Final Report: Classifying Music Subgenres Using Neural Networks

Coauthor
Sam Schrader
saschra@siue.edu

Coauthor
Daniel Tucek
dtucek@siue.edu

Coauthor
Rohith Kumar Sura
rosura@siue.edu

Coauthor
Ryan Cafarelli
rcafare@siue.edu

Abstract

Abstract. Music genre classification is an important component for the music information retrieval system. The problem of automatically identifying the subgenre of a given piece of music can be a lot more difficult to get high accuracy. One of the applications of solving this problem could be to suggest what subgenre tags musicians could use when they upload a track. Another application of this algorithm could be in recommendation systems which suggest music based on user preference. There are two important components to be considered for better genre classification, which are audio feature extraction and classifier model. In this project, for preprocessing, MFCCs and spectrograms were used for obtaining feature vectors for the classifiers from the EDM dataset [6]. Two classifiers, CNN and RNN-LSTM were trained and used to classify, each giving different accuracy in prediction. An important contribution of this work is the comparison of classification results of CNN and RNN-LSTM. CNN achieved test accuracy of 28% with a validation accuracy of 29%. RNN-LSTM achieved test accuracy of 54% with validation and 56.3% without.

Key words: music subgenre**,** EDM, classification, spectrograms, MFCCs, CNN, RNN-LSTM, testing, validation, accuracy, confusion matrix

1 Introduction

While there has been a fair amount of research into automatic classification of music genres [1, 2, 3, 4], less has been done with subgenre classification. It is a more difficult problem, as subgenres tend to have more common elements and overlap, leading to ambiguity in classification even among human experts. For our project we are tackling the problem of automatic classification of EDM subgenres by extending the results of Caparrini et al. to use neural networks on their dataset. We take raw audio files of EDM songs and process them into spectrograms and MFCCs, which are then fed into CNN and RNN-LSTM models; these models then output a classification label indicating the subgenre. The overall process flow is shown in Figure 1.

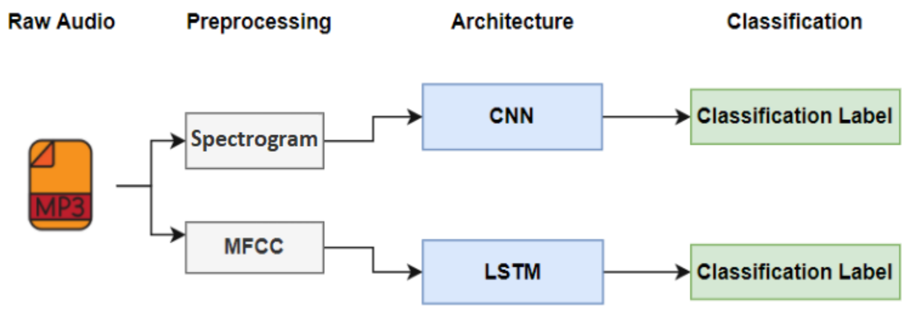


Figure 1: Process flow for EDM subgenre classification project.

2 Related Work

A number of different approaches have been taken to classify music by genre and subgenre. The project focused on the work of five of these approaches. In [1], the author gives the details of the genre classification using Support Vector Machine (SVM) and their comparison in terms of accuracy with other classifiers like decision trees, multilayer perceptrons, simple logistic, random tree, and Naive Bayes. The proposed classifier gives an accuracy of 92% when classifying 5 musical genres which are higher compared to other classifiers used in this experiment. In [5], a classifier was designed for jazz subgenre classification using an LSTM layer as the core and measured the performance of their new classifier with respect to the three techniques MLP, SVM, and KNN. They used Mel Frequency Cepstral Coefficients (MFCCs) for feature extraction of each song. The LSTM network achieved an accuracy of 80.30% whereas the MLP, SVM, and KNN achieved a maximum accuracy of 79.39%, 81.67%, and 77.43% respectively. Additionally, they added a MLP network before the LSTM layer that boosted the accuracy to 89.824%. In [2], the authors used a 5-layer Independently RNN with scattering coefficient for preprocessing the data to complete the music genre classification. The experimental results showed an excellent performance on the GTZAN dataset in terms of classification accuracy and training time. LSTM accuracy reaches a high accuracy of 97%, however, its training time per iteration is as long as 0.68s. IndRNN performed best with both comprehensive training time of 0.23s and classification accuracy of 96%. Compared to the state-of-the-art models, experimental results are very competitive. In [3], images of spectrograms were generated from time-slices of songs and used as the input to the neural network to classify the songs into their respective musical genres. Initially, they trained using 27 genres with no change in architecture and a 47% accuracy observed. Later, they trained on 7 genres with a change in activation function from ELU to ReLU where the accuracy improved to 67%. In [4], the authors examined the performance of convolutional neural networks (CNN) and recurrent neural networks (RNN). For pre-processing, they tested a Fast-Fourier Transform, a mel-spectrogram and mel-frequency cepstral coefficients. The combination of the GTZAN dataset, mel-spectrograms, and convolutional neural networks yielded the best result. They observed accuracy of 59% with CNN and 33% with RNN. Although the results fell behind the state-of-the-art performance, they outperformed other music classification studies that use a CNN by a large margin. Many of the papers implemented CNN algorithms and Machine Learning techniques, including k-NN, SVMs, LSTMs for music genre classification. We decided to implement LSTM and CNN to determine which approach provides the better accuracy.

