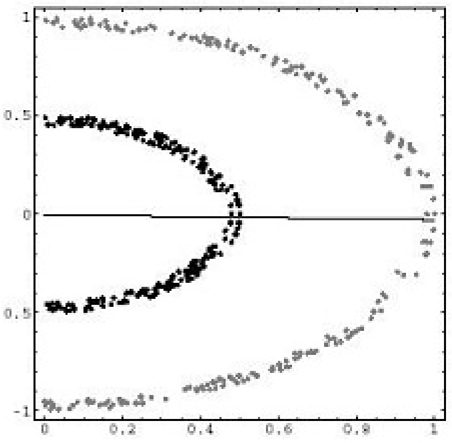


Fig 14: Showing the non-linear data set where k-means algorithm fails

4. Technologies

4.1 Python 2.7/3.5

Python is a widely used high-level programming language for general-purpose programming, created by Guido van Rossum and first released in 1991. An interpreted language, Python has a design philosophy that emphasizes code readability (notably using whitespace indentation to delimit code blocks rather than curly brackets or keywords), and a syntax that allows programmers to express concepts in fewer lines of code than might be used in languages such as C++ or Java.

The language provides constructs intended to enable writing clear programs on both a small and large scale. Python features a dynamic type system and automatic memory management and supports multiple programming paradigms, including object-oriented, imperative, functional programming, and procedural styles. It has a large and comprehensive standard library. Python interpreters are available for many operating systems, allowing Python code to run on a wide variety of systems.

4.2 Flask Framework

Flask is a micro web framework written in Python and based on the Werkzeug toolkit and Jinja2 template engine. It is BSD licensed. The latest stable version of Flask is 1.0 as of April 2018. Applications that use the Flask framework include Pinterest, LinkedIn, and the community web page for Flask itself. Flask is called a micro framework because it does not require particular tools or libraries.[6] It has no database abstraction layer, form validation, or any other components where pre-existing third-party libraries provide common functions. However, Flask supports extensions that can add application features as if they were implemented in Flask itself. Extensions exist for object-relational mappers, form validation, upload handling, various open authentication technologies and several common framework related tools. Extensions are updated far more regularly than the core Flask program.[12]

4.3 Front End Technologies:

4.3.1 HTML

Hypertext Markup Language (HTML) is the standard markup language for creating web pages and web applications. With Cascading Style Sheets (CSS) and JavaScript, it forms a triad of cornerstone technologies for the World Wide Web.

Web browsers receive HTML documents from a web server or from local storage and render the documents into multimedia web pages. HTML describes the structure of a web page semantically and originally included cues for the appearance of the document.

HTML elements are the building blocks of HTML pages. With HTML constructs, images and other objects such as interactive forms may be embedded into the rendered page. HTML provides a means to create structured documents by denoting structural semantics for text such as headings, paragraphs, lists, links, quotes and other items. HTML elements are delineated by tags, written using angle brackets. Tags such as <img /> and <input /> directly introduce content into the page. Other tags such as <p> surround and provide information about document text and may include other tags as sub-elements. Browsers do not display the HTML tags but use them to interpret the content of the page.[13]

4.3.2 CSS

Cascading Style Sheets (CSS) is a style sheet language used for describing the presentation of a document written in a markup language like HTML.[1] CSS is a cornerstone technology of the World Wide Web, alongside HTML and JavaScript.

CSS is designed to enable the separation of presentation and content, including layout, colors, and fonts. This separation can improve content accessibility, provide more flexibility and control in the specification of presentation characteristics, enable multiple web pages to share formatting by specifying the relevant CSS in a separate .css file, and reduce complexity and repetition in the structural content.

Separation of formatting and content also makes it feasible to present the same markup page in different styles for different rendering methods, such as on-screen, in print, by voice (via speech-based browser or screen reader), and on Braille-based tactile devices. CSS also has rules for alternate formatting if the content is accessed on a mobile device.

The name cascading comes from the specified priority scheme to determine which style rule applies if more than one rule matches a particular element. This cascading priority scheme is predictable.

The CSS specifications are maintained by the World Wide Web Consortium (W3C). Internet media type (MIME type) text/css is registered for use with CSS by RFC 2318 (March 1998). The W3C operates a free CSS validation service for CSS documents. In addition to HTML, other markup languages support the use of CSS, including XHTML, plain XML, SVG, and XUL.[14]

4.4 Feature Extraction:

4.4.1 Librosa

A python package for music and audio analysis. It provides the building blocks necessary to create music information retrieval systems.[15]

4.4.1.1 Installation

The latest stable release is available on PyPI, and you can install it by saying:

```
pip install librosa
```

4.4.1.2 Mel spectrogram

```
# Let's make and display a mel-scaled power (energy-squared) spectrogram
S = librosa.feature.melspectrogram(y, sr=sr, n_mels=128)

# Convert to log scale (dB). We'll use the peak power (max) as reference.
log_S = librosa.power_to_db(S, ref=np.max)

# Make a new figure
plt.figure(figsize=(12,4))

# Display the spectrogram on a mel scale
# sample rate and hop length parameters are used to render the time axis
librosa.display.specshow(log_S, sr=sr, x_axis='time', y_axis='mel')

# Put a descriptive title on the plot
plt.title('mel power spectrogram')

# draw a color bar
plt.colorbar(format='%+02.0f dB')

# Make the figure layout compact
plt.tight_layout()
```

4.5 Dataset

4.5.1 GTZAN Dataset

This dataset was used for the well-known paper in genre classification " Musical genre classification of audio signals " by G. Tzanetakis and P. Cook in IEEE Transactions on Audio and Speech Processing 2002.
Unfortunately, the database was collected gradually and very early on in my research, so I have no titles (and obviously no copyright permission etc.). The files were collected in 2000-2001 from a variety of sources including personal CDs, radio, microphone recordings, in order to represent a variety of recording conditions. Nevertheless, I have been providing it to researchers upon request mainly for comparison purposes etc.
The dataset consists of 1000 audio tracks each 30 second long. It contains 10 genres, each represented by 100 tracks. The tracks are all 22050Hz Mono 16-bit audio files in .wav format.[16]

4.6 scikit-learn

In addition to HTML, other markup languages support the use of CSS, including XHTML, plain XML, SVG, and XUL.

scikit-learn is a Python module for machine learning built on top of SciPy and distributed under the 3-Clause BSD license.

The project was started in 2007 by David Cournapeau as a Google Summer of Code project, and since then many volunteers have contributed. See the AUTHORS.rst file for a complete list of contributors.

It is currently maintained by a team of volunteers.

4.6.1 Installation

4.6.1.1 Dependencies

scikit-learn requires:

- Python (>= 2.7 or >= 3.4)
- NumPy (>= 1.8.2)
- SciPy (>= 0.13.3)

4.6.1.2 User installation

If you already have a working installation of numpy and scipy, the easiest way to install scikit-learn is using pip[17]

```
pip install -U scikit-learn
```

4.7 Keras

Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano. It was developed with a focus on enabling fast experimentation. Being able to go from idea to result with the least possible delay is key to doing good research.
Use Keras if you need a deep learning library that:

- Allows for easy and fast prototyping (through user friendliness, modularity, and extensibility).
- Supports both convolutional networks and recurrent networks, as well as combinations of the two.
- Runs seamlessly on CPU and GPU.[18]

4.7.1 Guiding principles

- **User friendliness.** Keras is an API designed for human beings, not machines. It puts user experience front and center. Keras follows best practices for reducing cognitive load: it offers consistent & simple APIs, it minimizes the number of user actions required for common use cases, and it provides clear and actionable feedback upon user error.
- **Modularity.** A model is understood as a sequence or a graph of standalone, fully-configurable modules that can be plugged together with as little restrictions as possible. In particular, neural layers, cost functions, optimizers, initialization schemes, activation functions, regularization schemes are all standalone modules that you can combine to create new models.
- **Easy extensibility.** New modules are simple to add (as new classes and functions), and existing modules provide ample examples. To be able to easily create new modules allows for total expressiveness, making Keras suitable for advanced research.
- **Work with Python**. No separate models configuration files in a declarative format. Models are described in Python code, which is compact, easier to debug, and allows for ease of extensibility.[18]

4.7.2 Installation

Before installing Keras, please install one of its backend engines: TensorFlow, Theano, or CNTK.

