Guitar Chord Recognition

Araju Nepal
CE 4th year
araju7nepal@gmail.com

Manasi Kattel
CE 4th year
manasikattel1@gmail.com

Ayush Kumar Shah
CE 4th year
ayush.kumar.shah@gmail.com

Deepesh Shrestha
CE 4th year
deepeshshrestha@outlook.com

Abstract

The convolutional neural networks (CNN) have been found very effective to learn patterns from audios and hence can be used for sound classification. In this study, we first try to build a deep convolutional neural network architecture and adjust this neural network for guitar chord recognition i.e. to classify the input tunes into 10 guitar chords classes like A, A-minor, B, B-minor, G, etc. These categorizations are based on the different chords present in a guitar. We also consider using data augmentation for improving the accuracy of the classification as there are few datasets available.

1 Introduction

Over the past few years the need of music information retrieval and classification systems has become more urgent and this brought to the birth of a research area called Music Information Retrieval (MIR). It is an important task in the analysis of music and music transcription in general, and it can contribute to applications such as key detection, structural segmentation, music similarity measures, and other semantic analysis tasks. The Automatic Chord Recognition (ACR) task is one of the main challenges in MIR. Despite early successes in chord detection by using pitch chroma features and Hidden Markov Models (HMMs), recent attempts at further increasing the detection accuracy are only met with moderate success.

In this work, we investigate Convolutional Neural Networks (CNNs) for learning chroma features in the context of chord recognition, effectively replacing the widely used pitch chroma intermediate representation.

2 Related Works

2.1 Chord Detection using Deep Learning

X. Zhou and A. Lerch (2017) utilized deep learning to learn high-level features for audio chord detection. They represented input audio into samples with sample rate of 11.056 kHz and then applied Constant Q Transform (CQT). They used Principal Component Analysis (PCA) for decorrelation, and applied Z-Score normalization. For pre-processing, time splicing followed by Convolution was done. Time splicing is a simple way to extend the current frame with the data of neighboring frames by concatenating the frames into one larger superframe. For training, a standard back propagation can be applied after pre-training to fine-tune the network in a supervised manner. The loss criterion used in this work is cross-entropy.

2.2 Deep Convolutional Neural Networks and Data Augmentation for Environmental Sound Classification

Salamon & Bello (2017) in their paper have proposed a deep convolutional neural network architecture for environmental sound classification. They have also proposed the use of audio data augmentation for overcoming the problem of data scarcity and explore the influence of different augmentations on the performance of the proposed CNN architecture.

2.3 Neural networks for musical chords recognition

Osmalskyj, et. al. (2012) in this paper have considered the challenging problem of music recognition and presented an effective machine learning based method using a feed-forward neural network for chord recognition. They have used the known feature vector for automatic chord recognition called the Pitch Class Profile (PCP). Although the PCP vector only provides attributes corresponding to 12 semitone values, they have shown that it is adequate for chord recognition. Their experiments establish a twofold result: (1) the PCP is well suited for describing chords in a machine learning context, and (2) the algorithm is also capable to recognize chords played with other instruments, even unknown from the training phase.

3 Methods

3.1 Software Specification

Front End Tools:

Programming language - Python 3.6.8
Operating System - Ubuntu linux,Windows

Back End Tools : Jupyter Notebook, Anaconda, Sublime Text Editor

Libraries used:

1. **Keras :** It is a high-level neural networks API, written in Python and capable of running on top of TensorFlow.

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

3. **Numpy** : It is the fundamental package for scientific computing with Python and can be used as an efficient multi-dimensional container of generic data.

4. **Pandas** : It takes data (like a CSV or TSV file, or a SQL database) and creates a Python object with rows and columns called data frame that looks very similar to table in a statistical software.

5. **Random** : It generates a random float uniformly in the semi-open range [0.0, 1.0)

6. **Tensorflow-gpu**: It performs highly parallelised computation. It is basically a mini supercomputer running in your PC.

3.2 Methodology

A. Dataset:

Guitar Chords Recognition dataset

We collected the chords dataset from MONTEFIORE RESEARCH GROUP of University of Liège - Montefiore Institute (Montefiore.ulg.ac.be, 2019). The chords dataset consists of 10 types of chords with 200 audio files of each chord. However, only 633 out of 2000 files of equal number of frame size were obtained. The chords are:

1. A 2. Am 3. Bm 4. C 5. D 6. Dm 7. E 8. Em 9. F 10. G

The data set has been divided into two parts:
1. Training Set [1-500]
2. Test Set [501-633]

We also created a metadata called Chords.csv which contains information such as class_id, classname, file_name of all the audio guitar chords datasets. This metadata is used for further processing such as loading of audio files and extraction of chroma features.

B. Mel-spectrogram (Chroma features) extraction

In music, the term chroma feature or chromagram closely relates to the twelve different pitch classes. Chroma-based features, which are also referred to as "pitch class profiles", are a powerful tool for analyzing music whose pitches can be meaningfully categorized (often into twelve categories) and whose tuning approximates to the equal-tempered scale. One main property of chroma features is that they capture harmonic and melodic characteristics of music, while being robust to changes in timbre and instrumentation.