3 Materials and Methods

3.1 Datasets and Features

Our primary dataset comes from [6], in which the authors investigate the use of tree-based classifiers in determining the subgenre of an EDM track. They prepared two datasets; we are currently utilizing Set 1, which consists of the top 100 tracks in each of the 23 subgenres on Beatport [17] as of November 29, 2016. Likely due to space and/or intellectual property concerns, the authors did not provide the raw audio for each track, instead making available the extracted audio features for each track in the form of a CSV file. Tracks can be uniquely identified in Beatport's database using a numeric ID, which is given in the files for Set 1 but not Set 2. We wanted to be able to extract new features from the tracks if necessary, so a VBA macro was written for Excel which parses the HTML document for a track on Beatport, locates the URL for the audio sample stream, and downloads it. These sample streams are typically 2 minutes of 96 kbps audio in MP3 format, although they can occasionally be of shorter duration. We were able to retrieve audio samples for 2,258 out of the 2,300 tracks in Set 1; we have more than 97 out of 100 tracks for most subgenres, with Hip-Hop being the least represented at 93. Contact was made with the authors of [6] to try to acquire the track IDs for Set 2, which would afford us another 2,900 tracks from Beatport's 2018 charts; we are still in the process of getting this information. Our VBA macro will make it straightforward to pull more audio from Beatport if we deem it necessary.

As detailed in subsequent sections, we chose to investigate two types of neural net: LSTMs and CNNs. LSTMs are often used for applications involving sequential data; however, in general it is not trivial to train them on overly long sequences (typically less than a few hundred time steps). The raw audio data we've collected has a sample rate of 44,100 Hz (samples per second), so a few hundred samples is on the order of milliseconds and likely wouldn't be useful for genre classification. Thus, for the LSTM so far, we have preprocessed the audio data into MFCCs (mel-frequency cepstral coefficients). MFCCs have been used successfully in other genre classification systems such as [6]. They are derived by mapping the Fourier transform (frequency spectrum) of a signal to the mel scale (which expresses frequencies logarithmically, more similar to our own perception of pitch), followed by performing a discrete cosine transform on the logarithm of the result. Much like generating a spectrogram, we can find the MFCCs for successive windowed segments of an audio signal to generate short sequences which still encapsulate a lot of audio information. There are two major parameters that can be adjusted when generating MFCCs which affect the resolution of the data: the number of audio segments (time resolution) and the number of coefficients (frequency resolution).

3.2 Methods

3.2.1 CNN

Convolutional Neural Networks (CNNs) are a type of neural network that are typically used for image recognition. Each layer of a CNN extracts features from an image by convolving (performing a mathematical operation of the data) a subset of the image onto the next output layer. After each convolutional layer many models will format and shrink the data using normalization, activation and pooling layers. These layers together help the model to learn higher level details about sections of the input data rather than focusing on individual pixels. A typical CNN incorporates multiple sets of theses layers before flattening the data into a fully connected layer to classify the overall image.

For music classification the audio data must be translated into an image to be fed into a CNN. The typical format for this image is called a spectrogram. A spectrogram is a visual representation of the frequencies of an audio signal over time. Figure 2 shows the spectrogram as a heat map calculated from an audio signal over time translated by a Fourier transform into the frequency domain.