Then, you can install Keras itself. There are two ways to install Keras:

- **Install Keras from PyPI (recommended):** [18]

```
sudo pip install keras
```

4.8 Tensorflow

TensorFlow is an open source software library for high performance numerical computation. Its flexible architecture allows easy deployment of computation across a variety of platforms (CPUs, GPUs, TPUs), and from desktops to clusters of servers to mobile and edge devices. Originally developed by researchers and engineers from the Google Brain team within Google's AI organization, it comes with strong support for machine learning and deep learning and the flexible numerical computation core is used across many other scientific domains.[19]

4.8.1 Features

TensorFlow provides a Python API as well as C++, Haskell, Java, Go and Rust APIs. Third party packages are available for C#, Julia, R, Scala and OCaml.
A Javascript API was released by Tensorflow.org on March 30, 2018.

4.8.2 Applications

Among the applications for which TensorFlow is the foundation, are automated image captioning software, such as DeepDream. RankBrain now handles a substantial number of search queries, replacing and supplementing traditional static algorithm-based search results.[19]

5. Implementation

5.1 Methodology

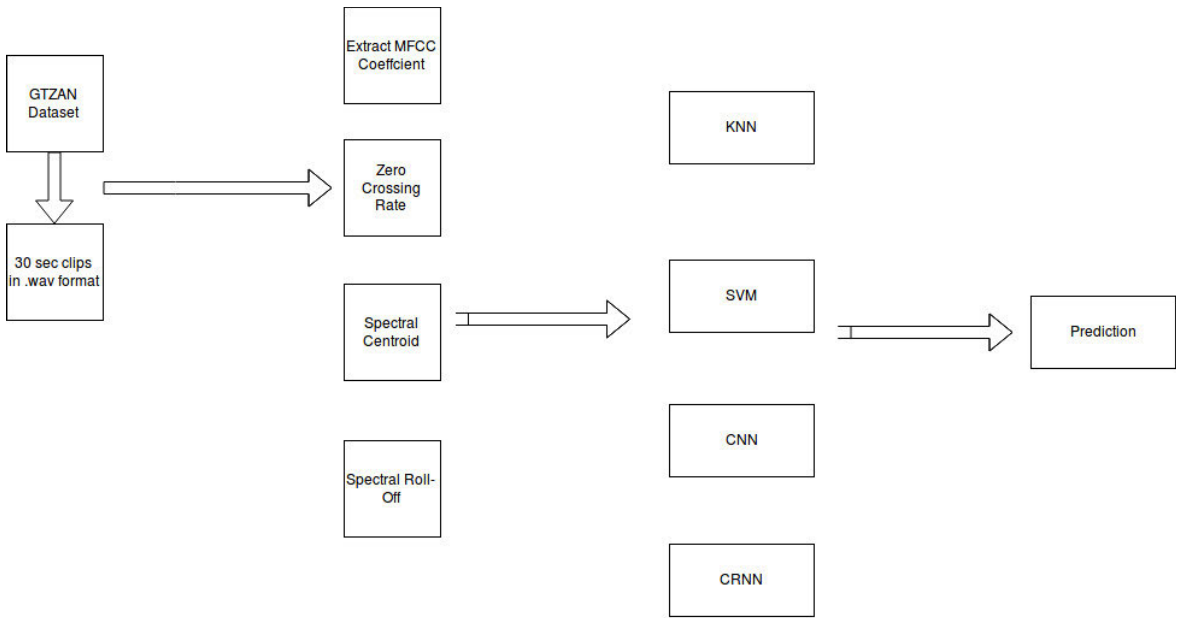


Fig 15 Diagrammatic representation of the method used

5.2 Data Retrieval Process

Marsyas (Music Analysis, Retrieval, and Synthesis for Audio Signals) is an open source software framework for audio processing with specific emphasis on Music Information Retrieval Applications. Its website also provides access to a database, GTZAN Genre Collection, of 1000 audio tracks each 30 second long. There are 10 genres represented, each containing 100 tracks. All the tracks are 22050Hz Mono 16-bit audio files in .au format. We have chosen four of the most distinct genres for our research: classical, jazz, metal, and pop because multiple previous work has indicated that the success rate drops when the number of classifications is above 4. Thus, our total data set was 400 songs, of which we used 70% for training and 30% for testing and measuring results. We wrote a python script to read in the audio files of the 100 songs per genre and combine them into a .csv file. We then read the .csv file into MATLAB and extract the MFCC features for each song. We further reduced this matrix representation of each song by taking the mean vector and covariance matrix of the cepstral features and storing them as a cell matrix, effectively modeling the frequency features of each song as a multi-variate Gaussian distribution.

5.2.1 Mel Frequency Cepstral Coefficients (MFCC)

For audio processing, we needed to find a way to concisely represent song waveforms. Existing music processing literature pointed us to MFCCs as a way to represent time domain waveforms as just a few frequency domain coefficients. 1 To compute the MFCC, we first read in the middle 50% of the mp3 waveform and take 20 ms frames at a parameterized interval. For each frame, we multiply by a hamming window to smooth the edges, and then take the Fourier Transform to get the frequency components. We then map the frequencies to the mel scale, which models human perception of changes in pitch, which is approximately linear below 1kHz and logarithmic above 1kHz. This mapping groups the frequencies into 20 bins by calculating triangle window coefficients based on the mel scale, multiplying that by the frequencies, and taking the log. We then take the Discrete Cosine Transform, which serves as an approximation of the Karhunen-Loeve Transform, to decorrelate the frequency components. Finally, we keep the first 15 of these 20 frequencies since higher frequencies are the details that make less of a difference to human perception and contain less information about the song. Thus, we represent each raw song waveform as a matrix of cepstral features, where each row is a vector of 15 cepstral frequencies of one 20 ms frame for a parameterized number of frames per song.

We further reduce this matrix representation of each song by taking the mean vector and covariance matrix of the cepstral features over each 20ms frame and storing them as a cell matrix. Modeling the frequencies as a multi-variate Gaussian distribution again compressed the computational requirements of comparing songs with KL Divergence.

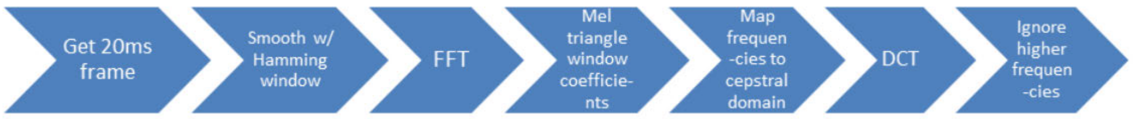

Get 20ms frame
Smooth w/ Hamming window
FFT
Mel triangle window coefficie-nts
Map frequen-cies to cepstral domain
DCT
Ignore higher frequen-cies

Fig-16 MFCC Flow

5.3 Techniques

5.3.1 Kullback-Lieber (KL) Divergence

The fundamental calculation in our k-NN training is to figure out the distance between two songs. We compute this via the Kullback-Leibler divergence [3]. Consider p(x) and q(x) to be the two multivariate Gaussian distributions with mean and covariance corresponding to those derived from the MFCC matrix for each song. Then, we have the following:

$$2KL(p||q) = \log \frac{|\Sigma_q|}{|\Sigma_p|} + Tr(\Sigma_q^{-1}\Sigma_p) + (\mu_p - \mu_q)^T \Sigma_q^{-1}(\mu_p - \mu_q) - d$$

However, since KL divergence is not symmetric, but the distance should be symmetric, we have:

$$D_{KL}(p, q) = KL(p||q) + KL(q||p)$$

5.3.2 kNN

The first machine learning technique we applied is the k-nearest neighbors (k-NN) because existing literature has shown it is effective considering its ease of implementation. The class notes on k nearest neighbors gave a succinct outline of the algorithm which served as our reference.[24]

```
nClasses = numpy.unique(self.Y).shape[0]
YDist = (distance.cdist(self.X, testSample.reshape(1,
testSample.shape[0]), 'euclidean')).T
iSort = numpy.argsort(YDist)
P = numpy.zeros((nClasses,))
for i in range(nClasses):
    P[i] = numpy.nonzero(self.Y[iSort[0][0:self.k]] == i)[0].shape[0]
/ float(self.k)
return (numpy.argmax(P), P)
```

5.3.3 CNN

CNN algorithm is designed to be used with images. A benefit of convolution network over other neural networks is that in a neural network an input node is required for each dimension. Taking an image as example, it would mean a different node for each pixel of the image. So, if an image has size 512*512 pixels we will need 2,62,144 input nodes. If the image is a colored instead of black and white, we will need 512*512*3 (=7,86,432) input nodes. (3 nodes for each red, green and blue channels). This no of input nodes is not feasible to program and does not give better results. Additionally, images of other sizes cannot be used with this model and with newer devices creating images of really large size, it is almost impossible to design these.

