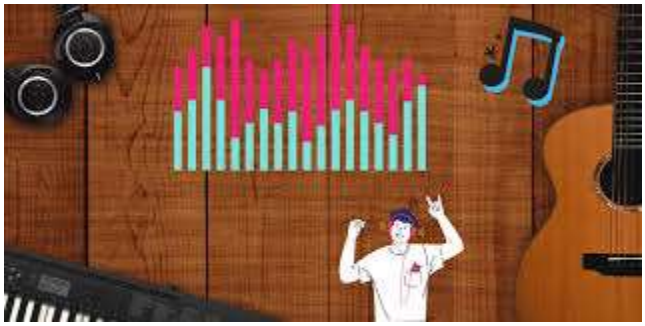
Music Genre Classification System



Abstract

Different kinds of music have different impacts on our moods. Many times, we wish to listen to a specific kind of music. It may be a problem to search for a perfect music genre according to our mood from many songs. This paper documents a machine-learning model that aims to solve this widespread problem. We first converted our dataset which consisted of Audio files into their respective spectrograms for further classification. Later, we used a convolutional neural network with three hidden layers to train our model to classify music into genres. The experimental results show that the use of an Ensembler in our model gives superior performance. In the end, we can achieve a model that could classify music into genres.

Problem statement

Our problem statement was to make a machine-learning model that could classify Music into genres. Our model will take any music as input and then tell which genre it belongs to.

Tech Stack

Python, Tensorflow Keras, numpy and Librosa

Literature review

We started our project by making a basic CNN model which had three layers with 32 neurons each. Initially we were using Kernels of size 2*2 each which we changed to 3*3 later. We emphasized on increasing the number of neurons in our model to make it denser. We added Batch Normalization in first and second layer with Dropout in third layer. Initially we were using both of them together but later we shifted to using them one by one. We then moved to adding regularization to our model. We referred to L1 and L2. We found that L1 is not suitable for our model as

our data is close to 1 or less than 1. We used Weight Decay. We also used Early stopping to prevent overfitting of our model. We finalized our model by adding an Ensembler to it. Our Ensembler uses soft voting.

Methodology

Our Problem could be solved by using two algorithms: KNN and CNN.So we first tried to understand both approaches and looked at the advantages and disadvantages of using both methods. So what we understood is that In KNN, output completely relies on nearest neighbors, which may or may not be a good choice. Also, it is sensitive to distance metrics. So we chose to use CNN as CNNs store much more information as parameters than other methods and CNNs build their own features from raw signal. These features make the use of CNN more appropriate.

Operations carried Out:

1. **Getting the Datset**: We have used the GTZAN dataset consisting of 10 genres x 100 audio files loaded using the Librosa library.
2. **Data preparation**: Signal Processing Via MFCC Feature extraction and STFT.
3. **Training the model**: We will be using two models: CNN and Ensembler using CNN.
4. **Evaluating the model**: Compare accuracies, identify problems, and test on Random data
5. **Improving the model**: Cases of Overfitting, Underfitting, Accuracy, and Generalisation are solved here
6. **Latency, Recommendation and Deployment:**
   Pruning, Quantization is carried out along with making a recommendation block.

Audio Pre-processing

To start with, we first load Audio files through librosa and get the signal (numpy array of amplitude +1 to -1) and sample rate (sample rate is the number of samples per second that are taken of the waveform to create discrete digital signal)

- **Fourier transforms (FT)** -Decompose complex periodic sound waves as a sum of sine waves oscillating at different frequencies. Fourier Transform converts Time Domain Dependency Waveform (Amplitude vs. Time) into Frequency Domain Dependency (Amplitude vs Frequency).

- **MFCC Features**: A signal's Mel frequency cepstral coefficients are a small set of features (usually 10-20) that concisely describe the overall shape of the spectral envelope.
- **Short Time Fourier transform(STFT)-**computes several FFT at different time intervals. So it preserves time information STFT produces a spectrogram (time +frequency+magnitude).