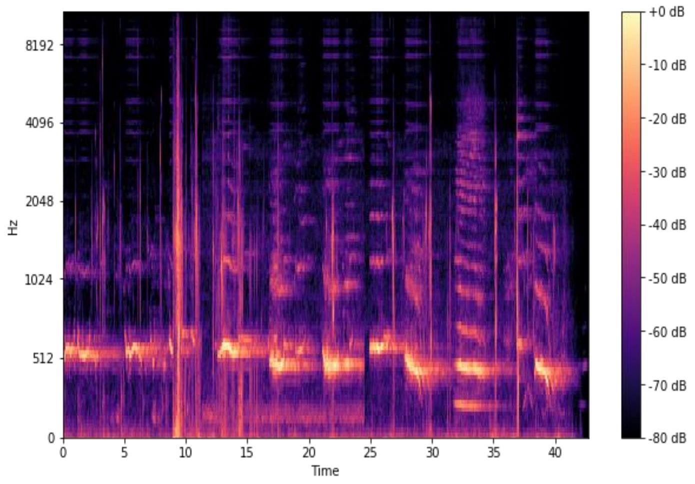


Figure 2: Spectrogram of a raw audio file.

The choice of hyperparameters for this model (learning rate, number of epochs, batch size and steps per epoch) were influenced by the limitations of Google Colab. The number of steps was defined as a function of the batch size (input size/batch size). The learning rate was influenced by the batch size and number of epochs.

3.2.2 RNN-LSTM

The other method being attempted is the recurrent neural network (RNN) with long short-term memory (LSTM). LSTMs are helpful with classifying time series data. They are able to detect patterns with longer dependencies in comparison to regular RNNs. This is because an LSTM stores information from multiple previous iterations while an RNN only stores the information from the previous iteration. A LSTM performs computations using an activation and recurrent activation function that allows it to forget and add information to its memory. Our LSTM model used a tanh activation function and a sigmoid recurrent activation function. The input for the network used MFCCs, discussed earlier, which were extracted from the data. Each song was processed into an audio time series for a certain duration. Each time series was divided up into the same number of segments to calculate the MFCCs for every segment.

The RNN-LSTM baseline model has several hyperparameters: learning rate was 0.001, batch size was 32, and 30 epochs were used. Attempting to lower the batch size led to the same accuracy but it took a longer amount of time since it's not exploiting vectorization as much, while increasing it gave lower accuracy. So, keeping the batch size close to 32 was optimal. Then, for the number of epochs, Figure 3 shows the loss/accuracy for each epoch for the training and validation set. Figure 3 shows that the model didn't have a trend to increase in loss or decrease in accuracy meaning the number of epochs didn't need to be changed. It could be decreased because the validation set usually started flat-lining, but when changes are made to the model having slightly more epochs might be beneficial.

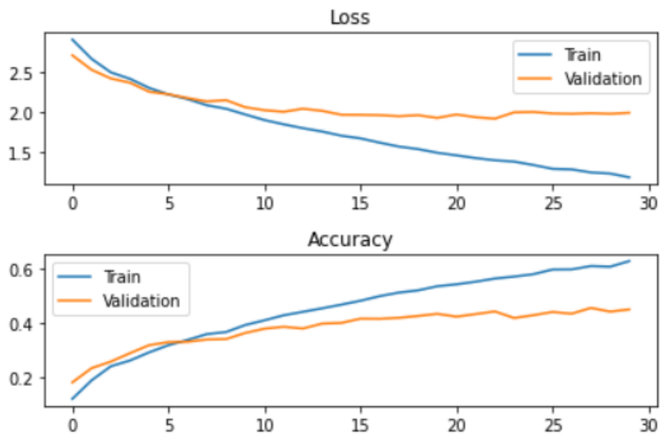


Figure 3: Training vs. validation loss (top) and accuracy (bottom) for the baseline LSTM model. The x-axis is the number of epochs, and the y-axis is the measurement.

The data was split by using 80% for the training set and 20% for the testing set. If a validation set was used, then 20% of the training set was used for the validation instead, resulting in a 64/16/20 split overall. This was chosen mostly because of convention; the other example models found had similar splits. The main metric used was accuracy. Additionally, precision and recall were calculated. Since this is a classification problem, the AUC and confusion matrix were also found. If a validation set was used, then the loss and accuracy were plotted, for each epoch, to compare the train and validation values.

4 Results/Discussion

4.1 CNN

For our project we used a CNN produced for genre classification of music that was used to try and classify subgenres of electronic dance music. The CNN was broken into groupings of layers. Each group consisted of a convolutional layer, batch normalization, relu activation layer and a max pooling layer. The groupings correspond to the filter size of the convolutional layers (8, 16, 32, 64, 128, and 256). After the groupings the data is flattened and a dropout of 30% is applied before using a softmax function to classify the image into the 23 subgenre categories.

