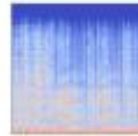


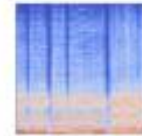
Music Genre Classification Computer Music Course 2022 Spring Semester

Aris Tsilifonis – 1115201700170 ,
Georgios Haritos 1115201300229

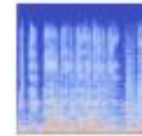
blues



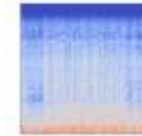
classical



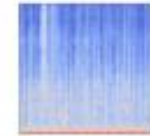
country



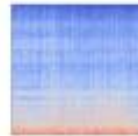
disco



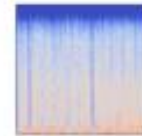
hiphop



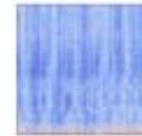
jazz



metal



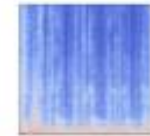
pop



reggae



rock



Project Description

Introduction : Page 3

Disclaimer: Page 4

Datasets used in training and testing : Page 5

Data manipulation : Page 6

Using LSTM : Page 13

- Theory : Page 13

- Implementation – Model Architecture : Page 15

- Results : Page 17

Using CNN : Page 18

- Theory : Page 18

- Implementation – Model Architecture : Page 19

- Results : Page 21

Conclusion : Page 23

Introduction

- The goal of our project was to classify .wav sound files into music genres. Music genre classification can play an important role in recommendation systems for music platforms like Spotify , YouTube and more . In order to build a more personalized playlists for the users , we need a model that can categorize the music for them. The idea behind the project was to learn how to manipulate music files in python and understand how to extract features from them. Finally, we implement some machine learning algorithms to get the information about the genres.

Disclaimer

- Our work was heavily based on Stanford paper
<http://cs229.stanford.edu/proj2018/report/21.pdf>
- We do not own the GTZAN dataset that was used for the experiments

Datasets used in training and testing

- GTZAN dataset consists of 1,000 song samples, each one last 30 seconds . The songs belong to a total of 10 standard music genres. The specific genres are : blues, classical, country, disco, hip-hop, jazz, metal, pop, reggae, and rock.
- We use a version of GTZAN dataset called feature_3_sec.csv. This file has the same structure as the main one, but the songs were split before into 3 seconds audio files (this way increasing 10 times the amount of data we fuel into our classification models). With data, more is always better.
- The feature_3_sec file contains features of the audio files that we represent below.

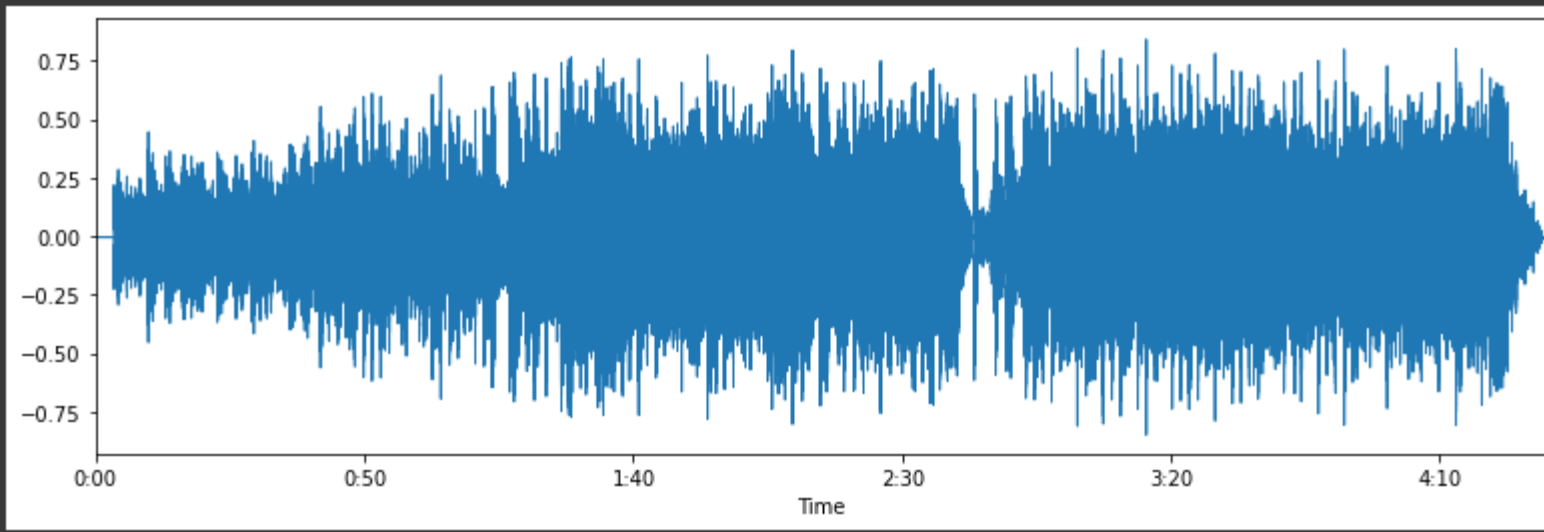
Data manipulation

A look at the data

- In our assignment we attempted to visualize the data to understand them better

We represent time on the x-axis and amplitude on the y-axis.