Identifying pitches that differ by an octave, chroma features show a high degree of robustness to variations in timbre and closely correlate to the musical aspect of harmony. This is the reason why chroma features are a well-established tool for processing and analyzing music data. For example, basically every chord recognition procedure relies on some kind of chroma representation. Also, chroma features have become the de facto standard for tasks such as music alignment and synchronization as well as audio structure analysis. (En.wikipedia.org, 2019)

Extraction process

- Speech is analyzed over short analysis window
- For each short analysis window a spectrum is obtained using FFT (Fast Fourier Transform)
- Spectrum is passed through Mel-Filters to obtain MelSpectrum

These Mel-filters are non-uniformly spaced on the frequency axis – more filters in the low frequency regions and less no. of filters in high frequency regions (similar to human ear)

An object of type MelSpectrogram represents an acoustic time-frequency representation of a sound: the power spectral density $P(f, t)$. It is sampled into a number of points around equally spaced times t_i and frequencies f_j (on a Mel frequency scale).

The mel frequency scale is defined as:

mel = 2595 * log10 (1 + hertz / 700)

We used librosa library to extract mel spectrogram chroma features from the audio datasets. We built a new dataset consisting of mel spectrograms of the chroma features of all the audio files in the guitar chord datasets as input and corresponding class_id as output. Since the audio files in our dataset are of varying duration (up to 4 s), we fixed the size of the input to 2 seconds (128 frames), i.e. $X \in \mathbf{R}^{128 \times 87}$

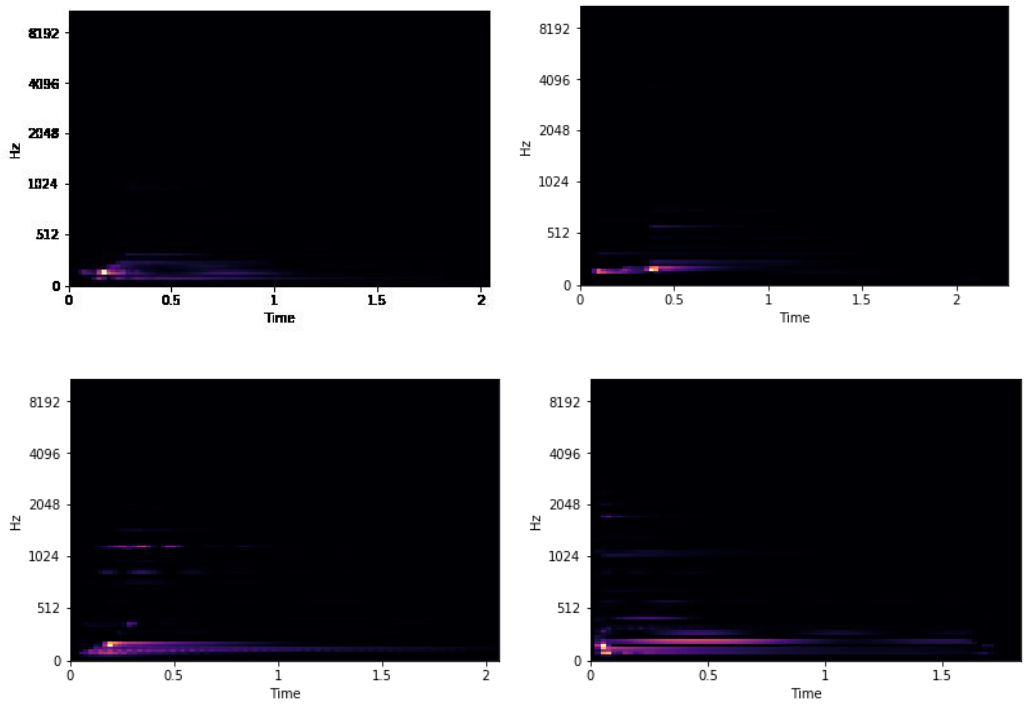

Fig 3.2.1: Mel Spectrogram Chroma features of audio files am89.wav, em69.wav, g200.wav and dm100.wav respectively from guitar chord dataset

C. Preprocessing of datasets

Thus, the input of the dataset was a numpy array of dimensions 128 X 87 and output was an integer class_id (0 to 9) corresponding to respective 10 classes of chords. We used this new dataset for further processing rather than the audio datasets. The dataset was then shuffled and split into training and test datasets.

The training and test datasets were further split into input (X_train and X_test) and output (y_train and y_test). The input dataset was reshaped to dimensions (128, 87, 1) for CNN input and one hot encoding of the output values were performed.

One hot encoding is a representation of categorical variables as binary vectors. This first requires that the categorical values be mapped to integer values. Then, each integer value is represented as a binary vector that is all zero values except the index of the integer, which is marked with a 1.
Eg. 3 is changed to [0 0 0 1 0 0 0 0 0 0]

D. Building a Convolution Neural Network:

Convolutional neural networks are deep artificial neural networks that are used primarily to classify images (e.g. name what they see), cluster them by similarity (photo search), and perform object recognition within scenes. They are algorithms that can identify faces, individuals, street signs, tumors, platypuses and many other aspects of visual data.

ConvNet architectures make the forward function more efficient to implement and vastly reduce the amount of parameters in the network. CNN take advantage of the fact that the input consists of images and they constrain the architecture in a more sensible way. In particular, unlike a regular Neural Network, the layers of a ConvNet have neurons arranged in 3 dimensions: width, height, depth.