Each track was broken into ten five second intervals then each interval was translated into a spectrogram with image size 360 x 360. For each subgenre, the spectrograms were shuffled then separated into training, validation, and testing sets (70%, 10% and 20%). The model was trained for 15 epochs with a learning rate of 0.001 and batch size of 100. The hyperparameters were chosen due to memory limitations when running the model, this required a smaller batch size which led to a smaller learning rate being applied to each batch. A secondary limitation was discovered when using Google Colab which was the cell execution time of 12 hours. Due to the size and number of the images in the training set the model exceeded this time constraint. Typical results achieved on Colab are described in Table 1; the average accuracy was around 28%

Interestingly, the model tended to perform better on a dedicated local machine. One of the best runs of this experiment produced a max of 45% validation accuracy and a final validation accuracy of 42%. The testing accuracy was 45% with a loss of 2.23. The training vs. validation accuracy and loss are shown in Figure 4. The testing precision was 46% with a recall of 54% and a ROC AUC of 72%. The confusion matrix heatmap, shown in Figure 5, shows that the model has an emphasis toward index 11 corresponding to the subgenre Hard Dance. Other runs would occasionally have bias towards predicting index 6 (Electro House) or other subgenres. This may be due to sensitivity of the model to uneven subgenre representation as the train/val/test splits are not stratified; this could be resolved by increasing the size of our dataset. It may also be due to overfitting during training.

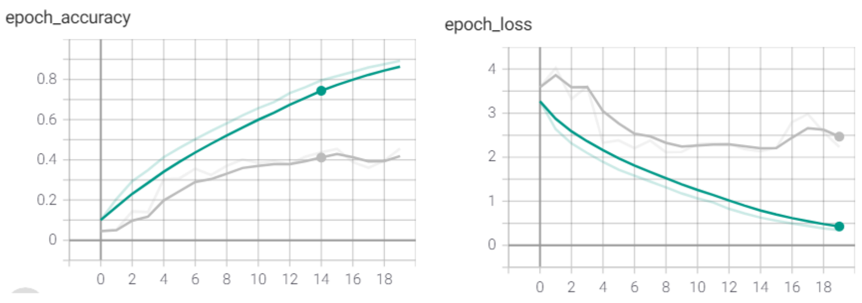
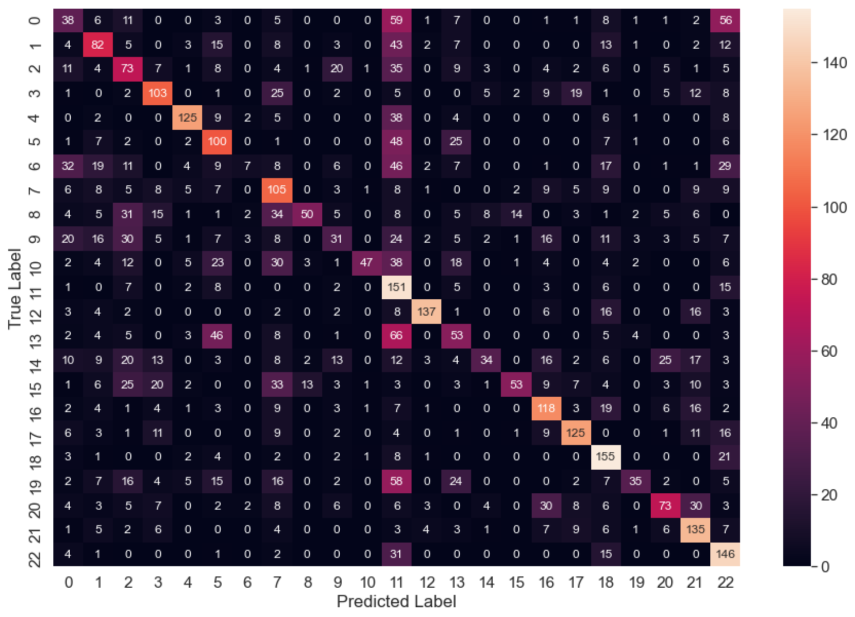


Figure 4: Training vs. validation accuracy (left) and loss (right) for the CNN.