Keep in mind ,similar sound graphs mean similar genres !

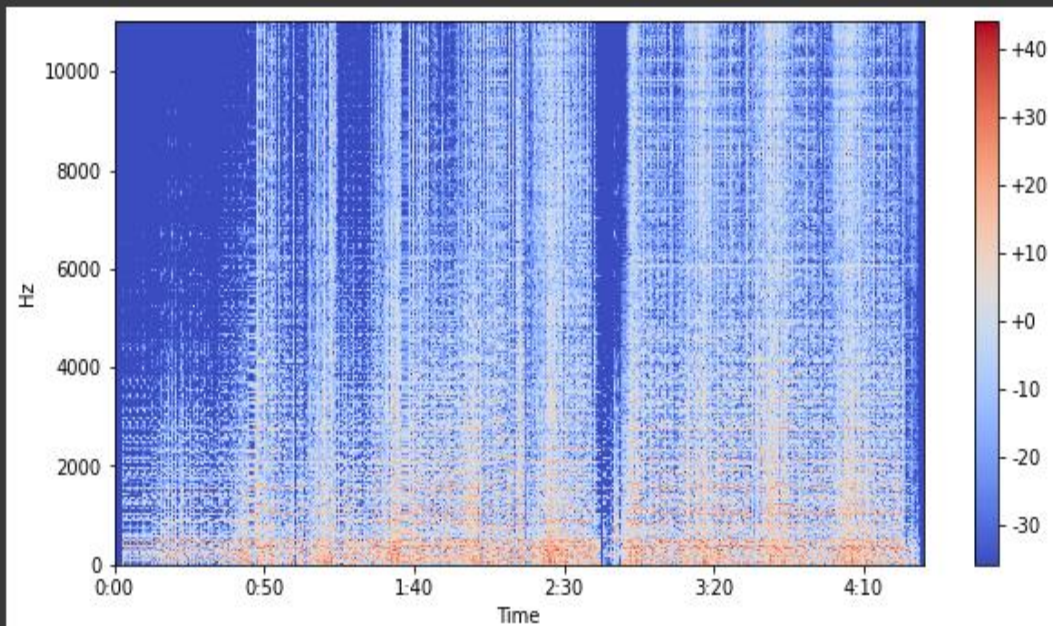


We used Librosa
Python library to
create the diagrams

This is sound wave graph of
Don't Speak song of No Doubt
band

Data manipulation

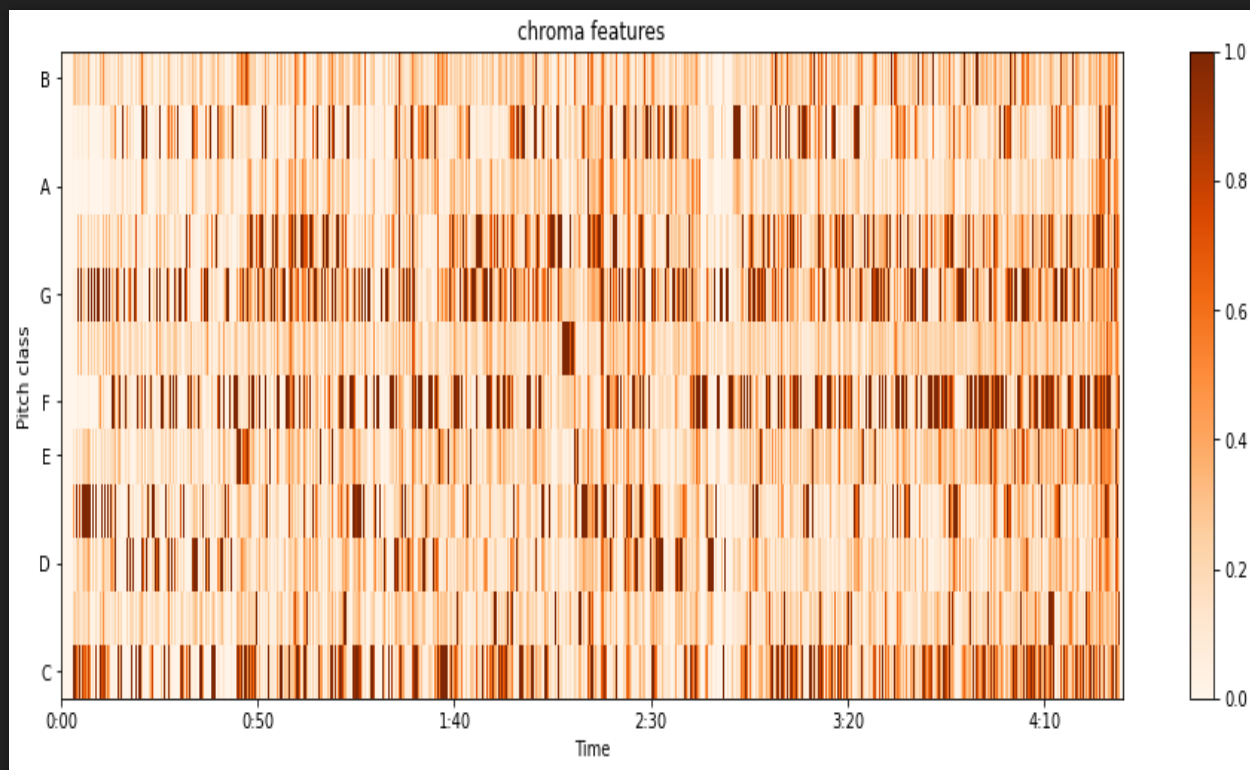
A look at the data



On the left, we represent the spectrogram of the audio file . The y axis is frequency while the x axis is the time. With this diagram we can see how the energy levels vary over time. Also, we can see where there is more or less energy . Its a way to show the signal loudness by the specific waveform.

Data manipulation

A look at the data



Chroma features are an interesting and powerful representation for music audio in which the entire spectrum is projected onto 12 bins representing the 12 distinct semitones (or chroma) of the musical octave. Since, in music, notes exactly one octave apart are perceived as particularly similar, knowing the distribution of chroma even without the absolute frequency (i.e. the original octave) can give useful musical information about the audio -- and may even reveal perceived musical similarity that is not apparent in the original spectra. They capture harmonic and melodic characteristics of music while being robust to changes in timbre and instrumentation.

Data manipulation

A look at the data

Review the file's columns The dataset has 58 columns which represent its characteristics. For example bandwidth, rolloff, tempo etc.

We use the 2 seconds version of this dataset to compensate for the small input.

```
[ ] gtzan_music_data.info()
```

0	filename	9990	non-null	object
1	length	9990	non-null	int64
2	chroma_stft_mean	9990	non-null	float64
3	chroma_stft_var	9990	non-null	float64
4	rms_mean	9990	non-null	float64
5	rms_var	9990	non-null	float64
6	spectral_centroid_mean	9990	non-null	float64
7	spectral_centroid_var	9990	non-null	float64
8	spectral_bandwidth_mean	9990	non-null	float64
