MACHINE LEARNING COURSE PROJECT: MUSIC GENRE CLASSIFICATION

Team members:

Phạm Việt Thành - 20194454 - thanh.pv194454@sis.hust.edu.vn
Nguyễn Tuấn Dũng - 20194427 - dung.nt194427@sis.hust.edu.vn
Đặng Vũ Hoàng Hiệp - 29194431 - hiep.dvh194431@sis.hust.edu.vn

I. Introduction

Music genre classification - the task of identifying the genre of a record, has appeared in many different applications nowadays. In the past ten years, the field of music genre classification has achieved exceptional performance and can be applied in real world situations. Leading companies use music genre classification either to create robust music recommender systems (Spotify, Youtube) or simply to make an application out of it (Shazam). This excellent result comes with the development of powerful machine learning and deep learning algorithms. In this report, we will discuss the architectures of 3 popular machine learning models, along with the performance of the three on a particular dataset.

1. Problem description

Our task will be classification of 10 popular music genres: blues, classical, country, disco, hiphop, jazz, metal, pop, reggae and rock.

2. Machine learning algorithms

- Support Vector Machines (SVM)
- Dense Neural Network (DNN)
- Convolutional Neural Network (CNN)

3. Input & Output

- Input: learning examples in the dataset are .wav files, which are records of different songs.
- Output: label for an example, in this case is the music genre of the record.

4. Representation of learning examples

We consider different options of features representations of learning examples, which we will extract directly from the .wav files:

- Using ***Mel spectrogram*** extracted from the .wav files as input features for CNN since spectrogram behaves similarly to image tensor.
- Using ***MFCC*** extracted from the .wav files as input features for DNN and SVM.

5. Dataset

GTZAN Genre Collection. This dataset contains 1000 tracks, each track is 30 seconds long. There are 10 genres, each has 100 tracks.

II. Dataset Introspection & Features Extraction

1. Dataset introspection

GTZAN dataset consists of 1000 tracks of 10 genres, each track is 30 seconds long. There are 10 different tracks for each music genre. The dataset was not intentionally

created for music genre classification, but its availability has made it a benchmark dataset, which is used for assessing the performance of music genre classification systems. However, the dataset was published in 2002, so it contains music tracks of the previous generation. Thus, systems performing well on the dataset may not achieve similar results on recent music tracks.

As the original dataset does not contain the metadata for each record, we have gathered information about the artists and how they contributed to the dataset, according to several researches.

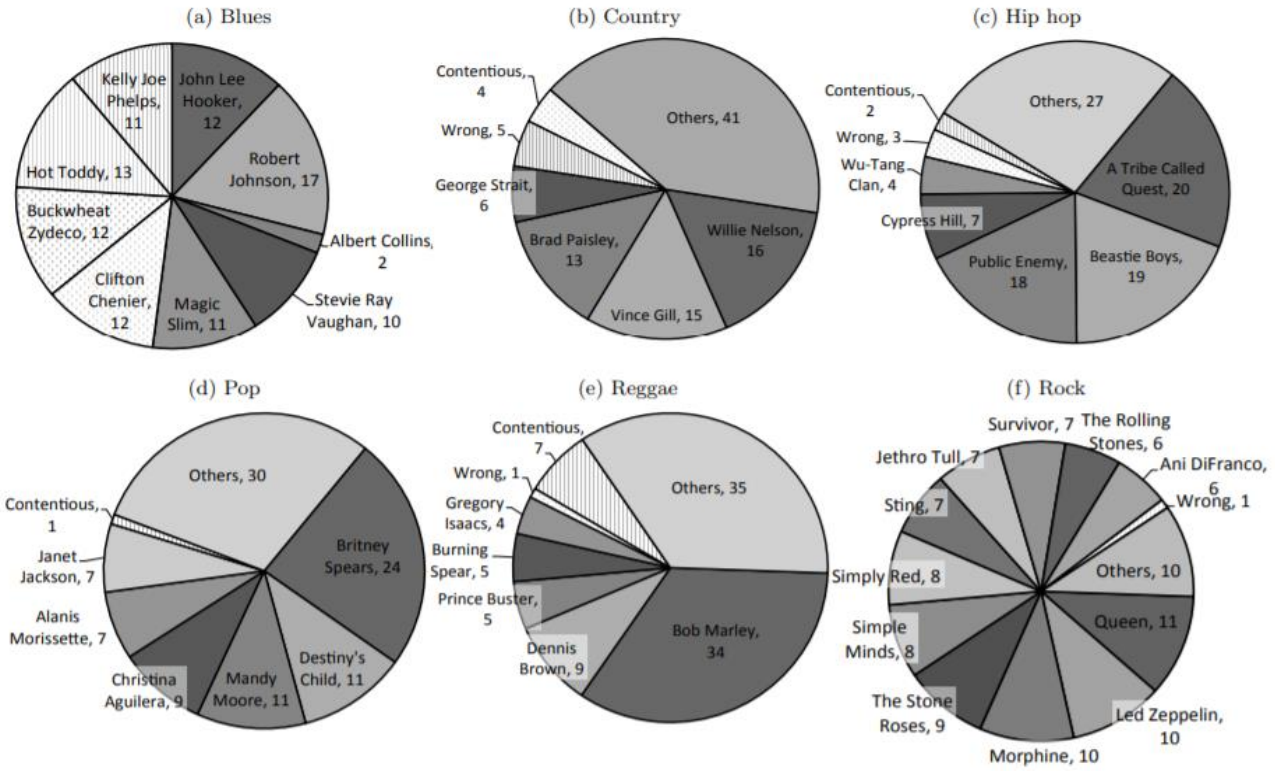


Figure 1. Artists information of GTZAN dataset

In some genres, some particular artists contributed much more than the others, such as Bob Marley having 34 records in the Reggae genre, which outclassed other artists in terms of quantity. This proves that the data is not balanced, although the numbers of samples for each genre are the same. Furthermore, there are some faults in the design of the dataset: 7.2% of the files come from the same track (in which 5% duplicated exactly), and 10.6% of the data is mislabeled.

2. Features extraction

Generally, feature extraction is the most important part in every task related to signal processing, including music genre classification. Having chosen the right set of features, even models with simple architectures can perform really well.

a. Mel scale

The two sets of features described below heavily depend on the concept of Mel scale frequency.