A simple ConvNet is a sequence of layers, and every layer of a ConvNet transforms one volume of activations to another through a differentiable function. We use three main types of layers to build ConvNet architectures: Convolutional Layer, Pooling Layer, and Fully-Connected Layer.

Convolutional Layer : It is the core building block of a Convolutional Network that does most of the computational heavy lifting. Convolution operations are performed in this layer using a filter.

Convolution operation:

$$(I * h)(x, y) = \int_0^x \int_0^y I(i, j).h(x - i, y - j)didj$$
$$= \int_0^x \int_0^y I(x - i, y - j).h(i, j)didj$$

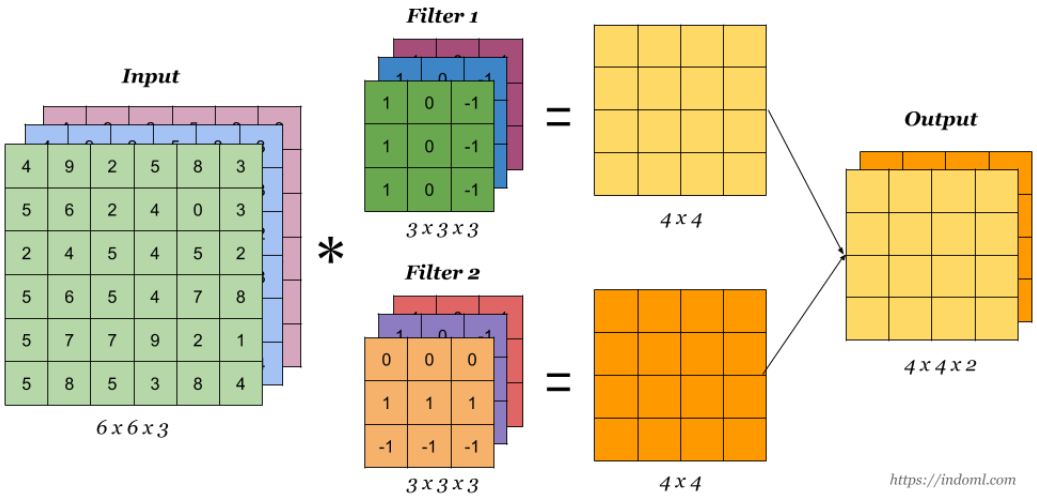

Fig 3.2.2: Convolution operation

Pooling Layer : Its function is to progressively reduce the spatial size of the representation to reduce the amount of parameters and computation in the network, and hence to also control overfitting. The Pooling Layer operates independently on every depth slice of the input and resizes it spatially, using the MAX operation. The most common form is a pooling layer with filters of size 2x2 applied with a stride of 2 downsamples every depth slice in the input by 2 along both width and height, discarding 75% of the activations.

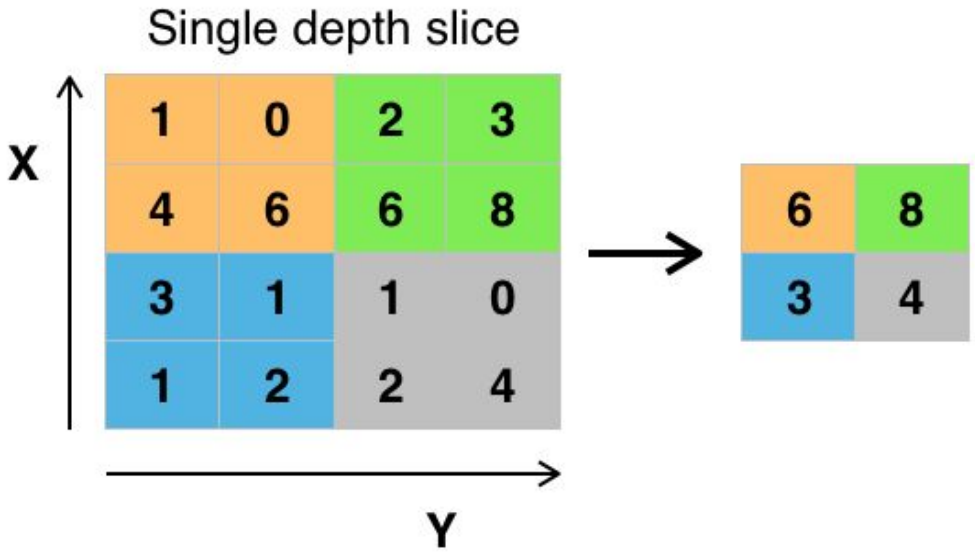

Fig 3.2.3: Pooling operation

Fully-connected layer : Neurons in a fully connected layer have full 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.