9	spectral_bandwidth_var	9990	non-null	float64
10	rolloff_mean	9990	non-null	float64
11	rolloff_var	9990	non-null	float64
12	zero_crossing_rate_mean	9990	non-null	float64
13	zero_crossing_rate_var	9990	non-null	float64
14	harmony_mean	9990	non-null	float64
15	harmony_var	9990	non-null	float64
16	perceptra_mean	9990	non-null	float64
17	perceptra_var	9990	non-null	float64
18	tempo	9990	non-null	float64
19	mfcc1_mean	9990	non-null	float64
20	mfcc1_var	9990	non-null	float64

Data manipulation

- In order to pass the data to the models, we need to prepare the data for them .

- For the LSTM algorithm:

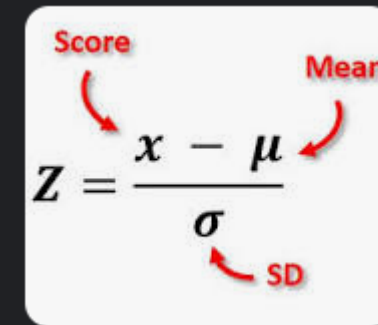
The LSTM class in Keras needs each input sample to be a 'block' of time or consisting of samples from a fixed number or window of time-steps.

For example, a structure of 5 time-steps – `samples[0:5]` – would be trained to predict `target[5]`. Thus, we will use samples falling inside the window to predict the sample immediately after the window ends.

Data Manipulation

```
] X_arrays = X.to_numpy()
print(X_arrays.shape)
X_list = []
y_list = []
window = 50

for i in range(X_arrays.shape[0]-window):
    X_list.append( X_arrays[i:window+i,:].tolist())
    y_list.append(y[i])
```



The diagram shows the Z-score formula $Z = \frac{x - \mu}{\sigma}$ with red arrows pointing to each part: 'Score' points to Z, 'x' is labeled as 'Score', 'μ' is labeled as 'Mean', and 'σ' is labeled as 'SD' (Standard Deviation).

- For the CNN algorithm we used a standard scaler to pass the data

Standard scaler is used to standardize features by removing the mean and scaling to unit variance . The standard score of sample x is calculated as:

$z = (x - \mu) / \sigma$.Standardization of a dataset is a common requirement for many machine learning estimators: they might behave badly if the individual features do not more or less look like standard normally distributed data.