Mel scale is a logarithmic transformation applied to a signal's frequency. An audio signal is a combination of multiple sound waves of different frequencies, which are calculated in Hertz (Hz). However, humans do not perceive sound as linear. That is, most people can easily distinguish between 200 Hz and 300 Hz sound but struggle to tell the difference between 900 Hz and 1000 Hz sound, although the frequency distances are the same (100Hz). Fitting features of Hertz scale frequency into models seems irrelevant, as DNN and CNN are well designed to replicate how neurons in the human brain work. This is where Mel scale transformation comes in. The transformation of f Hz to m mels is the following:

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

b. Mel frequency cepstral coefficients (MFCC)

The sound generated by humans is determined by the shape of the vocal tract. If we can find a way to represent these shapes, we can build a robust system to distinguish any sound. Mel frequency cepstral coefficients (MFCC) is a set of features which accurately represents these shapes of vocal tract. Figure 2 illustrates the process of calculating MFCC vectors from raw audio waves.

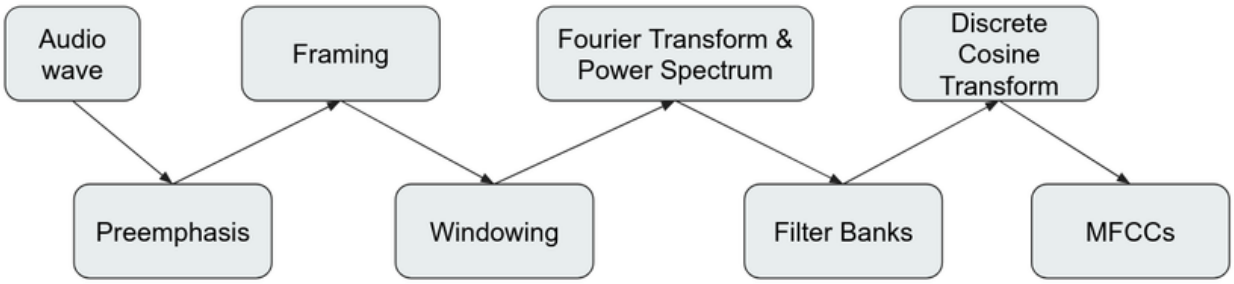


Figure 2. MFCC Calculation

We will not explain the detailed calculation of the process as it is out of scope for the project. The output of the process is a vector of 40 dimensions for each sample. Commonly, we will use the first 13-20 coefficients although latter coefficients are also informative. Taking all coefficients results in higher complexity of the model and thus need more data to train.

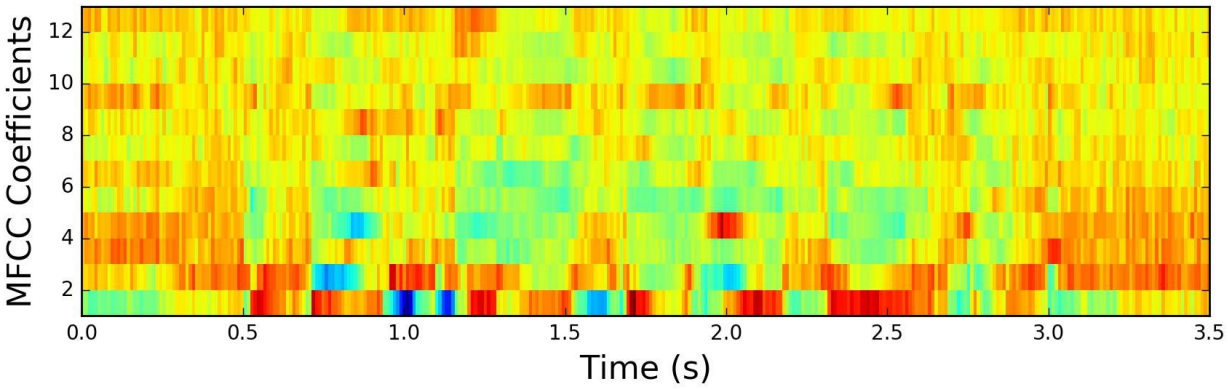


Figure 3. MFCC of an audio of 3.5 seconds long

Since audio is sequential data, we usually extract MFCC for every small time step, as shown in Figure 3, then feed it into sequential models such as RNN and LSTM. In the case of DNN and SVM, we cannot just feed a sequence of MFCC into the model, as the computational cost will be much higher. Therefore, in our strategy, we take the first 20 MFCC coefficients, compute the mean and variance by time steps of each coefficient for each audio sample, meaning that there will be 40 features for DNN and SVM. Mean and variance accurately capture the changes in the values of coefficients, which is a great alternative to sequential MFCC.

However, we find that if we just extract these 40 dimensional vectors directly from the audio samples in the dataset, the models will not be able to classify genres very well. Each audio file is 30 seconds long and thus there will be a large number of time steps. Extracting features from these files will result in information loss. Hence we split every audio file into 9 subfiles, each is about 3 to 4 seconds long, then extract features from them. Doing this also increases the number of samples in the datasets, which will help the models generalize better.

c. Mel spectrogram

MFCC has proven its popularity in the last decades, it is widely used in both traditional models (GMM, HMM) and deep learning models. With the development of complex model architectures, especially convolutional neural networks, Mel spectrogram is becoming more and more common in audio classification tasks, when using CNN models such as VGG and ResNet. The idea is that CNNs have the ability to extract different features from input tensors, so we expect them to learn some patterns in different music genres. After extracting the spectrogram, we have observed different spectral patterns in the genres, which we hope the model can identify correctly. In most recent research, people often use highly complex models, namely ResNet50 and VGG19, to do several tasks such as noise detection, speaker identification. In this project, due to the lack of computational resources, we will train from scratch a simple CNN model to assess its performance.

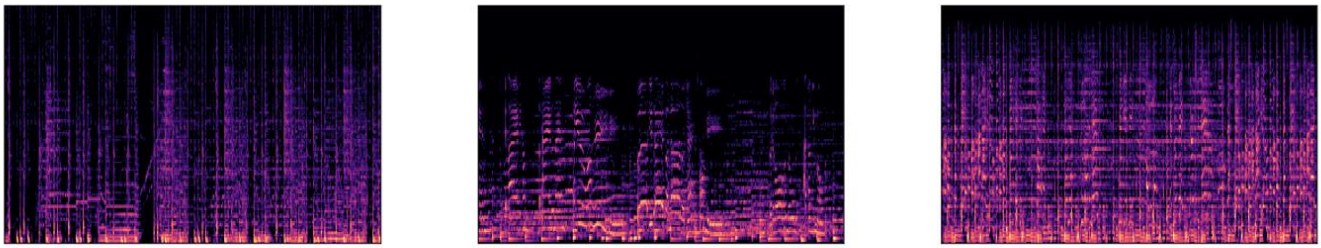


Figure 4. Mel spectrogram plot of 3 different genres

III. Machine Learning Algorithms

1. Support Vector Machine

a. Data preprocessing

SVM just works with 2 classes but in this problem, we have to classify 10 genres music so we need to convert to a set of 2-class classification problems, and then solve each of these 2-class problems separately.

b. Linear SVM

SVM constructs a hyperplane in multidimensional space to separate different classes. SVM generates optimal hyperplane in an iterative manner, which is used to minimize an error. The core idea of SVM is to find a maximum marginal hyperplane that best divides the dataset into classes.