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
0	38	6	11	0	0	3	0	5	0	0	0	59	1	7	0	0	1	1	8	1	1	2	56
1	4	82	5	0	3	15	0	8	0	3	0	43	2	7	0	0	0	0	13	1	0	2	12
2	11	4	73	7	1	8	0	4	1	20	1	35	0	9	3	0	4	2	6	0	5	1	5
3	1	0	2	103	0	1	0	25	0	2	0	5	0	0	5	2	9	19	1	0	5	12	8
4	0	2	0	0	125	9	2	5	0	0	0	38	0	4	0	0	0	0	6	1	0	0	8
5	1	7	2	0	2	100	0	1	0	0	0	48	0	25	0	0	0	0	7	1	0	0	6
6	32	19	11	0	4	9	7	8	0	6	0	46	2	7	0	0	1	0	17	0	1	1	29
7	6	8	5	8	5	7	0	105	0	3	1	8	1	0	0	2	9	5	9	0	0	9	9
8	4	5	31	15	1	1	2	34	50	5	0	8	0	5	8	14	0	3	1	2	5	6	0
9	20	16	30	5	1	7	3	8	0	31	0	24	2	5	2	1	16	0	11	3	3	5	7
10	2	4	12	0	5	23	0	30	3	1	47	38	0	18	0	1	4	0	4	2	0	0	6
11	1	0	7	0	2	8	0	0	0	2	0	151	0	5	0	0	3	0	6	0	0	0	15
12	3	4	2	0	0	0	0	2	0	2	0	8	137	1	0	0	6	0	16	0	0	16	3
13	2	4	5	0	3	46	0	8	0	1	0	66	0	53	0	0	0	0	5	4	0	0	3
14	10	9	20	13	0	3	0	8	2	13	0	12	3	4	34	0	16	2	6	0	25	17	3
15	1	6	25	20	2	0	0	33	13	3	1	3	0	3	1	53	9	7	4	0	3	10	3
16	2	4	1	4	1	3	0	9	0	3	1	7	1	0	0	0	118	3	19	0	6	16	2
17	6	3	1	11	0	0	0	9	0	2	0	4	0	1	0	1	9	125	0	0	1	11	16
18	3	1	0	0	2	4	0	2	0	2	1	8	1	0	0	0	0	0	155	0	0	0	21
19	2	7	16	4	5	15	0	16	0	2	0	58	0	24	0	0	0	2	7	35	2	0	5
20	4	3	5	7	0	2	2	8	0	6	0	6	3	0	4	0	30	8	6	0	73	30	3
21	1	5	2	6	0	0	0	4	0	0	0	3	4	3	1	0	7	9	6	1	6	135	7
22	4	1	0	0	0	1	0	2	0	0	0	31	0	0	0	0	0	0	15	0	0	0	146

Figure 5: Heatmap of the confusion matrix for a high-performing run of the CNN model.

The next steps for this approach were to experiment with the model structure, hyperparameters, image size, and corresponding track segment duration. The results of the successful experiments are shown in Table 1. A failed experiment was conducted using the full track length which led to overfitting of the model, this was likely due to a large number of model parameters compared to the size of the dataset. Another failed experiment was conducted using image sizes of 720 x 720. Due to the image size the memory usage exceeded the limitations of Google Colab as well as the cell execution time limitation.

Table 1: Results attempting different changes to the CNN baseline model.

	Accuracy			Experiment
Image Size	Training	Validation	Testing	Default: (Epoch: 20, Learning Rate: 0.001, Dropout 30%, Batch Size: 100)
360x360	0.88	0.29	0.28	Baseline
360x360	0.688	0.157		L2 Regularization
216x216	0.822	0.214	0.227	Baseline
216x216	0.741	0.089		Epochs: 15
216x216	0.039	0.044		Epochs: 15, Learning Rate: 0.1
216x216	0.673	0.098		Epochs: 15, Learning Rate: 0.01
216x216	0.744	0.054		Epochs: 15, Batch Size: 200
216x216	0.777	0.286	0.278	Epochs: 15, Batch Size: 50
216x216	0.636	0.177		Epochs: 15, Batch Size: 50, Dropout: 50%
216x216	0.513	0.167		Epochs: 15, L2 Regularization

The experiments conducted highlight a number of details about the overarching model. One interesting detail is that regardless of the input size the L2 regularization had a negative effect on the model accuracy, this may be attributed to overfitting. Another important observation was that decreasing the image size had a meaningful impact on overall accuracy; however, the much lower solve time allowed for many additional experiments which ended up producing comparable results. Learning rate was another parameter that had a great impact on accuracy. As the learning rate increased the accuracy decreased likely due to wild swings in weights due to the large swing during each batch. Finally, the batch size, in addition to a low learning rate, had a great impact on the overall accuracy. This was likely due to the model having many opportunities (313 steps per epoch) to make small tweaks to the weights.