Data Manipulation

- We used a label encoder because our targets in the dataset are words . We need to convert those words to numbers in order to pass them to the model.

As it is shown , every class has the same number of elements. As a result , the model will be less biased about one class(Equal probability to predict every class).

```
[ ] print(class_list )

0    blues
1    blues
2    blues
3    blues
4    blues
...
9985   rock
9986   rock
9987   rock
9988   rock
9989   rock
Name: label, Length: 9990, dtype: object

[ ] mapping = dict(zip(converter .classes_, range(1, len(converter .classes_)+1)))
print(mapping)

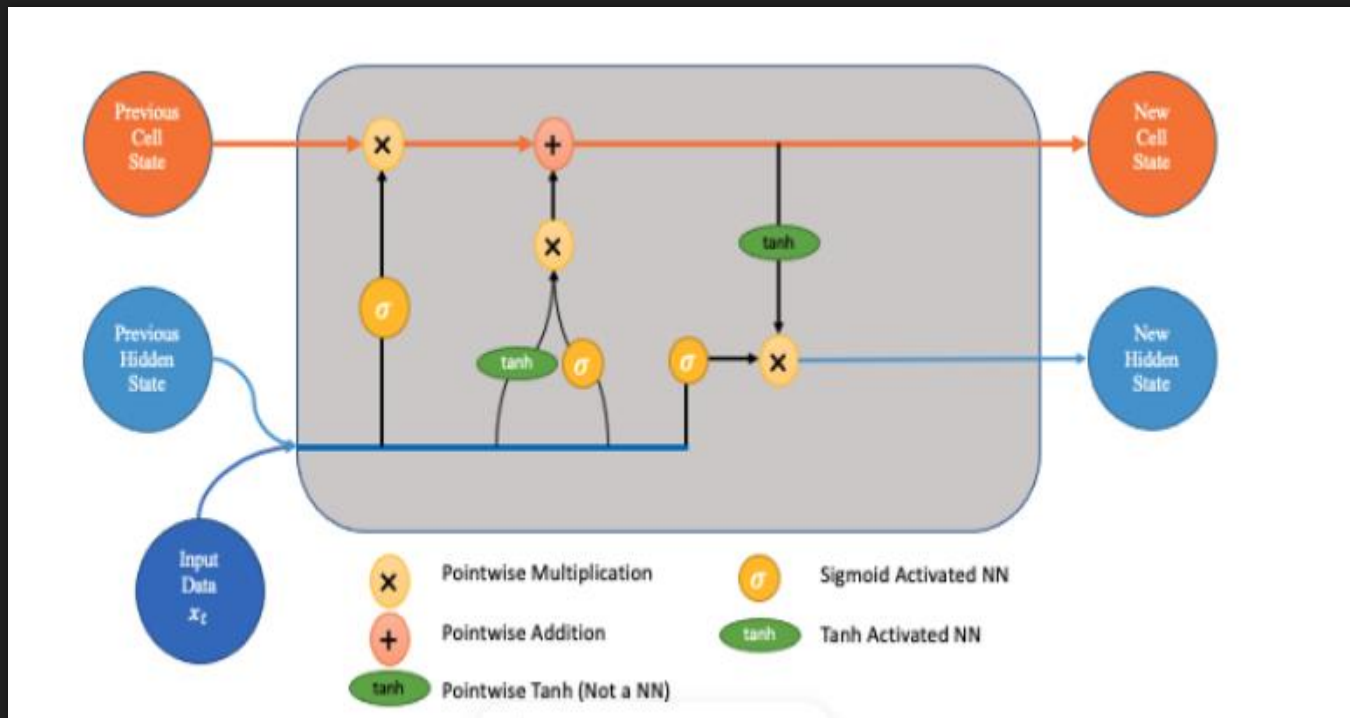
{'blues': 1, 'classical': 2, 'country': 3, 'disco': 4, 'hiphop': 5, 'jazz': 6, 'metal': 7, 'pop': 8, 'reggae': 9, 'rock': 10}
```

```
[ ] gtzan_music_data['label'].value_counts()
```

```
blues      1000
jazz       1000
metal      1000
pop        1000
reggae     1000
disco      999
classical  998
hiphop     998
rock       998
country    997
Name: label, dtype: int64
```

Using LSTM

How LSTM work - Theory



Firstly, at a basic level, the output of an LSTM at a particular point in time is dependent on three things:

- The current long-term memory of the network — known as the *cell state*
- The output at the previous point in time known as the previous *hidden state*
- The input data at the current time step

In the explanation on the left, we consider an LSTM cell as visualised in the following diagram. When looking at the diagrams in this article, imagine moving from left to right.