For context, a typical modern-day smartphone has a 12MP camera which produces an image of size 4000*3000 pixels. This image will require 3,60,00,000 input nodes.

A convolutional network reduces this requirement and by a very large amount. A single image can be recognized using a convolution network of any size ranging from 30 to any arbitrary value depending on the no of features extracted from the image.

Similarly, for music files a simple neural network will require an input node for every second or millisecond, if we need better precision. A convolution network only requires 32 input nodes in our model as only 32 features are extracted using MFCC.

We have used 5 hidden layers for processing the data along with 32 input layer and 10 output layers, one for each genre of music classified.

```
x = tf.reshape(melspectrogram,[-1,1,96,1366])
x = batch_norm(melspectrogram, 1366, phase_train)
x = tf.reshape(melspectrogram,[-1,96,1366,1])
conv2_1 = tf.add(tf.nn.conv2d(x, weights['wconv1'], strides=[1, 1, 1,
1], padding='SAME'), weights['bconv1'])
conv2_1 = tf.nn.relu(batch_norm(conv2_1, 32, phase_train))
mpool_1 = tf.nn.max_pool(conv2_1, ksize=[1, 2, 4, 1], strides=[1, 2,
4, 1], padding='VALID')
dropout_1 = tf.nn.dropout(mpool_1, 0.5)

conv2_2 = tf.add(tf.nn.conv2d(dropout_1, weights['wconv2'],
strides=[1, 1, 1, 1], padding='SAME'), weights['bconv2'])
conv2_2 = tf.nn.relu(batch_norm(conv2_2, 128, phase_train))
mpool_2 = tf.nn.max_pool(conv2_2, ksize=[1, 2, 4, 1], strides=[1, 2,
4, 1], padding='VALID')
dropout_2 = tf.nn.dropout(mpool_2, 0.5)

conv2_3 = tf.add(tf.nn.conv2d(dropout_2, weights['wconv3'],
strides=[1, 1, 1, 1], padding='SAME'), weights['bconv3'])
conv2_3 = tf.nn.relu(batch_norm(conv2_3, 128, phase_train))
mpool_3 = tf.nn.max_pool(conv2_3, ksize=[1, 2, 4, 1], strides=[1, 2,
4, 1], padding='VALID')
dropout_3 = tf.nn.dropout(mpool_3, 0.5)

conv2_4 = tf.add(tf.nn.conv2d(dropout_3, weights['wconv4'],
strides=[1, 1, 1, 1], padding='SAME'), weights['bconv4'])
conv2_4 = tf.nn.relu(batch_norm(conv2_4, 192, phase_train))
mpool_4 = tf.nn.max_pool(conv2_4, ksize=[1, 3, 5, 1], strides=[1, 3,
5, 1], padding='VALID')
dropout_4 = tf.nn.dropout(mpool_4, 0.5)

conv2_5 = tf.add(tf.nn.conv2d(dropout_4, weights['wconv5'],
strides=[1, 1, 1, 1], padding='SAME'), weights['bconv5'])
conv2_5 = tf.nn.relu(batch_norm(conv2_5, 256, phase_train))
mpool_5 = tf.nn.max_pool(conv2_5, ksize=[1, 4, 4, 1], strides=[1, 4,
4, 1], padding='VALID')
dropout_5 = tf.nn.dropout(mpool_5, 0.5)

flat = tf.reshape(dropout_5, [-1,
weights['woutput'].get_shape().as_list()[0]])
p_y_X =
tf.nn.sigmoid(tf.add(tf.matmul(flat,weights['woutput']),weights['bout
put']))
```

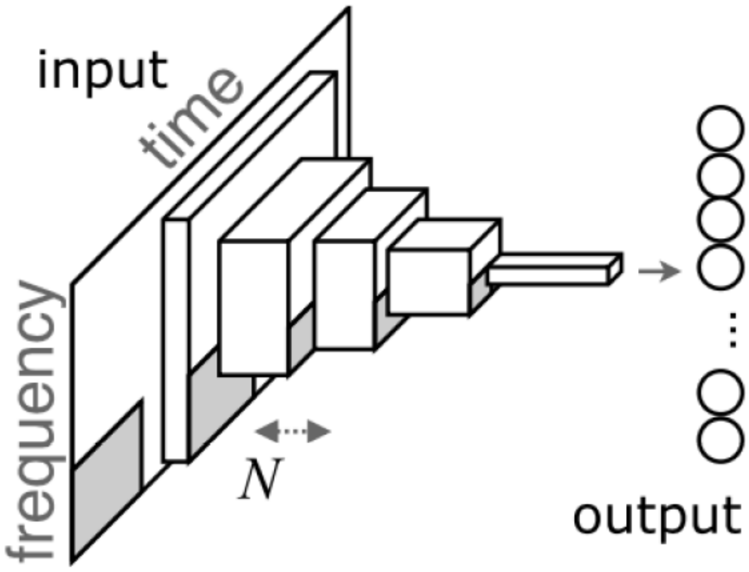


Fig 17 CNN model[21]

5.3.4 SVM

Here we are using kernel SVM not 'Linear' because our dataset will not be linearly separable if we visualize. We are using pyAudioAnanlysis library to extract all 34 features from music file. We have used '*.wav' as we have function in this library which can be used to convert all type of extension to .wav format.

```
ffmpegString = 'avconv -i ' + '\"' + f + '\"' + ' -ar ' +
str(samplingRate) + ' -ac ' + str(channels) + ' ' + '\"' +
os.path.splitext(f)[0] + '\"' + '.wav'
os.system(ffmpegString)
```

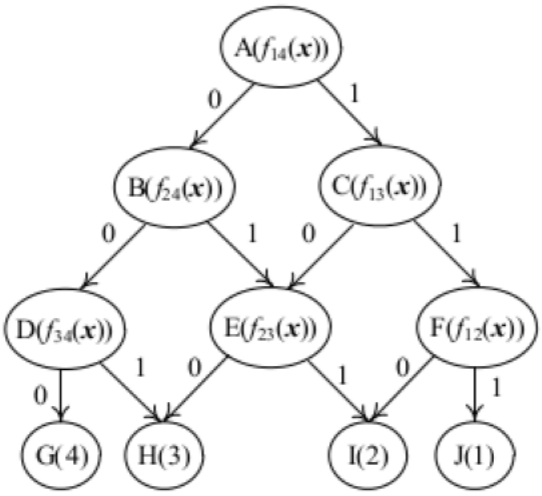


Fig 18 SVM model

So, when our features are extracted in a vector format we will feed these data to SVM model, but as mentioned before it is not possible to separate linearly. So we are using kernel SVM.

```
[X, Y] = listOfFeatures2Matrix(features)
svm = sklearn.svm.SVC(C = Cparam, kernel = 'linear',  probability = 
True)
svm.fit(X,Y)
```

After training our SVM classifier we will test on any file. Here we have selected 'isSignificant' level which is minimum probability to classify our test file.

```
isSignificant = 0.5
# P: list of probabilities
Result, P, classNames = aT.fileClassification(fileName, "svmModel", 
"svm")
winner = np.argmax(P)
```

5.3.5 CRNN

A CRNN model combines the use of best of CNN and RNN. It works similar to CNN but provides a recurrence for learning from previous outputs using backpropagation. This allows the model to improve its performance using the feedback received. A CRNN usually gives better accuracy as compared to CNN as it has the added benefit of feedback and delta weight optimization while training which is not available in CNN model.