4.2 RNN-LSTM

The starting baseline model used for the LSTM network had five layers: two LSTM, dense, dropout, and dense respectively. The two LSTM layers used a tanh activation function, sigmoid recurrent activation function, and has 64 units for the output. The first dense layer used the relu activation function with 64 units for output. The dropout layer is used during training to attempt to stop overfitting, and it was set to drop 30% of the input. The dropout layer used 64 units for output. The output dense layer has an output of 23 which corresponds with the 23 different subgenres in the dataset, and it uses a softmax activation function.

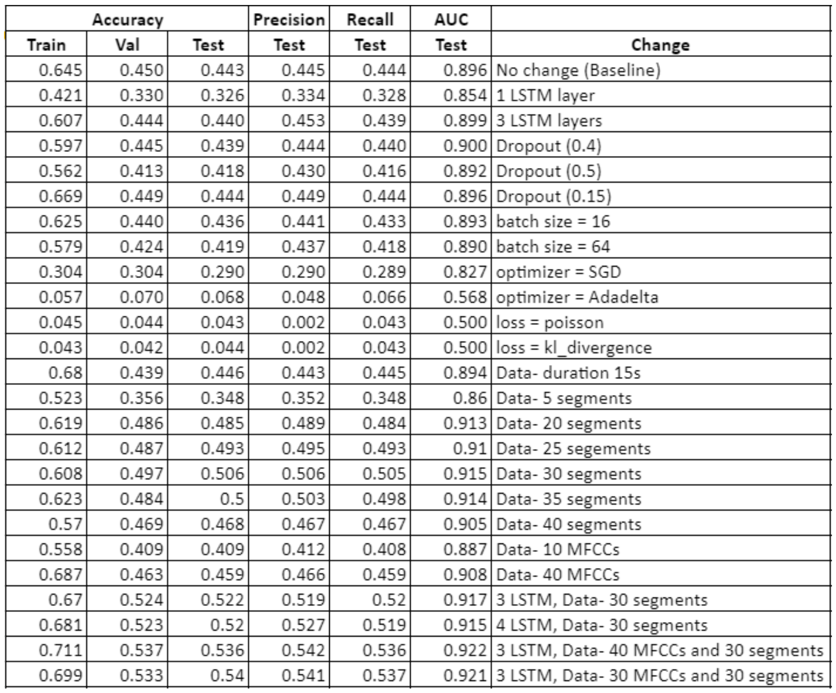
The baseline model was able to achieve a test accuracy of 44.3% with validation and 47.4% without. The data was preprocessed to be 30 second track durations, 10 segments for each track duration, and 20 MFCCs for each segment. So, the first 30 seconds of each MP3 file was used to process the data. Table 2 shows the different changes that were made to the model and the results. Most changes to the model itself produced about the same results or worse. The main increase in accuracy from the baseline model came from preprocessing the data differently.

The final LSTM model was able to achieve a test accuracy of 54% with validation and 56.3% without. The data was preprocessed to be 30 second track durations, 30 segments for each track duration, and 30 MFCCs for each segment. Also, this model had three LSTM layers instead of two. Everything else about the model was the same from the baseline. Figure 6 shows a heatmap of a confusion matrix based on the final model.

An interesting observation for the confusion matrix, Figure 6, is that labels 0 (Big Room) and 6 (Electro House) have the lowest accuracy and have a tendency to predict each other. This could mean that the subgenres are very similar to each other and don't have enough distinguishing characteristics to be separated into two different subgenres. Additionally, label 6 (Electro House) seems to be an issue with both the CNN and LSTM model which could mean Electro House is too similar to multiple subgenres.

Overall, the main contributor to increasing the LSTM model accuracy was from preprocessing the data differently. Improving the quality of the data is what allowed the model to show its true performance. Basically, the saying of "garbage in, garbage out" can be applied here. The only change in the model that made some positive impact was adding a third LSTM layer after making the preprocessing changes which can be seen in Table 2.

Table 2: Results attempting different changes to the LSTM baseline model.