Using LSTM

How LSTM work - Theory

- LSTMs use a series of 'gates' which control how the information in a sequence of data comes into, is stored in and leaves the network. There are three gates in a typical LSTM; forget gate, input gate and output gate. These gates can be thought of as filters and are each their own neural network. You can explore more on this site : <https://towardsdatascience.com/lstm-networks-a-detailed-explanation-8fae6aefc7f9>
- LSTM networks were designed specifically to overcome the long-term dependency problem faced by recurrent neural networks RNNs (due to the vanishing gradient problem). LSTMs have *feedback* connections which make them different to more traditional *feedforward* neural networks. This property enables LSTMs to process entire sequences of data (e.g. time series) without treating each point in the sequence independently, but rather, retaining useful information about previous data in the sequence to help with the processing of new data points. As a result, LSTMs are particularly good at processing sequences of data such as text, speech and general time-series.

LSTM

Implementation – Model Architecture

LSTM makes predictions given sequences of data , thats why we use tf.keras.Sequential()

```
import tensorflow as tf
input_shape = X_train.shape[1:]
model2 = tf.keras.Sequential()
model2.add(tf.keras.layers.LSTM(64,return_sequences = True, input_shape = input_shape))
model2.add(tf.keras.layers.LSTM(64))
model2.add(tf.keras.layers.Dense(64 , activation = 'relu'))
model2.add(tf.keras.layers.Dense(10 , activation = 'softmax'))

optimizer = tf.keras.optimizers.Adam(learning_rate = 0.001)
model2.compile(
    optimizer = optimizer,
    loss = 'sparse_categorical_crossentropy',
    metrics = ['accuracy']
)
model2.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
=====		
lstm (LSTM)	(None, 50, 64)	31488
lstm_1 (LSTM)	(None, 64)	33024
dense (Dense)	(None, 64)	4160
dense_1 (Dense)	(None, 10)	650
=====		
Total params: 69,322		
Trainable params: 69,322		
Non-trainable params: 0		