- Support Vectors

Support vectors are the data points, which are closest to the hyperplane. These points will define the separating line better by calculating margins. These points are more relevant to the construction of the classifier.

- Hyperplane

A hyperplane is a decision plane which separates between a set of objects having different class memberships.

Hyperplane: $\langle w \cdot x \rangle + b = 0$; w is an attribute weight; b is a real number

- Margin

A margin is a gap between the two lines on the closest class points. This is calculated as the perpendicular distance from the line to support vectors or closest points. If the margin is larger in between the classes, then it is considered a good margin, a smaller margin is a bad margin.

$$margin = \frac{2}{\|\mathbf{w}\|}$$

- Support Vector Machine ⇔ Solve maximization with the constraint :

Maximize:

$$margin = \frac{2}{\|\mathbf{w}\|}$$

Constraint:

$$\begin{cases} \langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b \geq 1, & if\ y_i = 1 \\ \langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b \leq -1, & if\ y_i = -1 \end{cases}$$

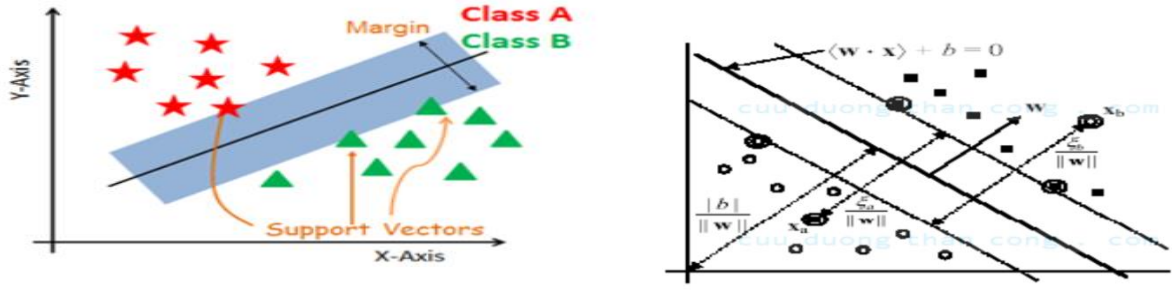


Figure 5. Illustration of Linear SVM and misclassification in data

- The dataset may contains many misclassification and error, as shown above.

- To deal with this situation, we assign the penalty parameter for the errors and integrate error in target optimal function:

Minimize:

$$\frac{\langle \mathbf{w} \cdot \mathbf{w} \rangle}{2} + C(\sum_{i=1}^{r} \xi_i)^k$$

C is the penalty degree parameter

- Choosing a suitable C for this problem

c. Non – linear SVM

Our dataset is non-linearly separable and the linear hyperplane can't solve it (as the illustration below).

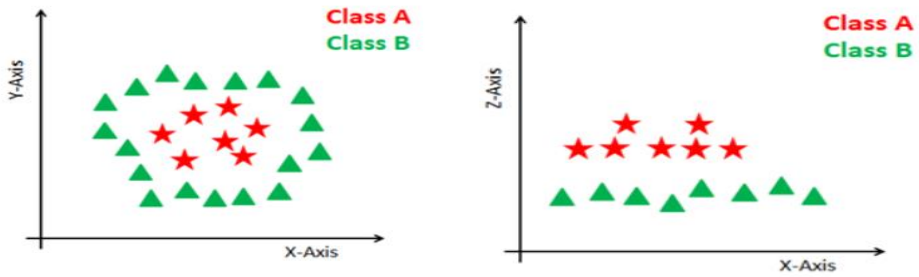


Figure 6.Non-linearly separable data and the result when applied kernel.

- To deal with this, we use a kernel trick to transform the input space to a higher dimensional space as shown on the right. The data points are plotted on the x-axis and z-axis (Z is the squared sum of both x and y: z=x^2=y^2). Now we can easily segregate these points using linear separation.

Kernel trick is a strategy using kernel function to take a low-dimensional input space and transform it into a higher dimensional space (converts nonseparable problem to separable problems by adding more dimension to it).

- Some common kernel function:

+ Polynomial:

$$K(\mathbf{x},\mathbf{z}) = \left(\langle \mathbf{x}\cdot\mathbf{z}\rangle + \theta\right)^d;\ \theta \in R, d \in N$$

+ Gaussian Radial Basis Function:

$$K(\mathbf{x},\mathbf{z}) = e^{\frac{\|\mathbf{x}-\mathbf{z}\|^2}{2\sigma}};\ \sigma > 0$$

+ Sigmoidal:

$$K(\mathbf{x},\mathbf{z}) = \tanh\left(\beta\langle \mathbf{x}\cdot\mathbf{z}\rangle - \lambda\right) = \frac{1}{1+e^{-(\beta\langle \mathbf{x}\cdot\mathbf{z}\rangle-\lambda)}};\ \beta,\lambda \in R$$

- To choose the most suitable kernel function and penalty parameter for this model, we use Grid Search algorithm.

2. Dense Neural Network

a. Data preparation

We will use the technique called feature scaling too normalize the range of independent variables or features of data.

Explanation**:** In our datasets, the range of our features differ greatly, whereas some features may have the values and the variance of hundreds, while other features might be even less than 1.

To solve this problem, we will use feature scaling:

Suppose we have a feature vector x. We will normalize the value of this feature x across all examples using the formula below:

$$x' = \frac{x - \bar{x}}{\sigma}$$

$\bar{x}$*: mean value of the feature vector*

σ*: variance of the feature vector*

By doing this, the mean of the attribute becomes zero and the resultant distribution has a unit standard deviation.

b. DNN architecture

- We will create a fully connected neural network with 3 hidden layers.
- Input layer:

The network's input layer will have the size of 40 neurons as there are 40 input features.

- Dropout regularization:

Since our model has quite a lot of parameters (over 160000 parameters), the model is quite easy to overfit.To avoid that, we will make use of a really popular regularization technique called Dropout. In Dropout regularization, we will go over each node in the hidden layers, and set a probability of eliminating that node off of our neural network.