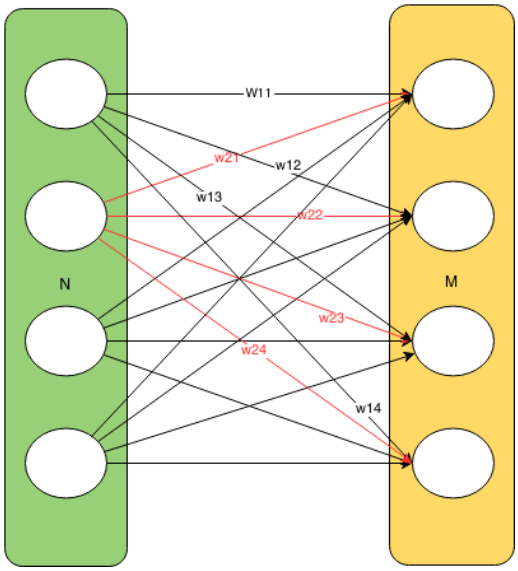

Fig 3.2.4: Fully connected layer

The deep convolutional neural network (CNN) architecture proposed in this study is comprised of 3 convolutional layers interleaved with 2 pooling operations, followed by 2 fully connected (dense) layers. The proposed CNN architecture is parameterized as follows: (Salamon and Bello, 2017)

1. 24 filters with a receptive field of (5,5), i.e., W has the shape (24,1,5,5). This is followed by (4,2) strided max pooling over the last two dimensions (time and frequency respectively) and a rectified linear unit (ReLU) activation function h(x) = max(x, 0).
2. 48 filters with a receptive field of (5,5), i.e., W has the shape (48, 24, 5, 5). Like `1, this is followed by (4,2) strided max-pooling and a ReLU activation function.
3. 48 filters with a receptive field of (5,5), i.e., W has the shape (48, 48, 5, 5). This is followed by a ReLU activation function (no pooling).
4. 64 hidden units, i.e., W has the shape (2400, 64), followed by a ReLU activation function.
5. 10 output units, i.e., W has the shape (64,10), followed by a softmax activation function.

Summary of our CNN

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 124, 83, 24)	624
max_pooling2d_1 (MaxPooling2)	(None, 31, 41, 24)	0
activation_1 (Activation)	(None, 31, 41, 24)	0
conv2d_2 (Conv2D)	(None, 27, 37, 48)	28848
max_pooling2d_2 (MaxPooling2)	(None, 6, 18, 48)	0
activation_2 (Activation)	(None, 6, 18, 48)	0
conv2d_3 (Conv2D)	(None, 2, 14, 48)	57648
activation_3 (Activation)	(None, 2, 14, 48)	0
flatten_1 (Flatten)	(None, 1344)	0
dropout_1 (Dropout)	(None, 1344)	0
dense_1 (Dense)	(None, 64)	86080
activation_4 (Activation)	(None, 64)	0
dropout_2 (Dropout)	(None, 64)	0
dense_2 (Dense)	(None, 10)	650
activation_5 (Activation)	(None, 10)	0

Total params: 173,850
Trainable params: 173,850
Non-trainable params: 0

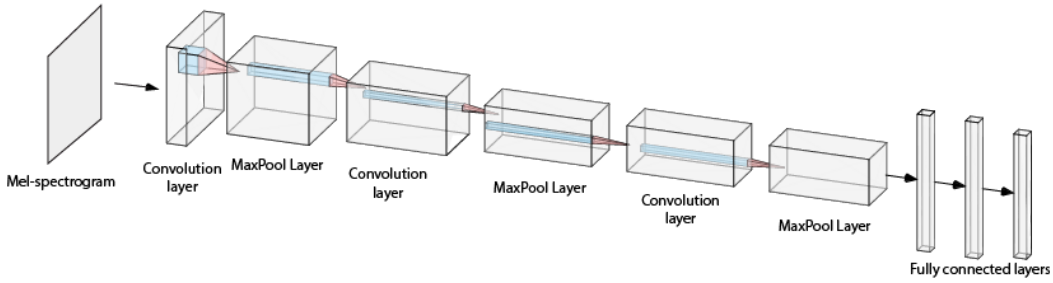

Fig 3.2.5: CNN Architecture

E. Other Operations performed in our CNN

ReLU Activation function

The Rectified Linear Unit is the most commonly used activation function in deep learning models. The function returns 0 if it receives any negative input, but for any positive value x it returns that value back. So it can be written as $f(x) = \max(0,x)$.

Graphically it looks like this

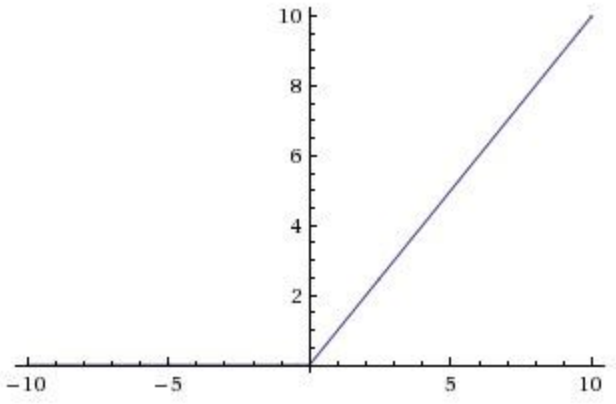

Fig 3.2.6 ReLU function

It's surprising that such a simple function (and one composed of two linear pieces) can allow your model to account for non-linearities and interactions so well. But the ReLU function works great in most applications, and it is very widely used as a result.

Dropout

Dropout is a technique where randomly selected neurons are ignored during training. They are "dropped-out" randomly. This means that their contribution to the activation of downstream neurons is temporarily removed on the forward pass and any weight updates are not applied to the neuron on the backward pass.