```
[ ] model2.fit(X_train,y_train, validation_data = (X_val,y_val) , batch_size = 32, epochs = 60 , verbose = 2 )
```

LSTM

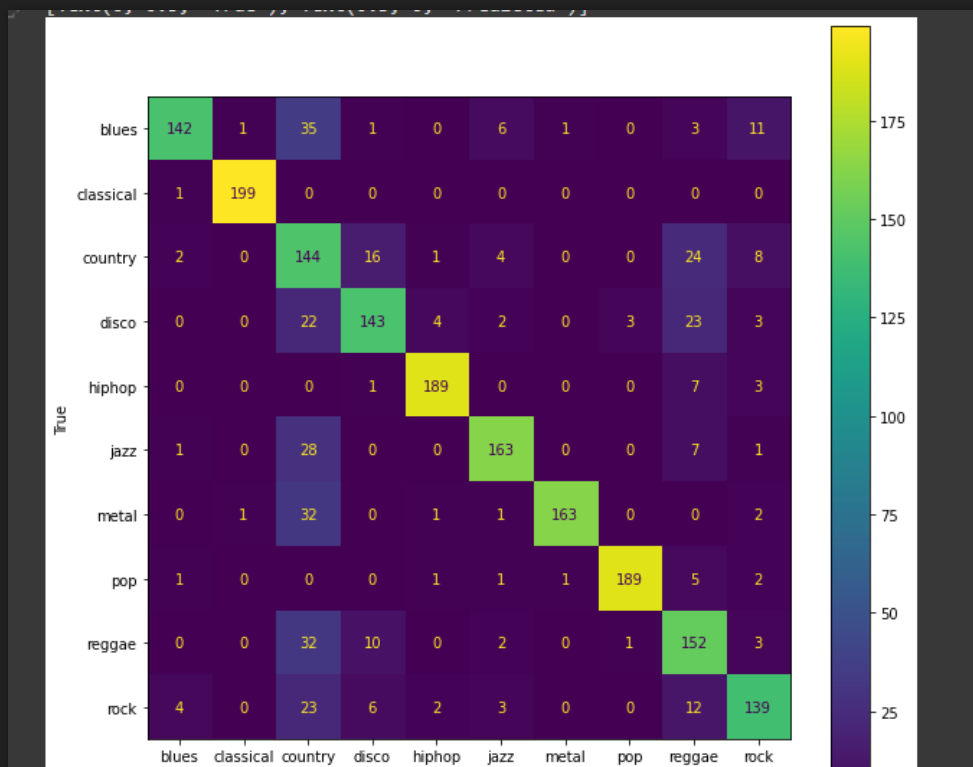
Implementation – Model Architecture

- On the diagram above , we have stacked two LSTM modules followed by Dense layers for activation. Also , we could change the implementation to include more LSTM layers or replace them with BiLSTM layers to improve performance.
- Next, we compile and fit the model using standard parameters: the Adam optimizer, sparse categorical cross-entropy loss (this performed better than categorical cross-entropy), and a 0.001 learning rate. Training the set on 60 epochs is enough for the classifier model to converge with 80% validation accuracy.

LSTM

Results

Confusion matrix

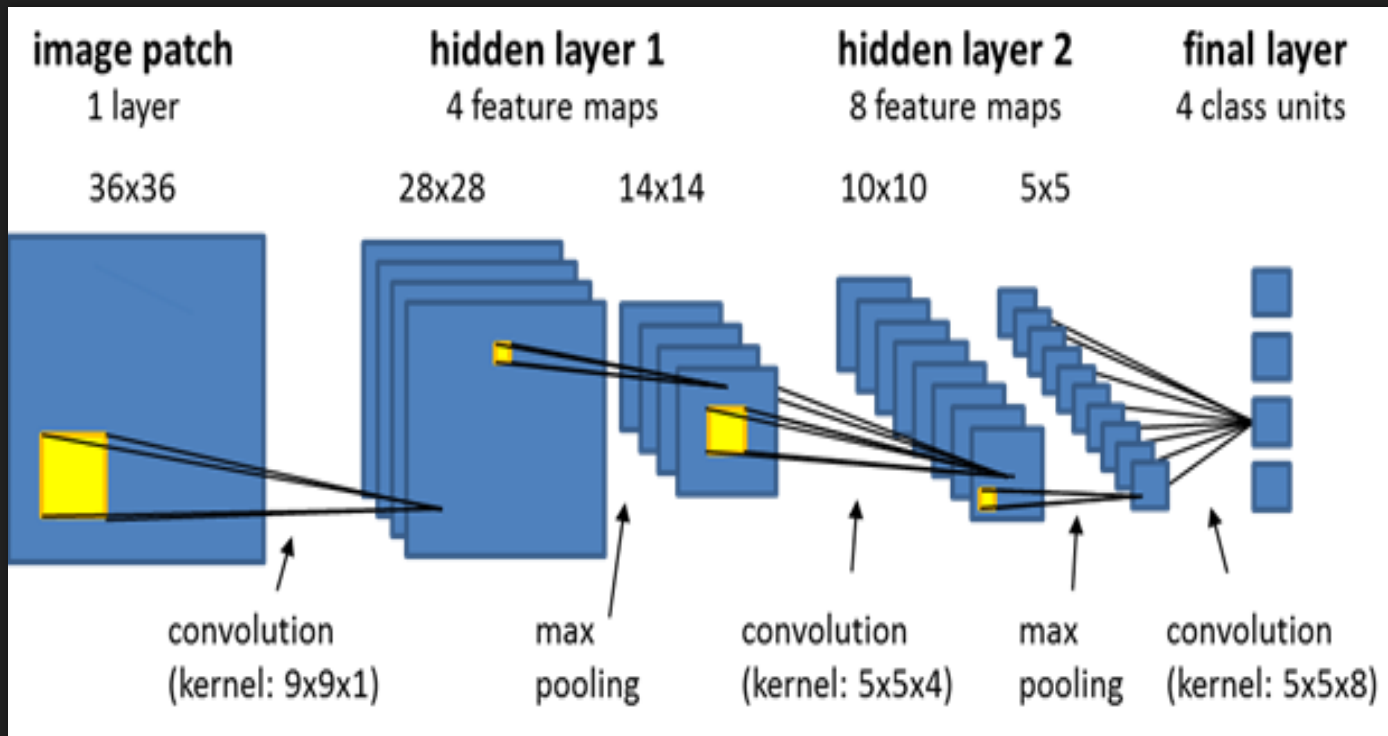


The diagonal of the diagram means that the predicted value was the same as the true value . The other values mean that the predicted values were not the same as the true values.

As we see from the diagram , the results are quite satisfying. The majority of the samples are predicted correctly. On the other hand there are some quite important flaws in our model. For example, blues genre has 35 samples that were wrongly predicted as country genre . Moreover, it has 11 samples that were wrongly predicted as rock. The model accumulates 58 false negatives which are substantial for our work . We will try to improve the results with cnn algorithm.

Using CNN

Theory



A CNN uses a system much like a multilayer perceptron that has been designed for reduced processing requirements. The layers of a CNN consist of an input layer, an output layer and a hidden layer that includes multiple convolutional layers, pooling layers, fully connected layers and normalization layers. The removal of limitations and increase in efficiency for image processing results in a system that is far more effective, simpler to train and limited for image processing and natural language processing.

Using CNN

Implementation – Model Architecture

```
[ ] model3 = tf.keras.Sequential()  
    model3.add(tf.keras.layers.Dense(512,activation='relu',input_shape = (x_train.shape[1],)))  
    model3.add( tf.keras.layers.Dropout(0.2))  
    model3.add(tf.keras.layers.Dense(256,activation='relu'))  
    model3.add(tf.keras.layers.Dropout(0.2))  
    model3.add(tf.keras.layers.Dense(128,activation='relu'))  
    model3.add(tf.keras.layers.Dropout(0.2))  
    model3.add(tf.keras.layers.Dense(64,activation='relu'))  
    model3.add(tf.keras.layers.Dropout(0.2))  
    model3.add(tf.keras.layers.Dense(32,activation='relu'))  
    model3.add(tf.keras.layers.Dropout(0.2))  
    model3.add(tf.keras.layers.Dense(10,activation='softmax'))  
  
    print(model3.summary())  
    model_history = trainModel(model=model3,epochs = 300,optimizer = 'Adam')
```