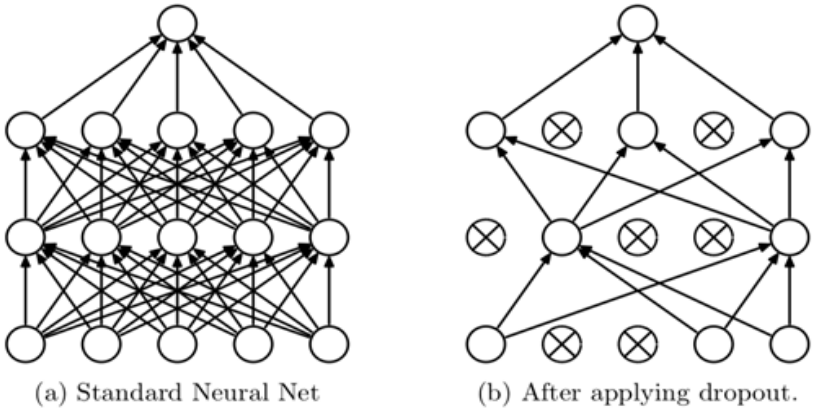


Figure 7. Visualization of the dropout regularization

➢ Explanation:

Since we randomly eliminate nodes from our neural network, the complexity of our model will be reduce, thus it helps us to avoid the overfitting problem. Also, dropout will stop the neural network from depending too much on one feature, so it has to spread out the weights more equally and makes our model more robust.

- Hidden layers:
  - We will use 3 fully connected hidden layers for our neural networks, with each of them having 512, 256, 32 neurons, respectively.
  - Activation function: we will choose the ReLU function as the activation as the output for all of our hidden layers, since the function has a cheap computational cost and converges quite easily.
  - Dropout regularization: similar to the input layer.
- Output layer:
  - Since we have 10 different classes representing 10 genres of music to classify, our output layer will have 10 neurons.
- Activation function:
  - Because this is a multi-class classification problem, our activation function for the output layer will be the softmax function.

➢ *Softmax function:*

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

z: *input vector of the softmax function*

z_i: *element of the input vector at index* i

K: number of classes in the multi-class classifier.

Here in our example, K = 10

$\sigma(z)_i$*: element at index* i *of the softmax function's output vector*

Here, the vector $\sigma(z)$ is a probability distribution for K classes in the classifier, with $\sigma(z)_i$ representing the probability that the example belongs to the class number i^{th}.

- For this particular problem, the input vector of size (10,1) and the output of the softmax function are shown below:

$$\vec{z} = \begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ \cdots \\ z_{10} \end{bmatrix} \quad \sigma(\vec{z}) = \begin{bmatrix} \frac{e^{z_1}}{\sum_{j=1}^{j=10} e^{z_j}} \\ \frac{e^{z_2}}{\sum_{j=1}^{j=10} e^{z_j}} \\ \frac{e^{z_3}}{\sum_{j=1}^{j=10} e^{z_j}} \\ \cdots \\ \frac{e^{z_{10}}}{\sum_{j=1}^{j=10} e^{z_j}} \end{bmatrix}$$

c. Backward Propagation

- Backward propagation algorithm: **Adam Optimizer**
  - The Adam Optimizer algorithm is the combination of Gradient Descent with Momentum and RMS Prop with mini batches.
- Gradient Descent with momentum:
  - During our training process, the standard gradient descent method might oscillate too much in undesired directions, making our learning process slower. For example, in the contour plot below, we can see that the gradient descent might oscillate in the horizontal axis quite much while moving slowly on the vertical axis. We can add momentum components into our learning process to speed up.

Figure 8. Contour plot of gradient descent

Implementation:

On iteration t:

Compute dW, db on the current mini-batch

$v_{dW} = \beta v_{dW} + (1 - \beta) dW$

$v_{db} = \beta v_{db} + (1 - \beta) db$

$W = W - \alpha v_{dW}, \; b = b - \alpha v_{db}$

W: weights matrices

b: bias matrices

α: learning rate

β: hyperparameter, usually initialized to 0.9

We can think of momentum components v_{dw}, v_{db} as the velocity of our current point, and dw, db as the accelerations of that point. Here, our velocity contains the information of the accelerations, and also the previous velocities, helping it to converge to our minimum faster.

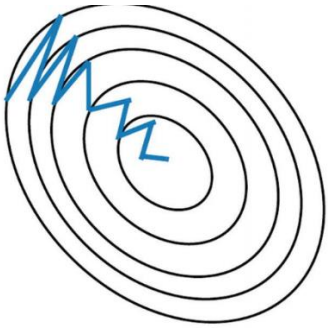
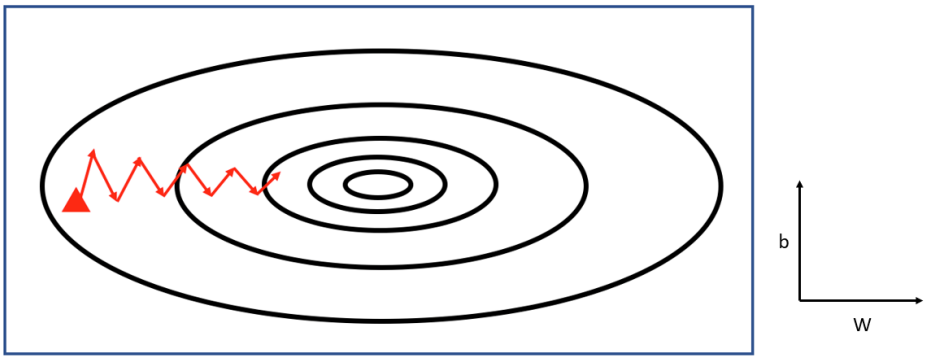


Figure 9: Demonstration of GD with momentum effects

- After using momentum, our gradient descent might look a little like the image above, where the gradient descent oscillates less on the horizontal axis while moving faster towards the vertical axis, thus speeding up our learning process.

- RMS Prop:
  - Similar to the Gradient Descent with momentum, RMSProp also helps dampen out the oscillation and speed up our training process.



- In the contour shown above,let's denote the vertical axis as the bias b, and the horizontal axis as the weight W. We would want the learning process to go pretty fast in the W direction, while slowing down in the b direction.
- Implementation:

On iteration t:

Compute dW, db on the current mini-batch

$$v_{dw} = \beta \cdot v_{dw} + (1 - \beta) \cdot dw^2$$

$$v_{db} = \beta \cdot v_{dw} + (1 - \beta) \cdot db^2$$

$$W = W - \alpha \cdot \frac{dw}{\sqrt{v_{dw}} + \epsilon}$$

$$b = b - \alpha \cdot \frac{db}{\sqrt{v_{db}} + \epsilon}$$

- We can see that dw is relatively small, and db is relatively large (based on the steeps of the contour). Thus the weighted average values Vdw is relatively small and Vdb is relatively large. When we update the weight W, we divide the learning rate α by the square root of Vdw, a small number, so we end up with a large learning rate, resulting in the learning process speeding up in the W direction. Similarly, the bias b is quite large, making the learning rate in the vertical direction to be smaller, thus dampen out the oscillation in the vertical axis.