In this, we have used the same CNN model used before but have modified it for recurrence.

```
x = 
tf.cast(tf.pad(melspectrogram,[[0,0],[0,0],[37,37],[0,0]],'CONST
ANT'),tf.float32)
x = batch_norm(tf.reshape(x,[-1,1,96,1440]), 1440, phase_train)
x = tf.reshape(x,[-1,96,1440,1])
conv2_1 = tf.add(tf.nn.conv2d(x, weights['wconv1'], strides=[1, 
1, 1, 1], padding='SAME'), weights['bconv1'])
conv2_1 = tf.nn.relu(batch_norm(conv2_1, 64, phase_train))
mpool_1 = tf.nn.max_pool(conv2_1, ksize=[1, 2, 2, 1], 
strides=[1, 2, 2, 1], padding='VALID')
dropout_1 = tf.nn.dropout(mpool_1, 0.5)

conv2_2 = tf.add(tf.nn.conv2d(dropout_1, weights['wconv2'], 
strides=[1, 1, 1, 1], padding='SAME'), weights['bconv2'])
```

```
conv2_2 = tf.nn.relu(batch_norm(conv2_2, 128, phase_train))
mpool_2 = tf.nn.max_pool(conv2_2, ksize=[1, 3, 3, 1],
strides=[1, 3, 3, 1], padding='VALID')
dropout_2 = tf.nn.dropout(mpool_2, 0.5)

conv2_3 = tf.add(tf.nn.conv2d(dropout_2, weights['wconv3'],
strides=[1, 1, 1, 1], padding='SAME'), weights['bconv3'])
conv2_3 = tf.nn.relu(batch_norm(conv2_3, 128, phase_train))
mpool_3 = tf.nn.max_pool(conv2_3, ksize=[1, 4, 4, 1],
strides=[1, 4, 4, 1], padding='VALID')
dropout_3 = tf.nn.dropout(mpool_3, 0.5)

conv2_4 = tf.add(tf.nn.conv2d(dropout_3, weights['wconv4'],
strides=[1, 1, 1, 1], padding='SAME'), weights['bconv4'])
conv2_4 = tf.nn.relu(batch_norm(conv2_4, 128, phase_train))
mpool_4 = tf.nn.max_pool(conv2_4, ksize=[1, 4, 4, 1],
strides=[1, 4, 4, 1], padding='VALID')
dropout_4 = tf.nn.dropout(mpool_4, 0.5)

gru1_in = tf.reshape(dropout_4,[-1, 15, 128])
gru1 = tf.nn.rnn_cell.MultiRNNCell([tf.nn.rnn_cell.GRUCell(32)]
* 15)
gru1_out, state = tf.nn.dynamic_rnn (gru1, gru1_in,
dtype=tf.float32, scope='gru1')

gru2 = tf.nn.rnn_cell.MultiRNNCell([tf.nn.rnn_cell.GRUCell(32)]
* 15)
gru2_out, state = tf.nn.dynamic_rnn(gru2, gru1_out,
dtype=tf.float32, scope='gru2')
gru2_out = tf.transpose(gru2_out, [1, 0, 2])
gru2_out = tf.gather(gru2_out, int(gru2_out.get_shape()[0]) - 1)
dropout_5 = tf.nn.dropout(gru2_out, 0.3)

flat = tf.reshape(dropout_5, [-1,
weights['woutput'].get_shape().as_list()[0]])
p_y_X =
tf.nn.sigmoid(tf.add(tf.matmul(flat,weights['woutput']),weights[
'boutput']))
return p_y_X
```

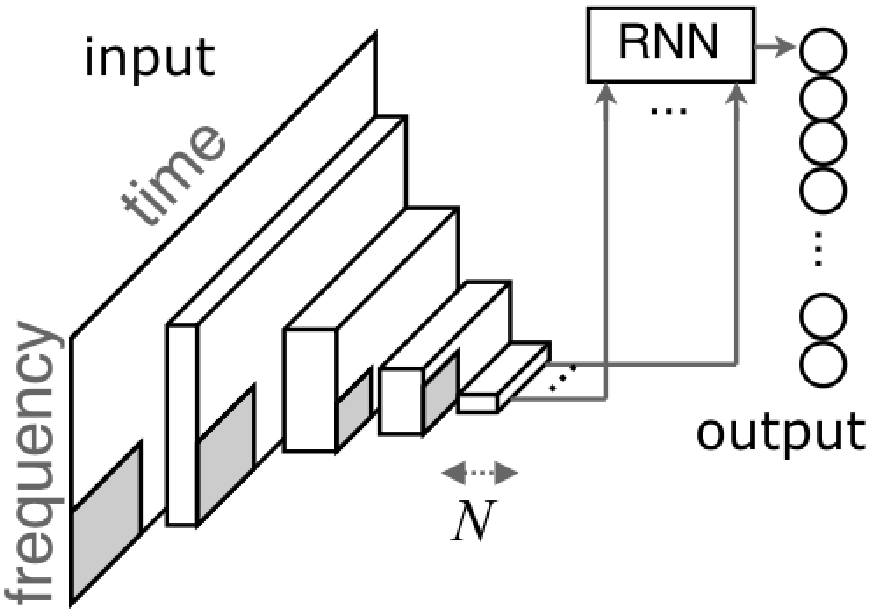


Fig 19 CRNN model

5.3.5.1 Transfer Learning

Transfer learning has proven to be very effective in the image processing scene, it studies and provides techniques on how adapt a model trained in large-scale database to perform well in other tasks different from the one that was trained for. We aim at learning from a source data distribution (multiclass tags) a well performing model on a different target data distribution (single class genres). Inside the transfer learning paradigm this is known as domain adaptation. The two most common practices and the ones that we will apply are:

- Using the network as feature extractor. That is removing the last fully-connected layer and treat the network as a feature extractor. Once we have extracted all the features at the top we can include a classifier like SVM or a Softmax classifier for the new dataset.
- Fine-tuning the network. This strategy is based on not only replace the classifier layer of the network, but also retrain part or the whole network. Through backpropagation we can modify the weights of the pre-trained model to adapt the model to the new data distribution. Sometimes its preferable to keep the first layers of the network fixed (or freezed) to avoid overfitting, and only fine-tune the deeper part. This is motivated because the lower layers of the networks capture generic features, that are similar to many tasks while the higher layers contain features that are task and dataset oriented.

5.3.5.2 Multiframe

We propose to use a multiframe strategy that allows us to extract more than one frame (or mel-spectogram image) per song. For each song we discard the first and last N seconds and then, we divide the rest into frames of equal time-length t. The final parameters are stated in the experiments. This approach has two advantages:

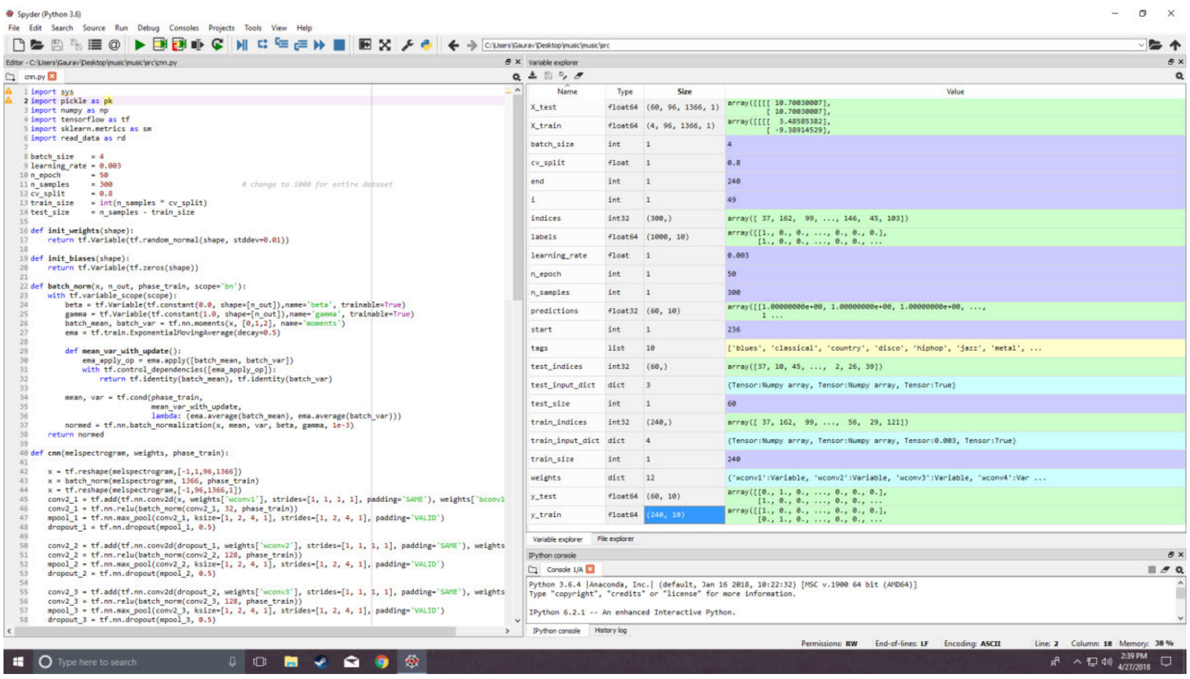
- At training time: we are able to generate more data to train the network than in the approach of, as they are only extracting the central part of the song. We are very limited by the number of songs, this is a very useful tool to provide data augmentation.
- At test time: we can average or perform a KNN with the scores of every frame to infer the genre tag for the complete song with more confidence.