Accuracy			Precision	Recall	AUC	
Train	Val	Test	Test	Test	Test	Change
0.645	0.450	0.443	0.445	0.444	0.896	No change (Baseline)
0.421	0.330	0.326	0.334	0.328	0.854	1 LSTM layer
0.607	0.444	0.440	0.453	0.439	0.899	3 LSTM layers
0.597	0.445	0.439	0.444	0.440	0.900	Dropout (0.4)
0.562	0.413	0.418	0.430	0.416	0.892	Dropout (0.5)
0.669	0.449	0.444	0.449	0.444	0.896	Dropout (0.15)
0.625	0.440	0.436	0.441	0.433	0.893	batch size = 16
0.579	0.424	0.419	0.437	0.418	0.890	batch size = 64
0.304	0.304	0.290	0.290	0.289	0.827	optimizer = SGD
0.057	0.070	0.068	0.048	0.066	0.568	optimizer = Adadelta
0.045	0.044	0.043	0.002	0.043	0.500	loss = poisson
0.043	0.042	0.044	0.002	0.043	0.500	loss = kl_divergence
0.68	0.439	0.446	0.443	0.445	0.894	Data- duration 15s
0.523	0.356	0.348	0.352	0.348	0.86	Data- 5 segments
0.619	0.486	0.485	0.489	0.484	0.913	Data- 20 segments
0.612	0.487	0.493	0.495	0.493	0.91	Data- 25 segements
0.608	0.497	0.506	0.506	0.505	0.915	Data- 30 segments
0.623	0.484	0.5	0.503	0.498	0.914	Data- 35 segments
0.57	0.469	0.468	0.467	0.467	0.905	Data- 40 segments
0.558	0.409	0.409	0.412	0.408	0.887	Data- 10 MFCCs
0.687	0.463	0.459	0.466	0.459	0.908	Data- 40 MFCCs
0.67	0.524	0.522	0.519	0.52	0.917	3 LSTM, Data- 30 segments
0.681	0.523	0.52	0.527	0.519	0.915	4 LSTM, Data- 30 segments
0.711	0.537	0.536	0.542	0.536	0.922	3 LSTM, Data- 40 MFCCs and 30 segments
0.699	0.533	0.54	0.541	0.537	0.921	3 LSTM, Data- 30 MFCCs and 30 segments

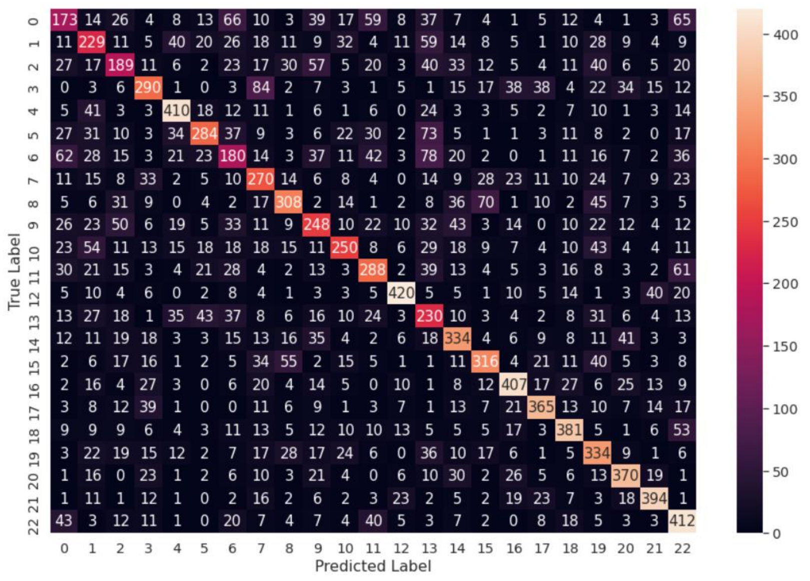


Figure 6: Heatmap of the confusion matrix for the final LSTM model.

5 Conclusion/Future Work

Our work here provides a good starting point for further investigation of using neural networks to automatically classify audio based on subgenre. While we did not achieve accuracy as high as the tree-based classifiers used in [6], both the RNN-LSTM and CNN models show promise in being used for this purpose. The LSTM performed better overall, with a test accuracy of 54% using a 64/16/20 train/validation/test split, while the CNN achieved a test accuracy of 28% with a 70/10/20 split (although it saw accuracies as high as 45% during some runs). The LSTM model was improved over the baseline by adding an additional LSTM layer and increasing the time and frequency resolution of the preprocessed MFCC series data. The CNN model was more resistant to changes, and was prone to overfitting on the training data.

There is likely still room for improving the CNN via the model architecture as well as changes to the data preprocessing; higher compute power and more memory would help as well, as many experiments attempting to increase data resolution were aborted due to Colab's space/time constraints. It also seems that there may be performance and precision losses on Colab compared to a dedicated machine [7], even when running identical experiments, although we did not ascertain if these differences were statistically significant. Further improvements in overall accuracy could be made by creating an ensemble model, possibly using both the CNN and LSTM in tandem. We would also like to try to better capture large-scale features across minutes of audio, rather than just across smaller sections of a few seconds. Another possibility is using unsupervised learning to try to find clusters which could be used to further fine-tune the models.