- Adam Optimizer: the combination of Gradient Descent with Momentum and RMS Prop.
  - Implementation:

On iteration t:

Compute dW, db on the current mini-batch

$$V_{dw} = \beta_1 V_{dw} + (1 - \beta_1) dw \ , \ V_{db} = \beta_1 V_{db} + (1 - \beta_1) db$$

$$S_{dw} = \beta_2 S_{dw} + (1 - \beta_2) dw^2$$

$$S_{db} = \beta_2 S_{db} + (1 - \beta_2) db^2$$

$$W = W - \alpha \frac{V_{dw}}{\sqrt{S_{dw}} + \epsilon} \ , \ b = b - \alpha \frac{V_{db}}{\sqrt{S_{db}} + \epsilon}$$

β1 : the hyperparameter from the momentum process (set to 0.9)

β2 : the hyperparameter from the RMSProp process (set to 0.999)

W: weights matrices b: bias matrices

α: learning rate

ε: a really small number (10^{-8}) to prevent dividing by 0

- Hyper-parameters: After using grid search and experimenting, we finally choose the hyper-parameters as below to achieve the best performance:
    - Learning rate *α :* 0.0001
    - Size of the mini-batches: 64
    - Training process:
      We set a specific number of training epochs instead of applying early stopping. As we can see in the loss value plot of the Neural Network as below, after the loss values have converged for quite some time, both the training set and the validation's set loss still decrease. Also, letting the training process to carry on for some more time after converging (without overfitting of course) instead of early stopping tends to yield slightly better accuracíe for the training set, validation set, and the test set, from what we've observed.

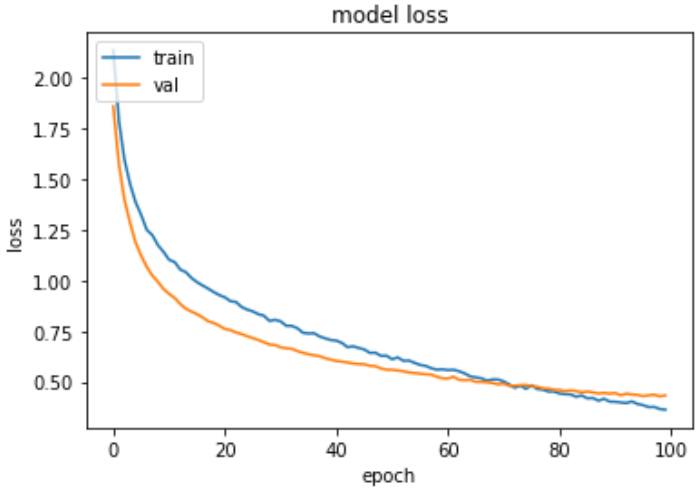


Figure 10: Loss changes of a training process.