So by performing STFT/MFCC we extract the audio features from the audio wav files and produce a spectrogram which is given as an input to our model.

Description of the model:

There are a total of 3 hidden layers in the model. They all consist of 32 channels of (3,3) filters and (1,1) stride with regularization parameter =0.01 and ReLu activation function. We have a max pooling layer with (3,3) filter and (2,2) stride for each layer. Also to reduce overfitting we have applied Batch Normalization to all the layers. Then we flatten our data and then pass through a dense layer. Then finally we have our output layer(Probabilistic layer) which uses the Softmax activation function.

:

Overfitting:

We see that the model performs well on the training data but does not perform that well on the evaluation data. This is because the model memorizes the data it has seen and is unable to generalize to unseen examples. Overfitting refers to an unwanted behaviour of a machine learning algorithm used for predictive modelling. Overfitting happens due to several reasons, such as The training data size being too small and does not contain enough data samples to accurately represent all possible input data values.

So we used various techniques in order to overcome it :

1. **Weight Decay** - We used the L2 regularization technique to reduce our kernel values. It adds up a square of kernel values in the loss functions
2. **Dropout-** this drops out some of the neurons by setting their activations to zero with a given probability, preventing it from relying on one feature

3. **Early Stopping** -If the performance on the validation dataset does not improve over a certain number of epochs, the training process is stopped
4. **Batch Normalization** -This Normalizes the Activations in each Channel using its Mean and Variance in each mini-batch. This can Regularize the Model.

ENSEMBLE-LEARNING:

The ensemble learning model is a collection of small models. It is mainly used to **increase accuracy** and obtain a **low-bias low variance** model. The disadvantage of using this is that it increases **Computational complexity.**

There are primarily two ways in which we can do ensemble learning::

1. By having different algorithms for different models.
2. By having the same algorithm for all models but trained on different datasets.

How does ensemble learning work for classification?

Suppose there are 50 models and each model predicts something(0/1). Suppose 40 models predict 1 and 10 models predict 0 then the final output will be 1.

How does ensemble learning work for regression?

Similarly, in this case, each model will predict a particular value. Then the final output will be the average or maximum of all the predicted values.

There are 4 different types of ensemble learning :

1.Voting - Base models have different algorithms or the same algorithm with different hyperparameters trained with the same dataset. In voting, all the base models have equal weightage.

2.Stacking-In this case in addition to what is done in voting another model is trained by giving inputs (which are outputs of all models) to decide the weights of these base models in providing the final output.

3.Bagging-Base models are trained with the same algorithm but on different datasets. A special case of this is a Random forest in which the base model algorithm is a decision tree. In bagging, weak learners are trained in parallel.

4.Boosting-In this case one model is first trained on the dataset and then the wrongly predicted data is passed for training to the next model with the same algorithm and this is done multiple number of times. So with each new model iteration, the weights of the misclassified data in the previous model are increased. This redistribution of weights helps the algorithm identify the parameters that it needs to focus on improving its performance.

In our model, we have implemented **Soft Voting** Ensembling in which all the models are trained with the same algorithm but different hyperparameters in 2 folds for Cross Validation, So In total we are training **2x24=48 base** models.

We used the following sequence to establish an ensembler:

1. **Creating a function:** We created a function to generate a Model by instruction for Primary Model.
2. **Model Instance:** Creating an instance of the CNN skeleton model.
3. **Parameter Grid**: Defining a Parameter Grid for Selecting different parameters to choose from for the ensemble. The one which gives maximum accuracy will be chosen.
4. **Voting Classifier Instance**: Creating a Voting Classifier instance by inputting default values.
5. **Grid CV Search:** Applying Cross Validation (any folds) as well as Searching through the Grid for the best suitable combination.
6. **Fitting:** Fitting data in the Model that tends to give the best results.