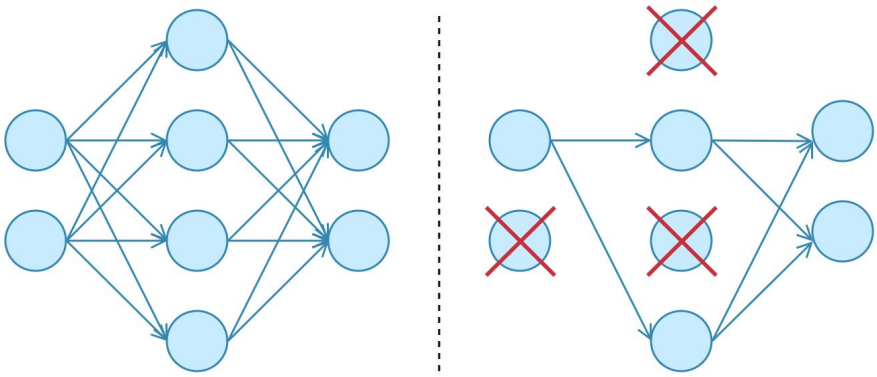

Fig 3.2.7: Dropout layer

Flattening

Flattening is the process of converting all the resultant 2 dimensional arrays into a single long continuous linear vector. The flattening step is needed so that you can make use of fully connected layers after some convolutional layers.

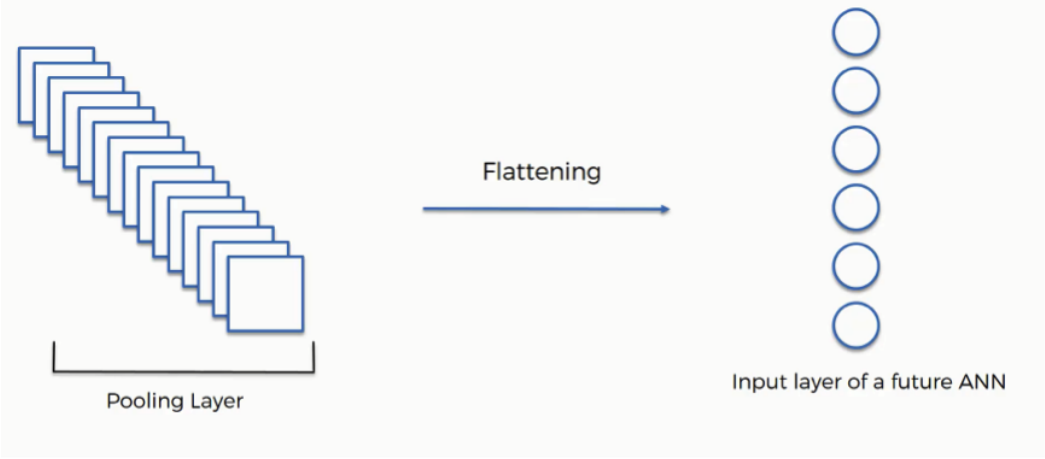

Fig 3.2.8: Flattening operation

Softmax Activation

Softmax it's a function, not a loss. It squashes a vector in the range (0, 1) and all the resulting elements add up to 1. It is applied to the output scores s. As elements represent a class, they can be interpreted as class probabilities.

The Softmax function cannot be applied independently to each si, since it depends on all elements of s. For a given class si, the Softmax function can be computed as:

$$f(s)_i = \frac{e^{s_i}}{\sum_j^C e^{s_j}}$$

Where sj are the scores inferred by the net for each class in C. Note that the Softmax activations for a class si depends on all the scores in s.

F. Training Process

Optimizer- Adam Optimizer

Kingma, D., & Ba, J. (2019). The training was performed using Adam Optimizer. We introduce Adam, an algorithm for first-order gradient-based optimization of stochastic objective functions, based on adaptive estimates of lower-order moments. The method is straightforward to implement, is computationally efficient, has little memory requirements, is invariant to diagonal rescaling of the gradients, and is well suited for problems that are large in terms of data and/or parameters. The method is also appropriate for non-stationary objectives and problems with very noisy and/or sparse gradients. The hyper-parameters have intuitive interpretations and typically require little tuning. Empirical results demonstrate that Adam works well in practice and compares favorably to other stochastic optimization methods[11].

Batch size and epoch

Epochs

One Epoch is when an entire dataset is passed forward and backward through the neural network only once. Since, we are using a limited dataset and to optimise the learning and the graph we are using Gradient Descent which is an iterative process. So, updating the weights with single pass or one epoch is not enough. As the number of epochs increases, more number of times the weight are changed in the neural network and the curve goes from underfitting to optimal to overfitting curve.

Batch

Since, one epoch is too big to feed to the computer at once we divide it in several smaller batches. Batch Size is the total number of training examples present in a single batch.

For training our dataset, we divided 500 training examples into batches of 20 (i.e. batch_size=20) and epoch of 70 times.

G. Data Augmentation for better performance

We experiment with 3 different audio data augmentations (deformations), resulting in 3 augmentation sets, as detailed below. Each deformation is applied directly to the audio signal prior to converting it into the input representation used to train the network (mel-spectrogram). The deformations and resulting augmentation sets are described below:

• Time Stretching (TS): slow down or speed up the audio sample (while keeping the pitch unchanged). Each sample was time stretched by 2 factors: {0.81, 1.07}.
Hence, 4000 augmented audio files were created.