- Final architecture

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_8 (Dense)              (None, 512)               20992
_________________________________________________________________
dropout_6 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_9 (Dense)              (None, 256)               131328
_________________________________________________________________
dropout_7 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_10 (Dense)             (None, 32)                8224
_________________________________________________________________
dropout_8 (Dropout)          (None, 32)                0
_________________________________________________________________
dense_11 (Dense)             (None, 10)                330
=================================================================
Total params: 160,874
Trainable params: 160,874
Non-trainable params: 0
```

3. Convolutional Neural Network

Convolutional Neural Network (CNN) is a type of neural network which is used widely with grid-like data (matrices and tensors) such as images. Image sample is represented as tensor with the dimension of (w, h, c) corresponding to the width and height of the image, c refers to the number of channels, which is often 1 for grayscale images and 3 (red, green. blue) for colored images. We can say that tensors are just c matrices stacked on each other. Each value in the tensor indicates how strong or bright a particular color (channel) is at a particular pixel. CNN has the ability to learn representative features of an image sample – it can learn the pattern of an image, or how every pixels in that image related to each other. The convolutional layers are designed such that they will learn to represent simple patterns first (lines, curves) and in the deeper layers there will be much

more complex patterns (face, eyes,…) and then we can put those features into fully connected layers to do classification or other tasks.

a. CNN Architecture

We will use one of the most popular CNN architecture, which contains three types of layers: Convolutional layer, Pooling Layer, Fully-connected layer.

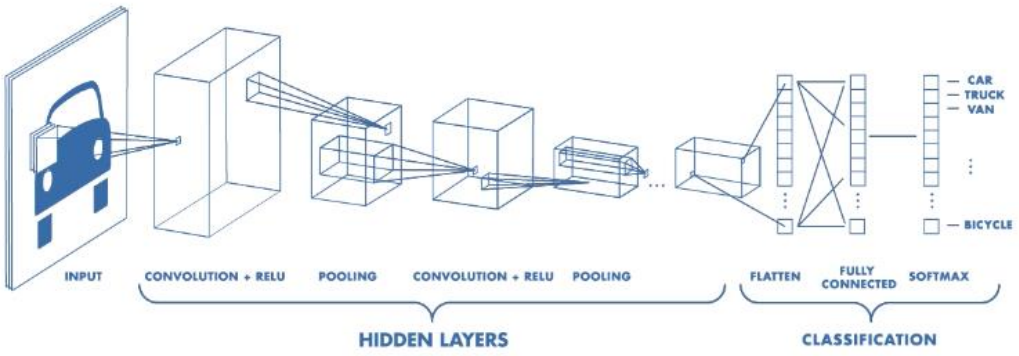


Figure 9. Example of a CNN architecture

- Convolutional layers

- This layer performs a dot product between corresponding matrices between 2 tensors, one of the tensor is called kernel, which contains learnable parameters, the other tensor is the one taken from part of the image tensor. The width and height of the kernel must be smaller than that of the image tensor and the number of channels between the two must be equals. Usually, the width and height of the kernel are equal.
- During the forward pass, the kernel slides from left to right, then from top to bottom to perform dot product of 1 channel. Then it sums up the values of the dot product matrix. The same process is done with all channels. Finally, the values from every channels are sumed up together with a bias value to produce a value in the output tensor. The sliding step of the kernel is called a stride. Figure 10. shows the process of a convolutional operation between an input tensor of size (w, h, 3) and a kernel of size (3, 3, 3), with stride of 1. The intuition of this process is that the kernel will try to find a relation of pixels in the regarding area to product the output value. Each kernel has its own role, such as a kernel extracts horizontal lines, the other finds vertical lines…

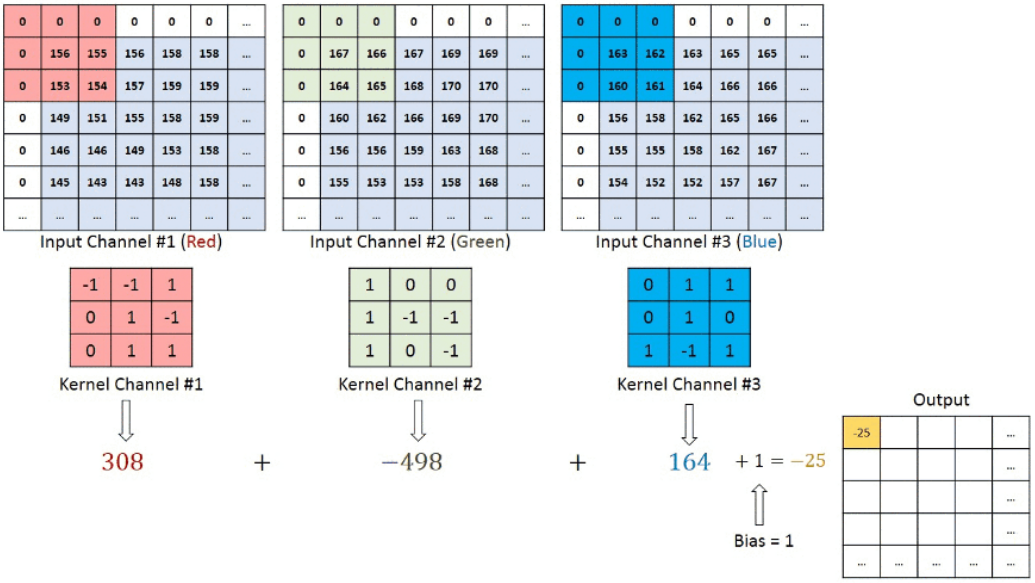


Figure 10. Example of a convolution operation

- Notice every time we perform this operation, the size of the output tensor is smaller than the size of the input one. This means that the pixels near egdes are checked by the kernel at most 2 times, while they may contains important information. We can apply a technique called padding, which concatenate lists of zeros to the 4 egdes so that the pixels at edges are checked more carefully. Usually, the padding size is 1 as shown in the figure above.
- The number of kernels used is freely chosen. If we have an input of size (W, W, D) and D_{out} number of kernels with a size of (F, F, D) with stride S and amount of padding P, then the size of output volume can be determined by the following formula:

$$W_{out} = \frac{W - F + 2P}{S} + 1$$

- This will result in an output tensor of size (W_{out}, W_{out}, D_{out}).
- The number of kernels used is freely chosen, but it is recommeded that the number of kernels of deeper layers should be bigger than that of the previous layers. The reason is that we want the deeper layers to extract more complex features, and the number of complex features will certainly bigger than the number of simple features. The first layers can just extract lines, curves, the second layers find rectangle, cirle and so on.

• Pooling layers

- Although using padding technique can help us keep the information of pixels lying on egdes, the computational cost might be a concern. Using padding =

1 and stride = 1 results in same width and height for the output layers, which can cause computational problem as we tend to use more kernels in deeper layers. So we need a way to shrink the output size, while still keeping important information. This is where Pooling layers come in to play.

- There are 2 widely used pooling layers, which are Average Pooling and Max Pooling. In this project, we will use Max Pooling. Max Pooling also use kernel to slide through the input tensor, except that the kernel does not compute dot product but to find the maximum values in the regarding area.
- If we have an input tensor of size (W, W, D), a pooling kernel of size (F, F, D), and stride S, then the size of output volume can be determined by the following formula:

$$W_{out} = \frac{W - F}{S} + 1$$

- This will result in an output volume of size (W_{out}, W_{out}, D).
- Extracting the maximum values in the regarding area means that we take only the color which is the brightest and this pixel contains more information than the others.
- In all cases, pooling provides some translation invariance which means that an object would be recognizable regardless of where it appears on the frame.

- Fully-connected layers

- The output tensor of the above-mentioned layers are then flatten to product a vector input that can be fed to fully-connected layers.
- These layers have nothing special, the structure of them is the same as the structure of DNN.

- Final architecture

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 108, 128, 16)      448
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 54, 64, 16)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 54, 64, 32)        4640
```

max_pooling2d_1 (MaxPooling2	(None, 27, 32, 32)	0
conv2d_2 (Conv2D)	(None, 27, 32, 64)	18496
max_pooling2d_2 (MaxPooling2	(None, 13, 16, 64)	0
conv2d_3 (Conv2D)	(None, 13, 16, 128)	73856
max_pooling2d_3 (MaxPooling2	(None, 6, 8, 128)	0
flatten (Flatten)	(None, 6144)	0
dropout (Dropout)	(None, 6144)	0
dense (Dense)	(None, 1024)	6292480
dropout_1 (Dropout)	(None, 1024)	0
dense_1 (Dense)	(None, 256)	262400
dense_2 (Dense)	(None, 128)	32896
dropout_2 (Dropout)	(None, 128)	0
dense_3 (Dense)	(None, 32)	4128
dropout_3 (Dropout)	(None, 32)	0
dense_4 (Dense)	(None, 10)	330

Total params: 6,689,674
Trainable params: 6,689,674
Non-trainable params: 0

IV. Experiments

1. Datasets used:

- GTZAN dataset.
- Mixed dataset

To improve the models' performance on 'real world' data, we decided to collect more audio files from the internet. We collected 10 different playlists from Youtube for each genre in the data, and downloaded the first 50 songs from each playlist for the training and validation process. For each class, to preserve the balance of the dataset, we choose 33% of the samples to mix into the original GTZAN to create a new dataset.

For the two datasets, each sample record is 30 seconds long. We will split each sample into 9 subsamples. By doing this we have much more data to train: the GTZAN dataset is now having 9000 samples, while the mixed one has about 15000 samples. Furthermore, now that each sample record is shorter, the features input to models describe the samples more correctly and reduce the sequential effect of the data.

- Test dataset

To prepare test data, we crawled 11 songs for each genre separately from Youtube (we actually crawled 2 more songs for the reggae class, since we came across a few problems with audio file corruption and extracting, but this won't have a lot of effects on our evaluation process).

To evaluate our models performances, we will split each track into 9 files, as the samples input to CNN model have to match the input sizes required in the network architectures, since we also split up the audio files in the training data. Otherwise, the input image will be resized to match the first layer requirements, which will affect the model prediction.

2. Algorithms used:

We will use the 3 algorithms discussed above.

3. Evaluation metrics:

Since the numbers of samples of every class are the same, we will just use accuracy to evaluate the performance of models.

4. Experimental results:

Table 1. Performance of models on the original GTZAN dataset.