Ultimately however, there is a limit to the accuracy that can be achieved by a system with the goal of music subgenre classification. Musical genre taxonomies (and classification schemes of creative work in general) are inherently ambiguous, and this lack of clear definition or distinction between classes grows as one follows the branches of the tree down. There may also be issues with using Beatport's genre tags as ground truth for distinct subgenres, as these are based more on popularity than on inherent separability. For example, Big Room is actually considered to be a subgenre of Electro House, but was tagged distinctly on Beatport due to its relative popularity at the time the data was collected; treating these as two distinct classes may be confusing to certain learning models. Future models would likely benefit from using unsupervised learning to determine which classes are more closely related, thus building their own hierarchy of subgenres.

References

[1] G. Mali and S. P. Mahajan, "Audio Genre Classification Employing Support Vector Machine," 2018 4th International Conference for Convergence in Technology (I2CT), Mangalore, India, 2018, pp. 1-5, doi: 10.1109/I2CT42659.2018.9058184.

[2] W. Wu, F. Han, G. Song and Z. Wang, "Music Genre Classification Using Independent Recurrent Neural Network," 2018 Chinese Automation Congress (CAC), Xi'an, China, 2018, pp. 192-195, doi: 10.1109/CAC.2018.8623623.

[3] N. Pelchat and C. M. Gelowitz, "Neural Network Music Genre Classification," 2019 IEEE Canadian Conference of Electrical and Computer Engineering (CCECE), Edmonton, AB, Canada, 2019, pp. 1-4, doi: 10.1109/CCECE.2019.8861555.

[4] Yang, J. "Music Genre Classification with Neural Networks: An Examination of Several Impactful Variables." (2018).

[5] R. J. M. Quinto, R. O. Atienza and N. M. C. Tiglao, "Jazz music sub-genre classification using deep learning," TENCON 2017 - 2017 IEEE Region 10 Conference, Penang, Malaysia, 2017, pp. 3111-3116, doi: 10.1109/TENCON.2017.8228396.

[6] Antonio Caparrini, Javier Arroyo, Laura Pérez-Molina & Jaime SánchezHernández (2020): Automatic subgenre classification in an electronic dance music taxonomy, Journal of New Music Research, DOI: 10.1080/09298215.2020.1761399

[7] The Tensorflow Authors. "Mixed precision," Tensorflow documentation on Github, 22 Jan. 2021. Available from: https://github.com/tensorflow/docs/blob/master/site/en/guide/mixed_precision.ipynb.

[8] Harris, Charles R., et al. "Array Programming with NumPy." *Nature*, vol. 585, no. 7825, 2020, pp. 357–362., doi:10.1038/s41586-020-2649-2.

[9] Virtanen, Pauli, et al. "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python." *Nature Methods*, vol. 17, no. 3, 2020, pp. 261–272., doi:10.1038/s41592-019-0686-2.

[10] Brian McFee, et al. librosa/librosa: 0.8.0. 0.8.0, Zenodo, 22 July 2020, doi:10.5281/zenodo.3955228. Software available from: https://librosa.org/.

[11] Martín Abadi, et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from: https://tensorflow.org/.

[12] Chollet, F., et al. Keras. GitHub. Software available from: https://github.com/fchollet/keras.

[13] Reback, Jeff, et al. Pandas 1.2.4. v1.2.4, Zenodo, 2021, doi:10.5281/ZENODO.4681666. Software available from: https://pandas.pydata.org/pandas-docs/stable/index.html.

[14] Robert, J., Webbie, M. & others, 2018. Pydub v0.25.1, GitHub. Software available from: http://pydub.com/.

[15] Thomas A Caswell, et al. matplotlib/matplotlib: REL: v3.4.1. v3.4.1, Zenodo, 31 Mar. 2021, doi:10.5281/zenodo.4649959. Software available from: https://matplotlib.org/stable/index.html.

[16] Carrera, Ero, et al. Pydot v1.4.2, Feb. 15, 2021. Software available from: https://github.com/pydot/pydot.

[17] Beatport, https://www.beatport.com/.

Contributions

Sam Schrader: Various coding / data wrangling, report and presentation writing

Daniel Tucek: Primary investigator for the RNN-LSTM model, report and presentation writing

Rohith Kumar Sura: Research, graphics, report and presentation writing

Ryan Cafarelli: Primary investigator for the CNN model, report and presentation writing