Experiments

Signal Processing:

Tried FT, but we learned that STFT is a better as it performs windowed FT and we can go back and forth , hence recreate sound functions from features

Model :

- Used Learning Parameters =0.001,0.01,0.1 and Used Different Optimizers like Adam And stochastic gradient descent. While Adam is most commonly used, Stochastic Gradient Descent is Relatively faster as the learning rate keeps changing.

- We used different numbers of neuron layers which increased the Complexity of the model . We kept working in the range of 2-4.
- We experimented with Basic ANN first, later on as we worked we started working on CNN's complexity by working on Pooling techniques and Padding in different ways.

Improvement Techniques:

- We experimented with our model using different improvement techniques. We used Batch Normalization and Regularization to fix the problem of overfitting. We also used Early stopping method.
- Later on, we added an ensembler to increase the accuracy. We ran our ensembler to pick up most suitable optimizer, drop-out rate, learning rate, number of neurons and layers for our model.

GPU Usage:

Since the Training Time was much higher due to complexity. It took almost 1.7 hours to train the Ensembler model and it took 21 minutes to train the base model. To reduce this , we researched a bit on the usage of GPU memory for the training of Ensemble model. We used Cuda toolkit to Train these TensorFlow Models

Results:

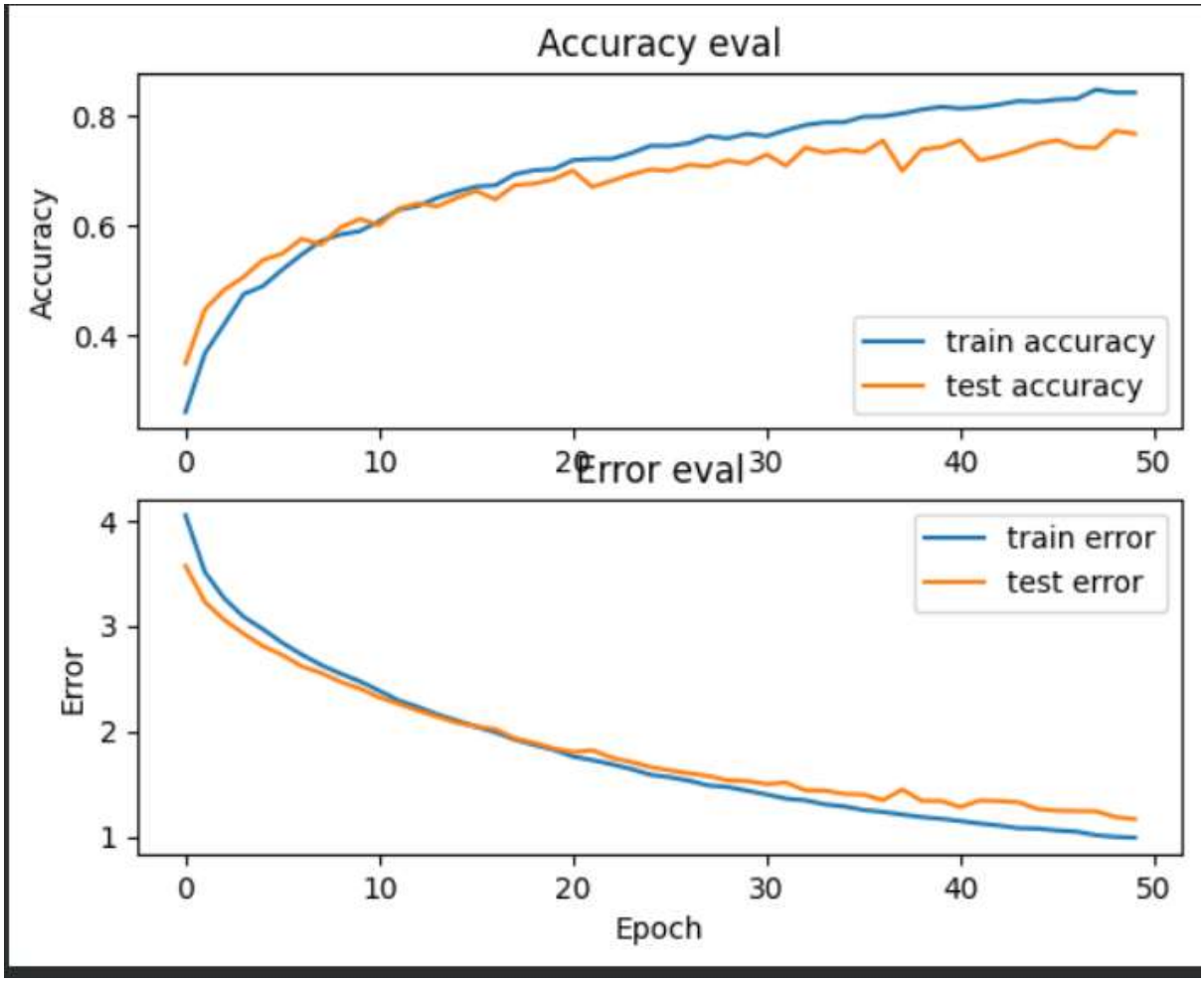
Validation accuracy :