```
53/53 [=====] - 1s 13ms/step - loss: 0.0226 - accuracy: 0.9930 - val_loss: 0.4682 - val_accuracy: 0.9236
```

```
Epoch 299/300
```

```
53/53 [=====] - 1s 12ms/step - loss: 0.0290 - accuracy: 0.9915 - val_loss: 0.4956 - val_accuracy: 0.9187
```

```
Epoch 300/300
```

```
53/53 [=====] - 1s 12ms/step - loss: 0.0382 - accuracy: 0.9895 - val_loss: 0.4783 - val_accuracy: 0.9202
```

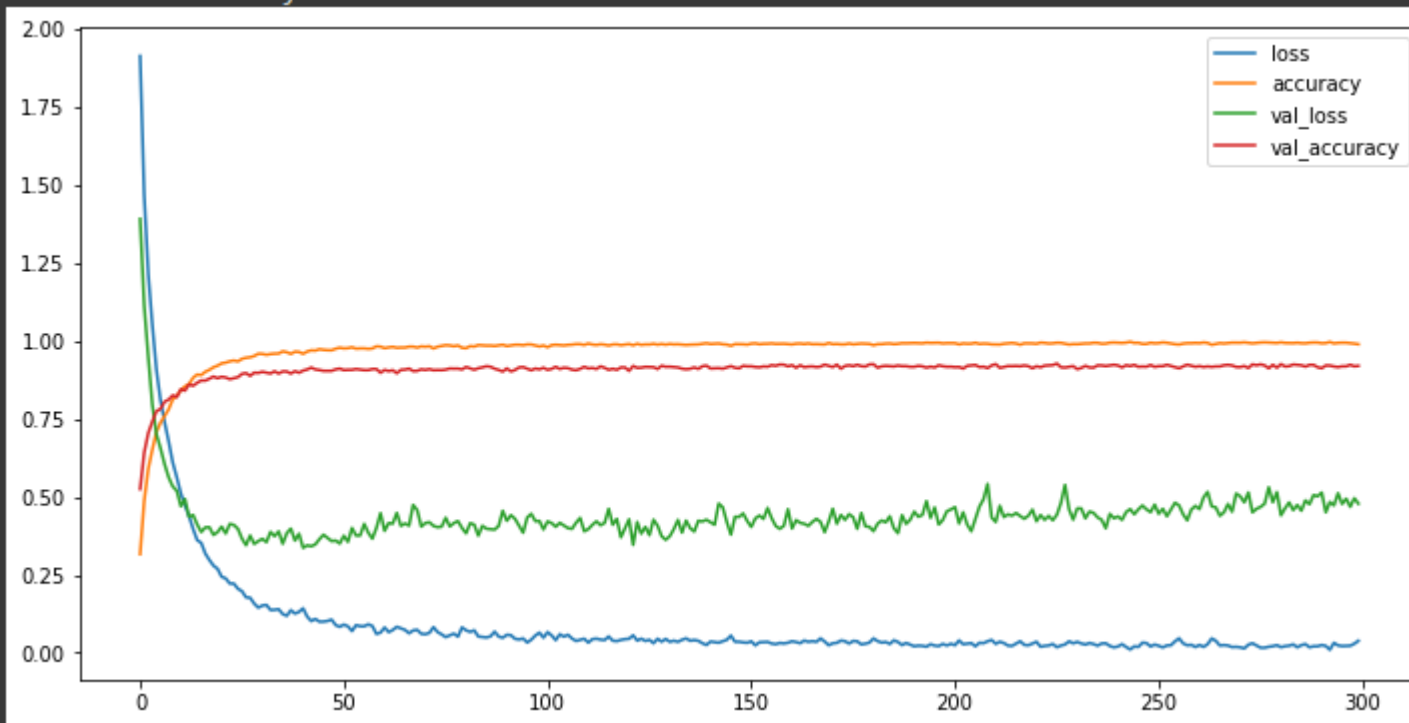
Using CNN

Implementation – Model Architecture

- Our model is fully connected , which means each node is connected to every other node in the next layer. At each layer, we applied a ReLU activation function to the output of each node, following the formula: $\text{ReLU}(x) = \{ x, x \geq 0 \text{ and } 0, x < 0 \}$
- Dropout randomly selects features to drop based off a specified constant .It is used to prevent overfitting
- We used Adam optimizer with learning rate 0.001 and 600 epochs to improve our results
- The model accuracy could further be improved by increasing the number of epochs. After a certain value though, we might achieve a threshold. So you increase the value accordingly
- The accuracy we manage on the validation is 92%, which is really satisfying

Using CNN

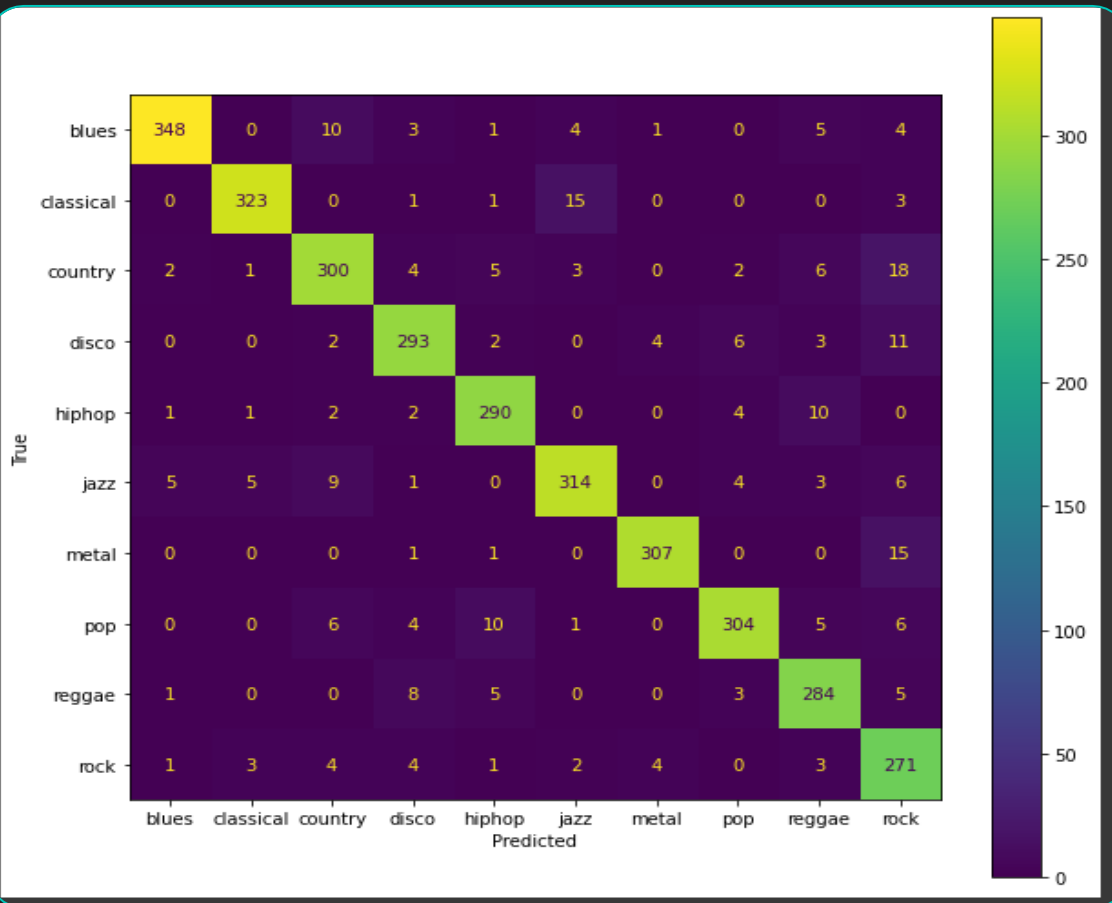
Results



Observing the plot, we understand that the model converges. The validation and train accuracy becomes similar after approximately 25 epochs. After that we have overfit because validation loss increases slightly.

Using CNN

Results



Confusion matrix

Elaborating on the diagrams, we see that the second model predicts the classes more accurately than the first model. The diagonal has elements with greater values. For example, Blues genre has very few false negatives (aka incorrect predictions) and many more correct predictions (diagonal). The diagrams justify our greater accuracy score and help us understand the efficacy of our convolution neural network.

Conclusion

- In our project , we experimented with two great technologies . Long-Term-Short Memory and Convolutional neural networks. Even though both produced good results, CNN was the more efficient (92% accuracy on the validation set).
- Music Genre classification was a great way to improve our Natural Language Processing skills in the tensorflow's Keras environment. Also , we experimented on real world data and we managed to pull off a very demanding task , which is to classify a very large dataset of songs to music genres. It would have been much more difficult for a human to do that amount of work .
- The difficulty of our project lied on three aspects:
 - Preparing the data to pass them to the model and deciding models' architecture.
 - Choosing the right technology for our model(Other tech : SVM , K-NN , K-Means)
 - Tweaking the model's parameters for better results

Sources

- There are links to the sources that we used in our jupyter notebook file.

Workload segmetation :

Aris' work was based on creating the CNN model and Giorgos' job was to create the LSTM model. We both contributed to visualization of the results.

Thank you !