• Pitch Shifting (PS1): raise or lower the pitch of the audio sample (while keeping the duration unchanged). Each sample was pitch shifted by 2 semitones:
Hence, 2000 augmented audio files were created.

• Pitch Shifting (PS2): since our initial experiments indicated that pitch shifting was a particularly beneficial augmentation, we decided to create a second augmentation set. This time each sample was pitch shifted by larger value i.e. 2.5 semitones:

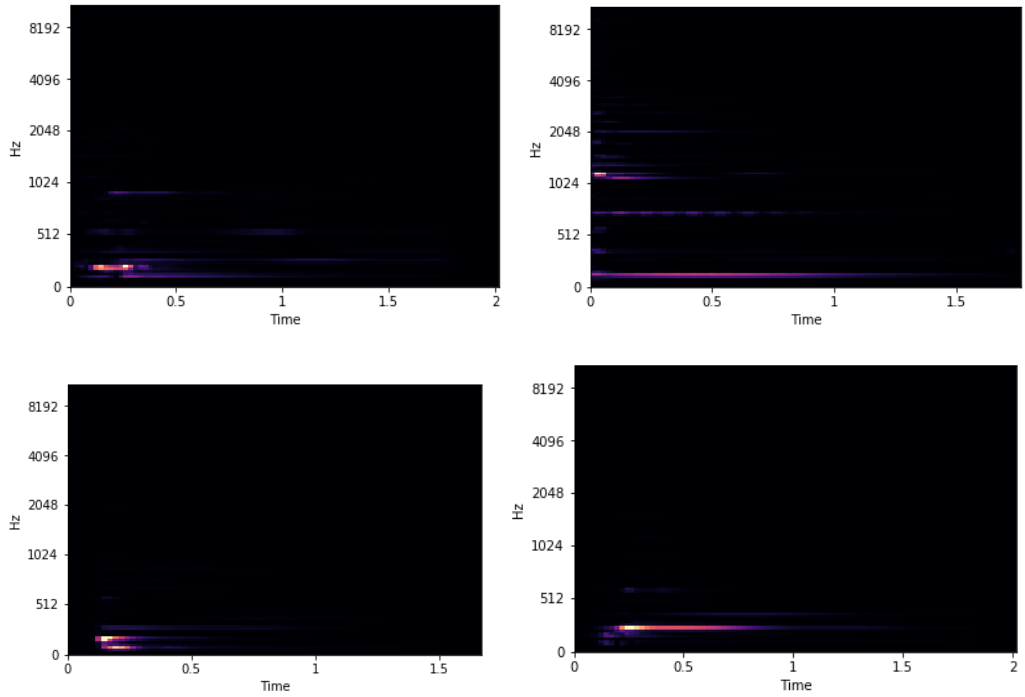

Fig 3.2.9: Mel Spectrogram Chroma features of augmented audio files /ps1_2/em50.wav, /speed_81/d200.wav, /speed_107/g75.wav and /ps2m_25/am25.wav respectively from guitar chord dataset.

We again used librosa library to extract chromagram from the augmented audio datasets. We built a new dataset consisting of mel spectrograms of the chroma features of all the audio files as well as of the new augmented audio files in the guitar chord datasets as input and corresponding class_id as output. Since the audio files in our dataset are of varying duration (up to 4 s), we fixed the size of the input to 2 seconds (128 frames), i.e. $X \in \mathbf{R}^{128 \times 87}$

We obtained 3860 mel spectrogram chroma features having equal number of frame size from 10000 audio files (2000 original and 8000 augmented).

The dataset consists of:

- 1-633 normal samples.
- 634-981 samples speed up by 1.07.
- 982-2594 samples Slowed down to 0.81.
- 2595-3227 samples Pitch modulated 2 semitones higher.
- 3228-3860 samples Pitch modulated 2.5 semitones higher.

The new augmented dataset was then shuffled and divided to training and test datasets as:

Training data: 1-3000
Test data: 3001-3860

Then the dataset was further preprocessed, fed to the same convolution neural network, trained and performance metrics were evaluated similar to the original dataset previously.

H. User interface for prediction

A simple user interface was designed to predict a new chord using the trained model. The user interface consists of a record button which records sound for 3 seconds and stores it as recorded.wav. The classify button predicts the chord of the recorded sound using the trained model. The play button is used to play the recorded sound.

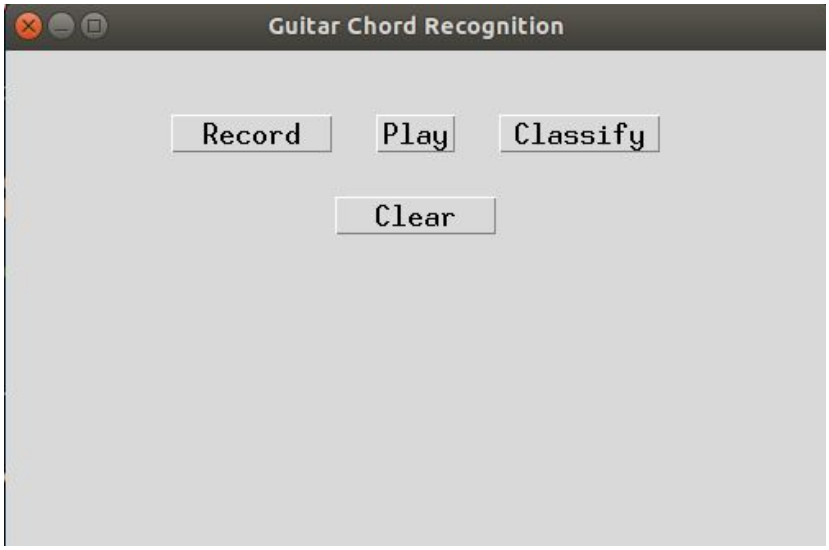

Fig 3.2.10: User interface for guitar chord recognition

4 Results and Evaluation

Performance Metrics used

1. Confusion Matrix

Confusion Matrix as the name suggests gives us a matrix as output and describes the complete performance of the model.