Set	DNN	SVM	CNN
Train	87.81	84.37	77.27
Val	86.00	87.79	72.52
Test_Youtube	25.10	20.14	29.13

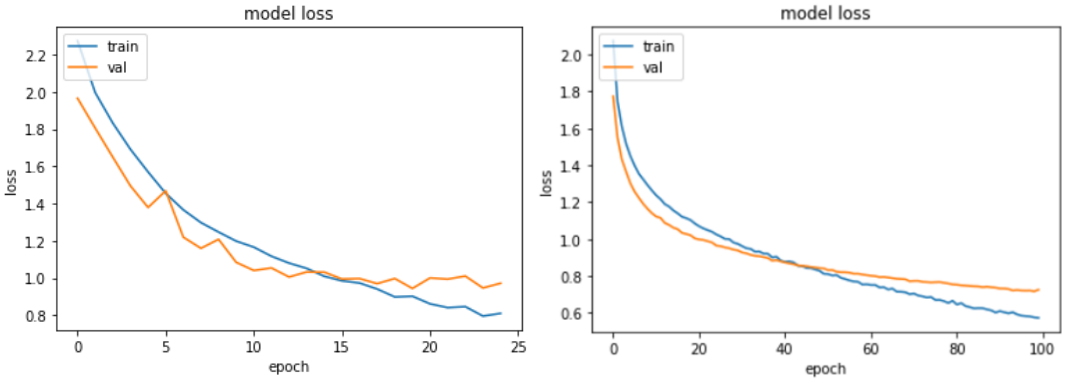
We can see that all three models achieved great accuracy on the training and the validation set, around 85 - 90% accuracy for Neural Network and Support Vector Machine, and 72.52% in the validation set for Convolutional Neural Network.

Despite this, their accuracy on our collected dataset, which can be considered more realistic, is significantly lower. However, as the CNN performs not so good on the original dataset, it achieves the best accuracy on the test set among the three. This proves that CNN has better generalization of the data.

The test results being quite low compared to the training and validation results can be attributed to the fact that the GTZAN dataset being quite old (from 2002), so it might not be up to date with the current songs anymore. Also, the GTZAN datasets are carefully picked and processed, while our test set might be more “messy”, meaning they’re randomly chosen and not carefully picked and processed as the GTZAN dataset. In reality, we might want to train on a more “messy” but realistic dataset, so that we can generalize the sound better and achieve greater performance on real data.

Table 2. Performance of models on the mixed dataset.

Set	DNN	SVM	CNN
Train	80.46	77.16	73.94
Val	74.09	76.31	68.67
Test_Youtube	36.71	40.48	48.19



Loss plot of CNN (left) and DNN (right).

We can see that the performance of the three algorithms on the mixed data is not as good as it is on the original dataset (with NN and SVM decreasing about 10% in the validation set's accuracy, and CNN slightly decreasing from 72.52% to 68.87%). This is mostly due to the fact that the GTZAN dataset is much more carefully prepared, and the audio files have a lot of similarities. Meanwhile, the mixed dataset is much more diverse and different from each other. We can conclude this as we examine the mel spectrograms of 2 random songs from the same class in the two datasets. We randomly took 9 samples from the training data, and there are three images of the country genre that do not look very similar to each other. The taken samples are illustrated in the figure below.

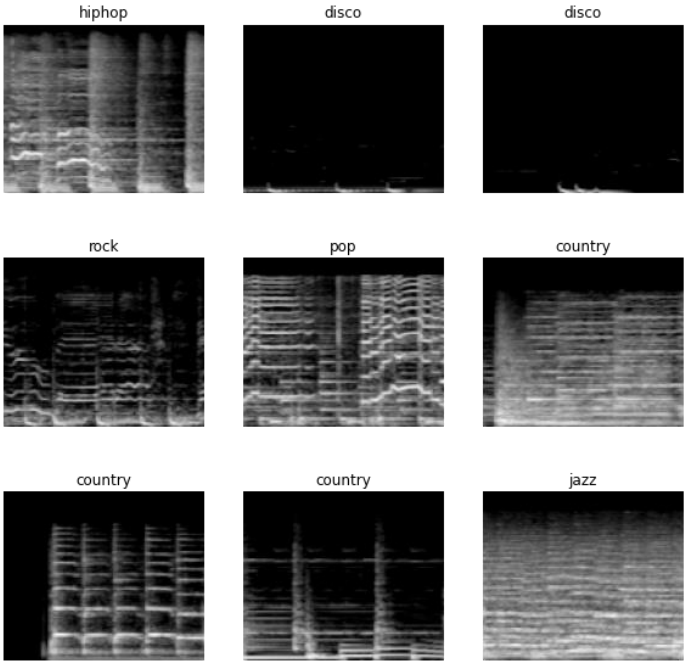


Figure 11. Random samples taken from mixed dataset

The models training on the GTZAN dataset will learn much better due to the similarities between training instances. But, it won't generalize as well as on our mixed dataset, since the sounds of other songs of the same genres might vary greatly. The models trained on our mixed dataset will likely perform better on the 'real world' data.

As we can see, all 3 models performed much better on the test dataset after training them with other mixed data, especially CNN going from 29.13% to 48.19% accuracy.

- Improvement for NN models:

During our evaluating process, we used the split dataset to evaluate our models(where one 30 second file is split into 9 subsamples). But during our experimentation, we found out that passing in a full 30 seconds file slightly increased the test result of the NN model, going from 36.71 % to 46.0 % for NN. This might be because the MFCC coefficients of the full audio file might represent its characteristics better than those of smaller subsamples.

5. Conclusions

After experimenting with the models, we have found that while the models training on the GTZAN dataset might achieve great accuracy on the training and validation set, their performance are quite poor on new data (only around 20 - 30%), due to the fact that the GTZAN dataset doesn't generalize well for all songs.

By adding more data collected from the internet, we managed to increase the models' performance, with the best being CNN (48.19%).

The CNN model stood out of the three models, achieving the best test results for both cases.

Despite the improvement, the models' accuracy is still not quite good, since the sounds of many genres are hard to generalize, especially for diverse genres like pop, rock...

V. Discussion

1. Performance & Implementation of models

a. Support Vector Machine

There are many C (penalty parameter) and kernel function having to choose, so we use Grid Search algorithm to choose the most suitable kernel function and penalty parameter for this model.

b. Dense Neural Network

Artificial Neural Network seems to work quite well for this problem and we didn't come across a lot of difficulties implementing it. After applying grid search on parameters such as the learning rate, drop-out rate, the optimization algorithm(SGD, RMS Prop, GD with momentum, Adam…)... we picked the model with the best performance.

c. Convolutional Neural Network

The CNN model used in this project is not achieving very good performance on the two datasets, as the accuracy is not really high as we expect. This is due to the simple architecture of the model we use, which has very few layers and parameters compared to other popular models such as VGG and ResNet. We want to test a simple CNN model to see if it can be used in 'real world' situations, as this model will certainly run faster than complicated CNN models described above. Furthermore, the model is trained from scratch, which tends to result in overfitting, using transfer learning technique may have better result.