5.3.6 k Means

For unsupervised k-means clustering to work on our feature set, we wrote a custom implementation because we had to determine how to represent cluster centroids and how to update to better centroids each iteration. To solve this, we chose to represent a centroid as if it were also a multi-variate Gaussian distribution of an arbitrary song (which may not actually exist in the data set) and initialized the four centroids as four random songs whose distances (as determined by KL divergence) were above a certain empirically determined threshold. Once a group of songs is assigned to a centroid, the centroid is updated according to the mean of the mean vectors and covariance matrices of those songs, thus represented as a new song that is the average of the real songs assigned to it. Finally, as random initialization in the beginning and number of iterations are the two factors with notable influence on the cluster outcomes, we determined the iteration number empirically and repeatedly run k-means with different random initial centroids and pick the best, as determined by the calculated total percent accuracy.

5.4 Training

We have fine-tuned two different networks, a CNN and a CRNN. We have made experiments by freezing the lowest layers and fine-tuning different top layers to see the differences. The parameters have been set as in standard fine-tuning, setting the learning rate a bit slower than in the original model. We have tried two different optimizers, the adaptive learning rate ADAM method as in the original model work and SGD with Nesterov Momentum as it is widely used in machine learning. We set categorical cross-entropy as the loss function. Batch normalization and dropout layers are implemented as the original authors did. To perform the training, we use all of our handmade dataset as training data (2215 frames) and the GTZAN dataset as testing data (1000 frames).


Spyder (Python 3.6)
File Edit Search Source Run Debug Consoles Projects Tools View Help
C:\Users\Gaurav\Desktop\music\music\src
Editor - C:\Users\Gaurav\Desktop\music\music\src\cnn.py
cnn.py
Variable explorer
File explorer
IPython console
Console 1/A
History log
Type here to search

```python
import sys
import pickle as pk
import numpy as np
import tensorflow as tf
import sklearn.metrics as sm
import read_data as rd

batch_size    = 4
learning_rate = 0.003
n_epoch       = 50
n_samples     = 300                              # change to 1000 for entire dataset
cv_split      = 0.8
train_size    = int(n_samples * cv_split)
test_size     = n_samples - train_size

def init_weights(shape):
    return tf.Variable(tf.random_normal(shape, stddev=0.01))

def init_biases(shape):
    return tf.Variable(tf.zeros(shape))

def batch_norm(x, n_out, phase_train, scope='bn'):
    with tf.variable_scope(scope):
        beta = tf.Variable(tf.constant(0.0, shape=[n_out]),name='beta', trainable=True)
        gamma = tf.Variable(tf.constant(1.0, shape=[n_out]),name='gamma', trainable=True)
        batch_mean, batch_var = tf.nn.moments(x, [0,1,2], name='moments')
        ema = tf.train.ExponentialMovingAverage(decay=0.5)

        def mean_var_with_update():
            ema_apply_op = ema.apply([batch_mean, batch_var])
            with tf.control_dependencies([ema_apply_op]):
                return tf.identity(batch_mean), tf.identity(batch_var)

        mean, var = tf.cond(phase_train,
                            mean_var_with_update,
                            lambda: (ema.average(batch_mean), ema.average(batch_var)))
        normed = tf.nn.batch_normalization(x, mean, var, beta, gamma, 1e-3)
    return normed

def cnn(melspectrogram, weights, phase_train):

    x = tf.reshape(melspectrogram,[-1,1,96,1366])
    x = batch_norm(melspectrogram, 1366, phase_train)
    x = tf.reshape(melspectrogram,[-1,96,1366,1])
    conv2_1 = tf.add(tf.nn.conv2d(x, weights['wconv1'], strides=[1, 1, 1, 1], padding='SAME'), weights['bconv1
    conv2_1 = tf.nn.relu(batch_norm(conv2_1, 32, phase_train))
    mpool_1 = tf.nn.max_pool(conv2_1, ksize=[1, 2, 4, 1], strides=[1, 2, 4, 1], padding='VALID')
    dropout_1 = tf.nn.dropout(mpool_1, 0.5)

    conv2_2 = tf.add(tf.nn.conv2d(dropout_1, weights['wconv2'], strides=[1, 1, 1, 1], padding='SAME'), weights
    conv2_2 = tf.nn.relu(batch_norm(conv2_2, 128, phase_train))
    mpool_2 = tf.nn.max_pool(conv2_2, ksize=[1, 2, 4, 1], strides=[1, 2, 4, 1], padding='VALID')
    dropout_2 = tf.nn.dropout(mpool_2, 0.5)

    conv2_3 = tf.add(tf.nn.conv2d(dropout_2, weights['wconv3'], strides=[1, 1, 1, 1], padding='SAME'), weights
    conv2_3 = tf.nn.relu(batch_norm(conv2_3, 128, phase_train))
    mpool_3 = tf.nn.max_pool(conv2_3, ksize=[1, 2, 4, 1], strides=[1, 2, 4, 1], padding='VALID')
    dropout_3 = tf.nn.dropout(mpool_3, 0.5)
```

Name	Type	Size	Value
X_test	float64	(60, 96, 1366, 1)	array([[[[10.70030007], [10.70030007],
X_train	float64	(4, 96, 1366, 1)	array([[[[3.48585382], [-9.38914529],
batch_size	int	1	4
cv_split	float	1	0.8
end	int	1	240
i	int	1	49
indices	int32	(300,)	array([37, 162, 99, ..., 146, 45, 103])
labels	float64	(1000, 10)	array([[1., 0., 0., ..., 0., 0., 0.], [1., 0., 0., ..., 0., 0., ...
learning_rate	float	1	0.003
n_epoch	int	1	50
n_samples	int	1	300
predictions	float32	(60, 10)	array([[1.00000000e+00, 1.00000000e+00, 1.00000000e+00, ..., 1 ...
start	int	1	236
tags	list	10	['blues', 'classical', 'country', 'disco', 'hiphop', 'jazz', 'metal', ...
test_indices	int32	(60,)	array([37, 10, 45, ..., 2, 26, 39])
test_input_dict	dict	3	{Tensor:Numpy array, Tensor:Numpy array, Tensor:True}
test_size	int	1	60
train_indices	int32	(240,)	array([37, 162, 99, ..., 56, 29, 121])
train_input_dict	dict	4	{Tensor:Numpy array, Tensor:Numpy array, Tensor:0.003, Tensor:True}
train_size	int	1	240
weights	dict	12	{'wconv1':Variable, 'wconv2':Variable, 'wconv3':Variable, 'wconv4':Var ...
y_test	float64	(60, 10)	array([[0., 1., 0., ..., 0., 0., 0.], [1., 0., 0., ..., 0., 0., ...
y_train	float64	(240, 10)	array([[1., 0., 0., ..., 0., 0., 0.], [0., 1., 0., ..., 0., 0., ...

```
Python 3.6.4 |Anaconda, Inc.| (default, Jan 16 2018, 10:22:32) [MSC v.1900 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 6.2.1 -- An enhanced Interactive Python.
```

```
Permissions: RW   End-of-lines: LF   Encoding: ASCII   Line: 2   Column: 18   Memory: 38 %
2:39 PM
4/27/2018
```

Fig 20 Training CNN model

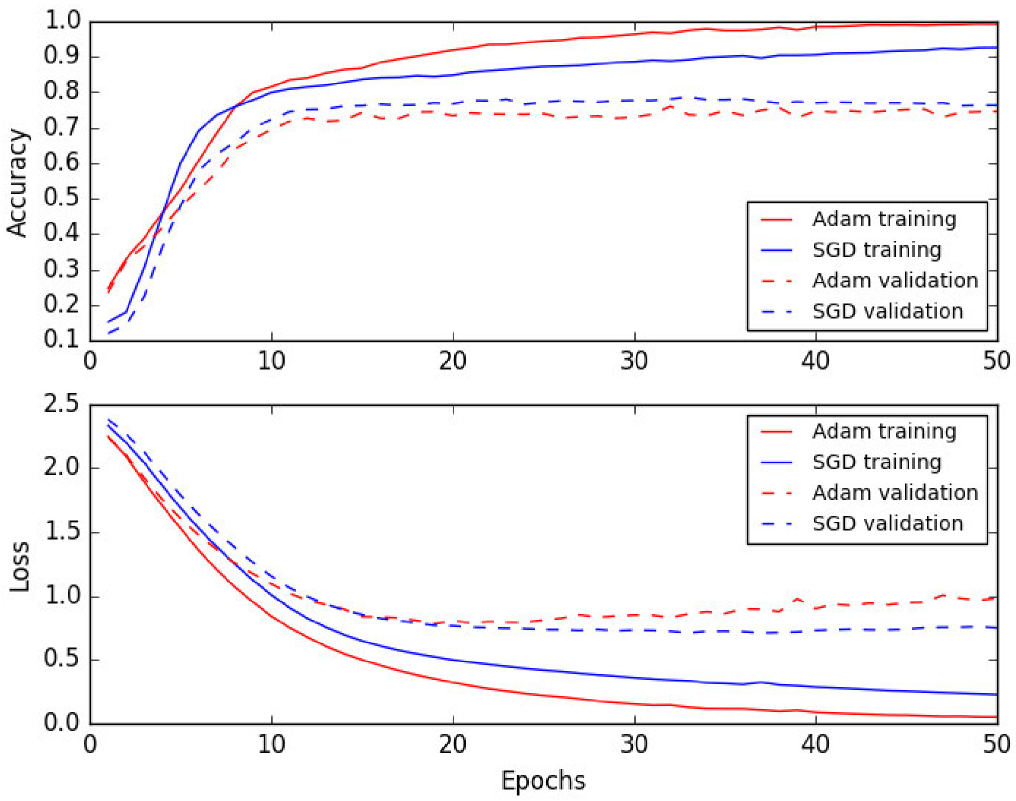


Fig 21 Training on model

5.5 Running Screenshots

We have designed a frontend for our model using Flask framework. We have used HTML and CSS to create a web page that allows the user to input the path of a file.