There are 4 important terms :

True Positives : The cases in which we predicted YES and the actual output was also YES.
True Negatives : The cases in which we predicted NO and the actual output was NO.
False Positives : The cases in which we predicted YES and the actual output was NO.
False Negatives : The cases in which we predicted NO and the actual output was YES.

		Predicted class	
		P	*N*
Actual Class	*P*	True Positives (TP)	False Negatives (FN)
	N	False Positives (FP)	True Negatives (TN)

Before data augmentation

The following confusion matrix was obtained before data augmentation on 133 test data after training on 500 training data.

		Predicted class									
		A	**Am**	**Bm**	**C**	**D**	**Dm**	**E**	**Em**	**F**	**G**
Actual class	**A**	**3**	3	0	0	1	0	0	0	0	0
	Am	0	**15**	0	1	0	0	0	0	0	0
	Bm	0	0	**0**	0	0	0	0	0	0	0
	C	0	0	0	**20**	0	0	0	0	1	1
	D	0	0	0	0	**17**	4	0	0	0	0
	Dm	0	0	0	0	5	**4**	0	0	0	0
	E	1	0	0	1	0	0	**15**	2	2	0
	Em	0	1	0	0	0	1	0	**8**	0	1
	F	0	0	0	0	0	0	0	0	**7**	0
	G	0	1	0	0	0	0	0	0	0	**18**

.

After data augmentation

The following confusion matrix was obtained after data augmentation on 867 test data after training on 3000 training data.

		Predicted class									
		A	**Am**	**Bm**	**C**	**D**	**Dm**	**E**	**Em**	**F**	**G**
Actual class	**A**	**40**	3	2	0	1	0	0	0	0	1
	Am	2	**104**	5	0	0	0	0	0	2	1
	Bm	1	1	**22**	0	0	0	0	1	0	0
	C	0	0	0	**122**	3	0	0	0	1	0
	D	0	0	0	0	**97**	12	1	1	0	2
	Dm	0	1	1	1	10	**51**	2	1	1	0
	E	0	0	1	0	1	0	**110**	1	0	0
	Em	0	0	0	0	1	0	8	**65**	0	0
	F	0	0	0	0	0	0	4	1	**49**	8
	G	3	2	1	0	0	0	0	1	1	**108**

2. Classification Accuracy

Classification Accuracy is what we usually mean, when we use the term accuracy. It is the ratio of number of correct predictions to the total number of input samples.

$$Accuracy = \frac{Number\ of\ Correct\ predictions}{Total\ number\ of\ predictions\ made}$$

It works well only if there are equal number of samples belonging to each class.

For example, consider that there are 98% samples of class A and 2% samples of class B in our training set. Then our model can easily get 98% training accuracy by simply predicting every training sample belonging to class A.

When the same model is tested on a test set with 60% samples of class A and 40% samples of class B, then the test accuracy would drop down to 60%. Classification Accuracy is great, but gives us the false sense of achieving high accuracy.

Before data augmentation:

The following classification accuracy was obtained before data augmentation on 133 test data after training on 500 training data.

Training accuracy: 0.8520
Test accuracy: 0.8045

After data augmentation:

The following confusion matrix was obtained after data augmentation on 867 test data after training on 3000 training data.

Training accuracy: 0.9227
Test accuracy: 0.8904

3. Precision

It is the number of correct positive results divided by the number of positive results predicted by the classifier.

$$Precision = \frac{TruePositives}{TruePositives + FalsePositives}$$

Before data augmentation:

The following precision was obtained before data augmentation on 133 test data after training on 500 training data.

Training precision: 0.8688
Test precision: 0.8297

After data augmentation:

The following precision was obtained after data augmentation on 867 test data after training on 3000 training data.

Training precision: 0.9355
Test precision: 0.9035

4. Recall

It is the number of correct positive results divided by the number of all relevant samples (all samples that should have been identified as positive).

$$\text{Recall} = \frac{True\ Positive}{True\ Positive + False\ Negative}$$

Before data augmentation:

The following recall was obtained before data augmentation on 133 test data after training on 500 training data.

Training recall: 0.8300
Test recall: 0.8045

After data augmentation:

The following recall was obtained after data augmentation on 867 test data after training on 3000 training data.

Training recall: 0.9037
Test recall: 0.8846597462514417

5. F1 Score

F1 Score is used to measure a test's accuracy. It is the Harmonic Mean between precision and recall. The range for F1 Score is [0, 1]. It tells you how precise your classifier is (how many instances it classifies correctly), as well as how robust it is (it does not miss a significant number of instances).