2. Results assesments

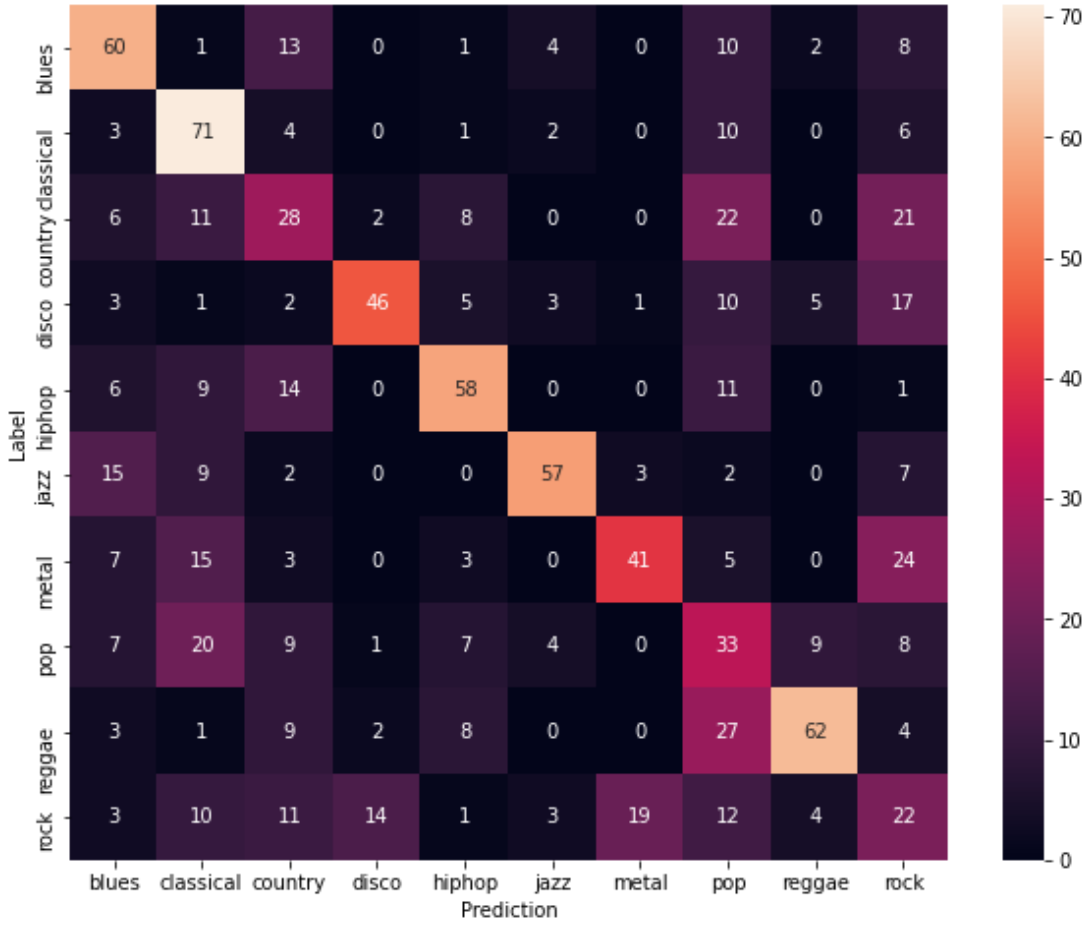


Figure 12. Confusion matrix of CNN evaluated on Youtube test set

We have plotted the confusion matrix of the CNN - our best model – on the Youtube test set, as shown in Figure 12 above. The horizontal axis denotes predicted genres by our model, and the squares on the vertical axis denote the actual genres. Thus, the number of correctly classified instances for classes is represented on the left diagonal of this matrix.

Classical is the genre in which our model achieved the best performance (with 71 songs out of 97 correctly identified, respectively). This is mostly because this genre has quite unique sounds compared to other modern genres, and thus are really easy to distinguish from other genres of music.

Blues,reggae, hiphop, jazz achieved quite great accuracy as well. Like classical, these genres have unique characteristics that can be quite different from conventional pop/rock songs (like hiphop's drum beats, or jazz improvised instrumental solos…), and thus helps them to be more identifiable.

In contrast, some genres' predicted accuracy is quite low, and is frequently misidentified like rock, country , pop (only 22/118 , 28/95, 33/98, respectively). They tend to get confused with similar genres of music (20/98 pop songs were identified as rock, 22/95 country songs were identified as pop, or 19/118 rock songs were identified as metal...). As claimed by human experts, there is not really a bright line to separate these types of music, and one song can be classified as multiple genres. Many people have difficulties classifying the genres, if not music specialists.

To solve this problem in the future, we can have a better grouping of classes for this problem (like grouping rock and metal...), and also have other new genres appear as well. Also, a multi-label classification model might yield greater accuracies, and realistically more applicable, since not every song has characteristics from only one genre.

3. Future works

The simple CNN model easily gets overfit if trained with a large number of epochs. We will implement a data augmentation technique on spectrograms called SpecAugment, which has gained its popularity on speech processing tasks such as speech recognition and voice biometrics.

We will try other audio features such as zero-crossing rate, spectral centroid to see if they are more appropriate for the task.

Using a sequence model on these datasets will likely have better performance, as the data is time series data. We will try models such as RNN and LSTM and hopefully achieve higher accuracy and better generalization.

VI. Members Assignments

a. Features extraction & Dataset Introspection

- Phạm Việt Thành

b. SVM Implementation

- Đặng Vũ Hoàng Hiệp

c. DNN Implementation

- Nguyễn Tuấn Dũng

d. CNN Implementation

- Phạm Việt Thành

e. Youtube data crawling

- Nguyễn Tuấn Dũng

f. Performing evaluation & Results Assesments

- Phạm Việt Thành
- Nguyễn Tuấn Dũng

g. Discussions

- Phạm Việt Thành
- Nguyễn Tuấn Dũng
- Đặng Vũ Hoàng Hiệp

VII. References

- https://www.coursera.org/learn/machine-learning
- https://www.coursera.org/specializations/deep-learning
- https://www.youtube.com/watch?v=zfiSAzpy9NM
- http://marsyas.info/downloads/datasets.html - GTZAN dataset
- https://vbn.aau.dk/ws/portalfiles/portal/74499095/GTZANDB.pdf - GTZAN dataset analysis.
- https://librosa.org/ - library used for features extraction.
- https://www.tensorflow.org/ - framework used for DNN, CNN implementation.
- https://scikit-learn.org/ - framework used for SVM implementation.
- https://github.com/ytdl-org/youtube-dl - library used for Youtube data crawling.