Initially when we used only CNN model - validation accuracy was 96% which was clearly signifying overfitting. Hence we fixed overfitting by using batch normalization, dropout and weight decay and reached an accuracy of 93%. Then we implemented ensemble learning by using Voting Classifier and GridCVSearch which increased the accuracy to 94%.

Test Accuracy :

Initially when we used only the CNN model- the test accuracy was 70%. After using batch normalization, dropout and weight decay, we reached an accuracy of 77%. Then we implemented ensemble learning by using Voting Classifier and GridCVSearch which increased the accuracy to 80%.

Inference time: We are able to make a prediction in 30 milliseconds or less.



Learning Takeaways

In our course throughout this project, we got an idea of how we can handle large datasets efficiently. We used the GTZan dataset for the training of our model. We divided our dataset into various sets to use them separately for training and testing.

We used CNN algorithm with multiple hidden layers. We also got familiar with the use of the TensorFlow library. We also considered using KNN but we found that CNN has an edge over it hence we shifted to its use.

Through this project, we got to learn about many new techniques that can be used to make a Machine learning model more efficient in terms of latency and accuracy.

We learnt about the use of Ensembler which is a technique to improve the accuracy of the model. We tried using different kinds of ensemblers in our model. We used them over hyperparameters such as dropout rate, learning rate, number of neurons, number of layers and optimizers(Adam and sgd). The usage of ensembler gave a large boost to our accuracy.

We explored techniques to reduce latency in ML models such as Model Pruning, Knowledge distillation, and Quantization.
While working on our model, we also used measures to keep overfitting in check. On the go, we got an idea about what overfitting actually is and how it can be reduced. We learned about different types of Regularization methods and we ended up using weight decay which is an L2 regression regularization technique. We also used batch normalization, dropout and early stopping to keep track of overfitting. We got to learn about these techniques and their use.

References

For training model:
Deep Learning (for Audio) with Python - YouTube
GTZAN Dataset - Music Genre Classification using Python

For learning about ensemble:
Voting Ensemble | Introduction and Core Idea | Part 1

For learning about latency-reducing methods:
Knowledge Distillation Explained with Keras Example | #MLConcepts
Inside TensorFlow: TF Model Optimization Toolkit (Quantization and Pruning)
Post-training quantization | TensorFlow Lite

Contributions made by:-

Pranjal Vanjale, Siddhant Gupta, Vidhi Gupta and Yashovardhan Pandey. (Students of IIT Roorkee)

We are grateful to Ashwarya Rao Maratha and Shreshth Mehrotra for their guidance.