High precision but lower recall, gives you an extremely accurate, but it then misses a large number of instances that are difficult to classify. The greater the F1 Score, the better is the performance of our model. Mathematically, it can be expressed as :

$$F1 = 2 * \frac{1}{\frac{1}{precision} + \frac{1}{recall}}$$

F1 Score tries to find the balance between precision and recall.

Before data augmentation:

The following F1-score was obtained before data augmentation on 133 test data after training on 500 training data.

Training F1-score: 0.8481
Test F1-score: 0.8169

After data augmentation:

The following F1-score was obtained after data augmentation on 867 test data after training on 3000 training data.

Training F1-score: 0.9189

Test F1-score: 0.8938

Comparison of performance metrics before and after augmentation

	Before Augmentation (test)	After Augmentation (test)	Before Augmentation (training)	After Augmentation (training)
Accuracy	0.8045	0.8904	0.8520	0.9227
Precision	0.8297	0.9035	0.8688	0.9355
Recall	0.8045	0.8846	0.8300	0.9037
F1-score	0.8169	0.8938	0.8481	0.9189

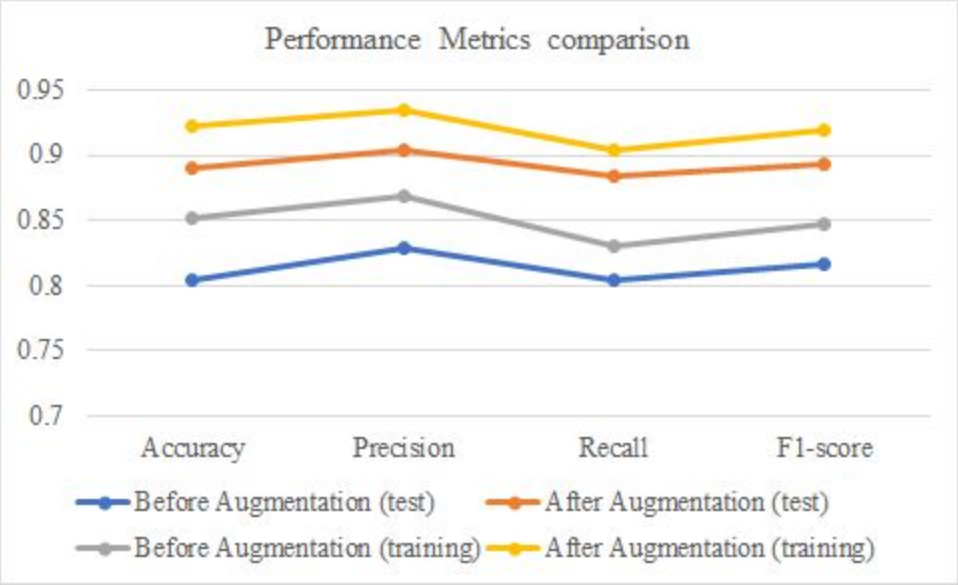

Fig 4.1: Performance metrics comparison

7 Conclusion

In this article we proposed a deep convolutional neural network architecture in which original chords dataset in combination with a set of audio data augmentations produces the desired prediction for the guitar chords classifier. We compared the results and performance metrics before and after data augmentation and came to a conclusion that data augmentation helped to make the classifier more accurate.

References:

1. Bzamenik, “chord-recognition.” https://github.com/bzamecnik/ml/tree/master/chord-recognition. 2017. Accessed :2019-01-03.

2. J. Osmalskyj, J-J. Embrechts, S. Piérard, M. Van Droogenbroeck, “Neural Networks for Musical Chord Recognition.” http://jim.afim-asso.org/jim12/pdf/jim2012_08_p_osmalskyj.pdf, 2012 Accessed : 2019-01-10.

3. K. Lee & M. Slaney, “Automatic Chord Recognition from Audio Using an HMM with Supervised Learning ” https://ccrma.stanford.edu/~kglee/pubs/klee-ismir06.pdf, Accessed : 2019-01-12.

4. Ajhalthor, “audio-classifier-convNet. ” https://github.com/ajhalthor/audio-classifier-convNet, Accessed : 2019-01-09.

5. CodeEmporium, “Sound play with Convolution Neural Networks. “ https://www.youtube.com/watch?v=GNza2ncnMfA, 2018 Accessed : 2019-01-09.

6. Salamon, J., & Bello, J. P. , “Deep Convolutional Neural Networks and Data Augmentation for Environmental Sound Classification”. *I*

7. UrbanSound8K, “ UrbanSunk8K dataset” https://urbansounddataset.weebly.com/urbansound8k.html

8. Liège, U. D. (n.d.).. “Research and education in sound and image techniques. ” http://www.montefiore.ulg.ac.be/services/acous/STSI/downloads.php

9. En.wikipedia.org, “Chroma feature. ” https://en.wikipedia.org/wiki/Chroma_feature Accessed 2019-01-10

10. Osmalskyj, Julien & Embrechts, Jean J. & Droogenbroeck, Marc & Piérard, Sébastien., “Neural networks for musical chords recognition.” , 2012

11. Kingma, D., & Ba, J. (2019). Adam: A Method for Stochastic Optimization. Retrieved from https://arxiv.org/abs/1412.6980v8