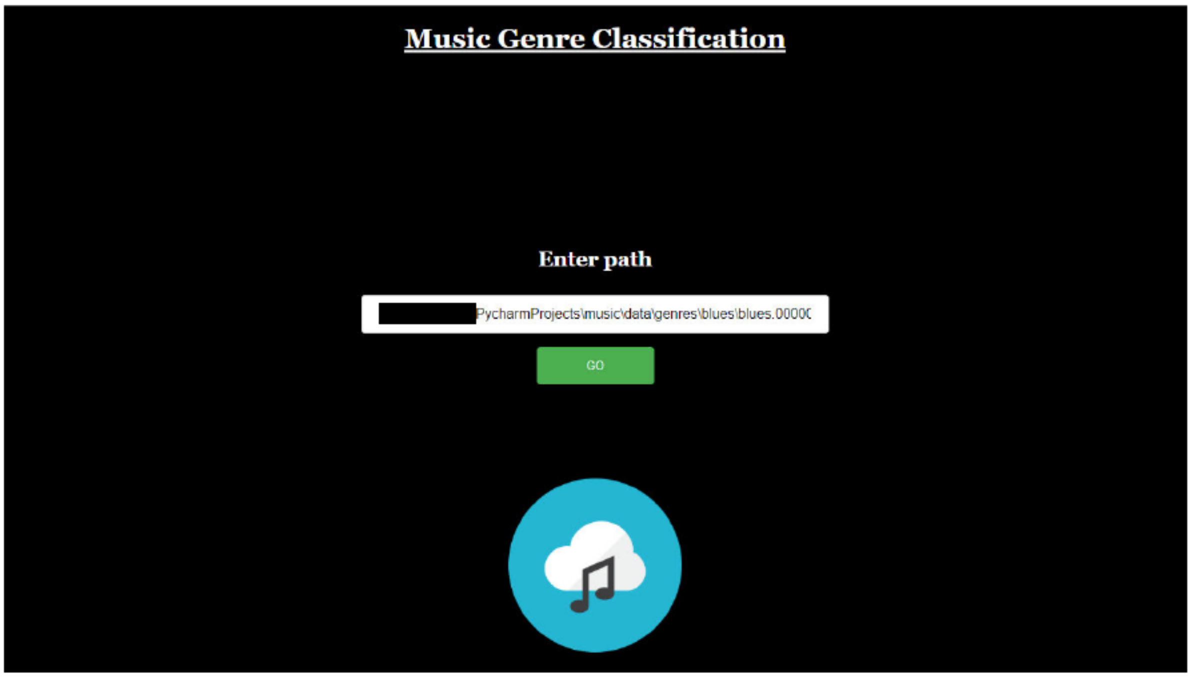

Music Genre Classification
Enter path
[redacted]PycharmProjects\music\data\genres\blues\blues.0000(
GO

Fig 22 Screen showing the page where the user has to input a file for analysis

The user presses GO and a new screen is shown as shown in figure 22. This screen shows the label predicted by the model.

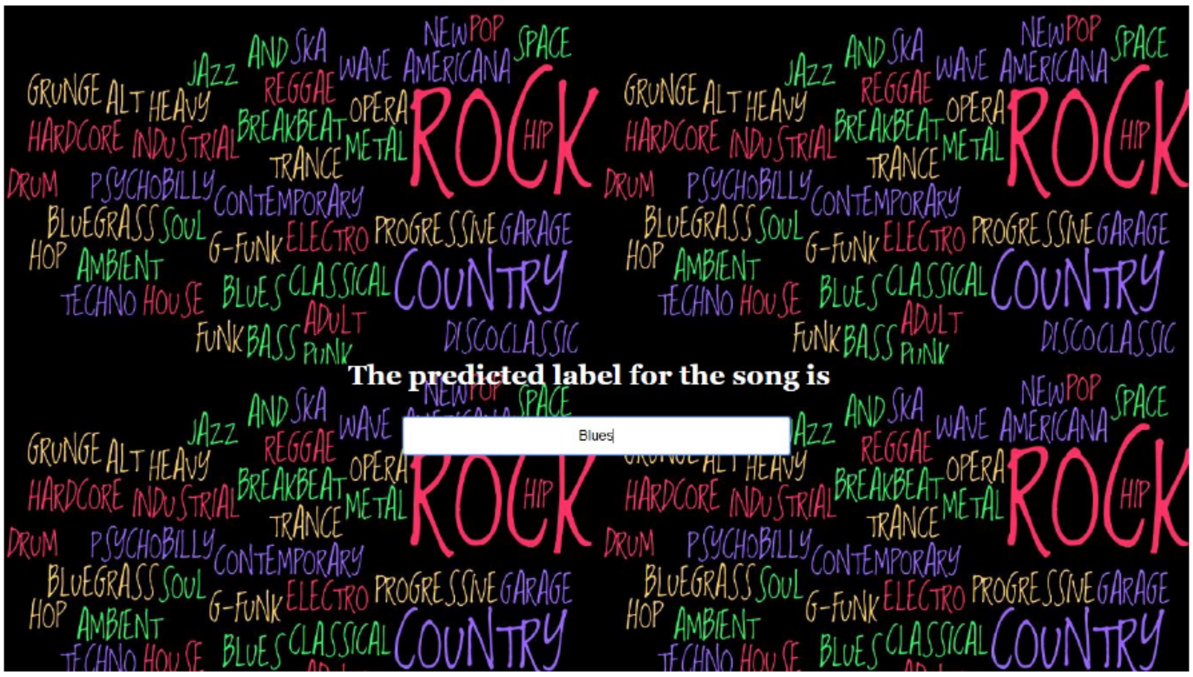


Fig 23 Screen showing the predicted label

5.6 Result and observation

We can observe in figure 2 that ADAM method converges faster than SGD. However, the plot shows that SGD is generalizing a bit better, although both are reaching overfitting in approximately 25 epochs. We decide to use ADAM to perform the rest of experiments. We show the comparison of fine-tuning both architectures. For the CNN case, we can observe

that it gets overfitted very fast, in less than 10 epochs. The best results are produced when we only train a classifier layer in the top of the network. However, as we can see in the plot, the network is getting stalled and it cannot improve more because of the lack of data. On the other hand, with the CRNN, we can see the same behavior, if we fine-tune the recurrent layers and the last convolutional layer, the network is getting overfitted very fast. As in this case the network complexity is increased it improves more before convergence. If only a classifier is trained we observe the same as in the CNN case, it needs more data to improve the performance and not getting stalled.
The best results are around 78%.

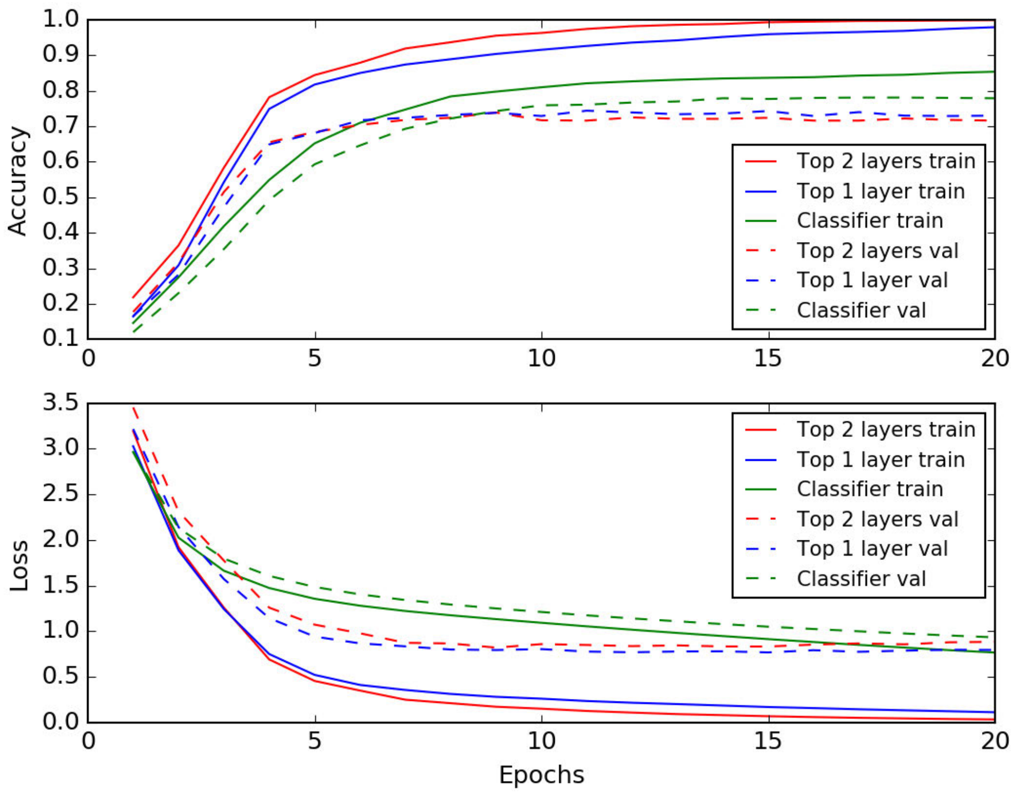


Fig 24 Training on CNN architecture

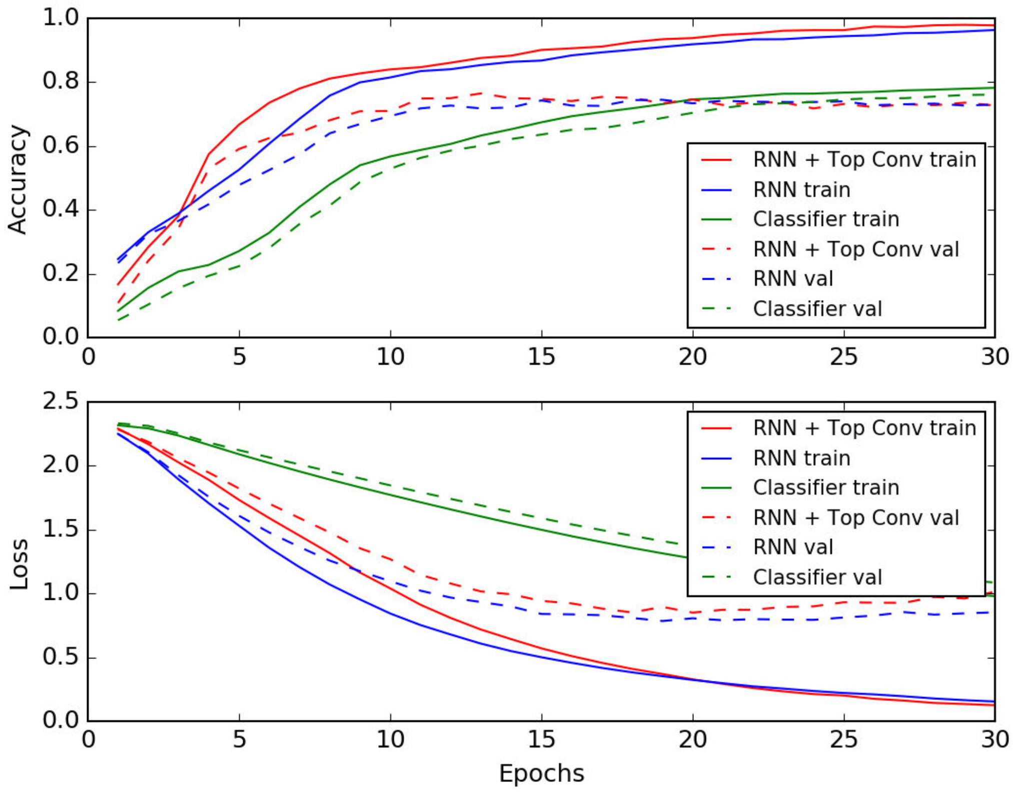
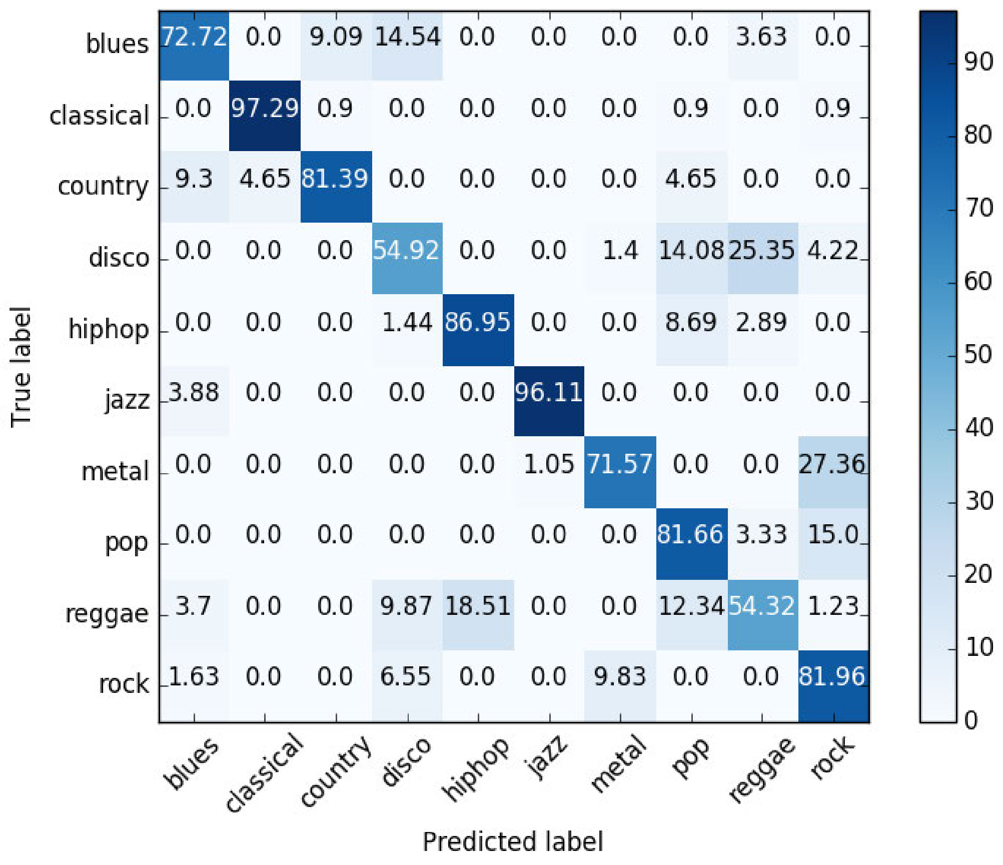


Fig 25 Training on CRNN architecture

5.6.1 Confusion matrix



	blues	classical	country	disco	hiphop	jazz	metal	pop	reggae	rock
blues	72.72	0.0	9.09	14.54	0.0	0.0	0.0	0.0	3.63	0.0
classical	0.0	97.29	0.9	0.0	0.0	0.0	0.0	0.9	0.0	0.9
country	9.3	4.65	81.39	0.0	0.0	0.0	0.0	4.65	0.0	0.0
disco	0.0	0.0	0.0	54.92	0.0	0.0	1.4	14.08	25.35	4.22
hiphop	0.0	0.0	0.0	1.44	86.95	0.0	0.0	8.69	2.89	0.0
jazz	3.88	0.0	0.0	0.0	0.0	96.11	0.0	0.0	0.0	0.0
metal	0.0	0.0	0.0	0.0	0.0	1.05	71.57	0.0	0.0	27.36
pop	0.0	0.0	0.0	0.0	0.0	0.0	0.0	81.66	3.33	15.0
reggae	3.7	0.0	0.0	9.87	18.51	0.0	0.0	12.34	54.32	1.23
rock	1.63	0.0	0.0	6.55	0.0	0.0	9.83	0.0	0.0	81.96

Fig 26 Using all frames

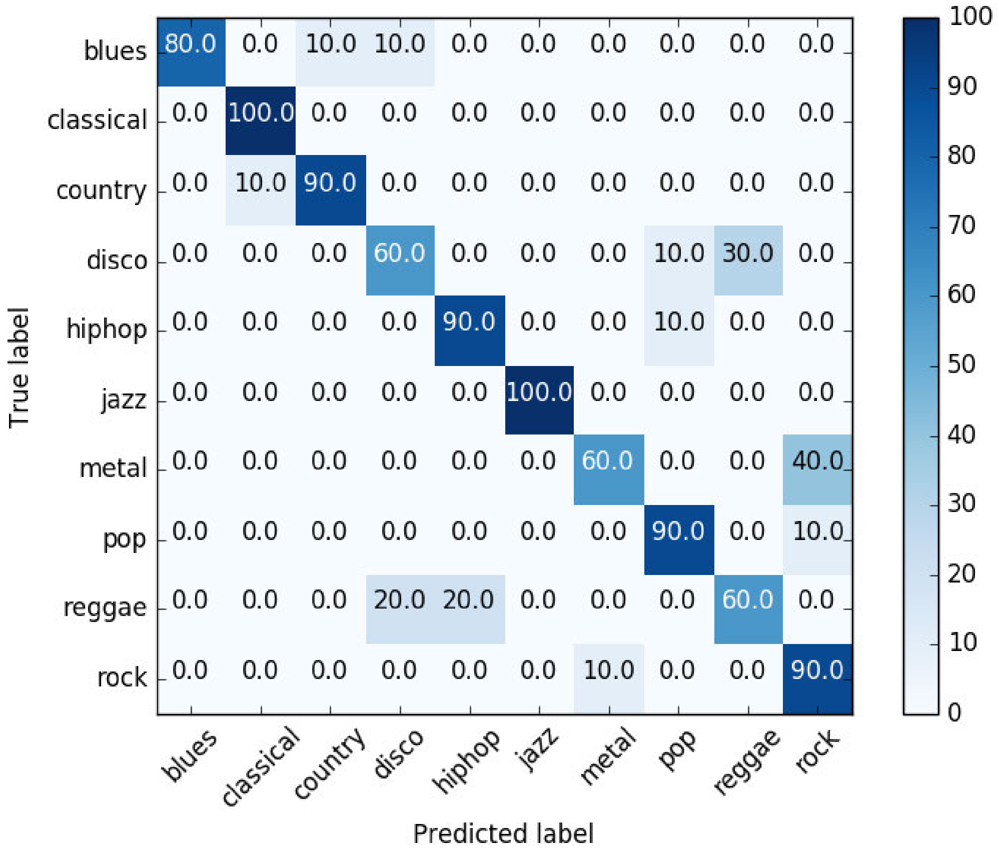


Fig 27 Using the mean

Classification accuracy varied between the different machine learning techniques and genres. The SVM had a success rate of only 66% when identifying jazz , most frequently misidentifying it as classical or metal. The Neural Network did worst when identifying metal with a 76% success rate. Interestingly, the Neural Network only ever misidentified metal as jazz. k-Means did well identifying all genres but Jazz, which was confused with Classical 36% of the time. k-NN had difficulty differentiating between Metal and Jazz in both directions. Of its 33% failures identifying Jazz, it misidentifies as Metal 90% of the time. Similarly, k-NN incorrectly predicted that Metal songs would be Jazz in 66% of all its failed Metal identifications. Overall we found that k-NN and k-means yielded similar accuracies of about 80%. A DAG SVM gave about 87% accuracy and neural networks gave 96% accuracy.

5.6.2 SVM Result

		Actual			
		Classical	Jazz	Metal	Pop
Predicted	Classical	29	4	1	1
	Jazz	1	20	1	0
	Metal	0	4	26	0
	Pop	0	2	2	29
Accuracy		97%	67%	87%	97%

Table - 1 Result of SVM model

5.6.3 K means result

		Actual			
		Classical	Jazz	Metal	Pop
Predicted	Classical	14	16	0	0
	Jazz	2	27	1	0
	Metal	0	0	27	3
	Pop	0	1	1	28
Accuracy		88%	61%	93%	90%

Table - 2 Result of k Means model

5.6.4 kNN result

		Actual			
		Classical	Jazz	Metal	Pop
Predicted	Classical	26	9	0	2
	Jazz	4	20	4	1
	Metal	0	1	24	0
	Pop	0	0	2	27
Accuracy		87%	67%	80%	90%

Table - 3 Result of kNN model

5.6.5 Neural network result

		Actual			
		Classical	Jazz	Metal	Pop
Predicted	Classical	14	0	0	0
	Jazz	1	12	4	0
	Metal	0	0	13	0
	Pop	1	0	0	19
Accuracy		88%	100%	76%	100%

Table - 4 Result of neural network model

6. Conclusion and Future Scope

6.1 Conclusion

Musical genres are categorical labels created by humans to characterize pieces of music. A musical genre is characterized by the common characteristics shared by its members. These characteristics typically are related to the instrumentation, rhythmic structure, and harmonic content of the music. Genre hierarchies are commonly used to structure the large collections of music available on the Web. Currently musical genre annotation is performed manually. Automatic musical genre classification can assist or replace the human user in this process and would be a valuable addition to music information retrieval systems. In addition, automatic musical genre classification provides a framework for developing and evaluating features for any type of content-based analysis of musical signals.

6.2 Future Scope

With the successful completion of the detection of various genres of music, the future goals involve increasing the number of classes for genres. Also, the better resources such as GPU would significantly decrease the computation time in training the module.
Furthermore, our system can be extended to Music Recommendation System as well as Music Playlist Generation.

7. References

[1] SAS,&quot;https://www.sas.com/en_us/insights/analytics/machine-learning.html,&quot;[Online].

[2] Silver, David; Huang, Aja; Maddison, Chris J.; Guez, Arthur; Sifre, Laurent; Driessche, George van den; Schrittwieser, Julian; Antonoglou, Ioannis; Panneershelvam, Veda (January 2016). "Mastering the game of Go with deep neural networks and tree search".

[3] Bengio, Yoshua (2009). "Learning Deep Architectures for AI" (PDF). Foundations and Trends in Machine Learning.

[4] https://en.wikipedia.org/wiki/Deep_learning

[5] Startups, Requests for (2016-08-12). "Deep Learning in Healthcare: Challenges and Opportunities". Medium.

[6] http://recognize-speech.com/feature-extraction/mfcc

[7] J. Hinton, Coursera lectures on Neural Networks, 2012, Url: https://www.coursera.org/learn/neural-networks

[8] https://en.wikipedia.org/wiki/Recurrent_neural_network

[9] https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm

[10] https://en.wikipedia.org/wiki/Support_vector_machine

[11] https://en.wikipedia.org/wiki/K-means_clustering

[12] https://en.wikipedia.org/wiki/Flask_(web_framework)

[13] https://en.wikipedia.org/wiki/HTML

[14] https://en.wikipedia.org/wiki/Cascading_Style_Sheets

[15] https://librosa.github.io/

[16] http://marsyasweb.appspot.com/download/data_sets/

[17] https://github.com/scikit-learn/scikit-learn

[18] https://keras.io/

[19] https://www.tensorflow.org/

[20] LeCun, Yann. "LeNet-5, convolutional neural networks". Retrieved 27 April 2018.

[21] Zhang, Wei (1990). "Parallel distributed processing model with local space-invariant interconnections and its optical architecture". Applied Optics. 29 (32): 4790–7. Bibcode:1990ApOpt..29.4790Z. doi:10.1364/AO.29.004790. PMID 20577468.

[22] Krizhevsky, Alex. "ImageNet Classification with Deep Convolutional Neural Networks"(PDF). Retrieved 17 November 2013.

[23] Min Xu; et al. (2004). "HMM-based audio keyword generation". In Kiyoharu Aizawa; Yuichi Nakamura; Shin'ichi Satoh. Advances in Multimedia Information Processing – PCM 2004: 5th Pacific Rim Conference on Multimedia (PDF). Springer. ISBN 3-540-23985-5. Archived from the original (PDF) on 2007-05-10.

[24] Altman, N. S. (1992). "An introduction to kernel and nearest-neighbor nonparametric regression". The American Statistician. 46 (3): 175–185. doi:10.1080/00031305.1992.10475879.

[25] https://medium.com/machine-learning-101/chapter-2-svm-support-vector-machine-theory-f0812effc72

[26] Altman, N. S. (1992). "An introduction to kernel and nearest-neighbor nonparametric regression". The American Statistician. 46 (3): 175–185. doi:10.1080/00031305.1992.10475879.