Classifying Audio Music Genres Using CNN and RNN

Ahmed A. Khamees[1], Hani D. Hejazi[1], Muhammad Alshurideh[2,3], and Said A. Salloum[4,5](✉)

[1] Faculty of Engineering and IT, The British University in Dubai, Dubai, UAE
[2] University of Sharjah, Sharjah, UAE
[3] Faculty of Business, University of Jordan, Amman, Jordan
[4] Machine Learning and NLP Research Group, Department of Computer Science, University of Sharjah, Sharjah, UAE
ssalloum@sharjah.ac.ae
[5] School of Science, Engineering, and Environment, University of Salford, Salford, UK

Abstract. This paper discusses applying different types of neural networks to classify a dataset of type audio. We used a GTZAN dataset that includes various audio music records representing different conventional categories of music genres. Each shares a set of common traditions; these traditions we call features. We build our proposed Python models using the Anaconda toolkit with TensorFlow (TF) an open-source deep-learning library. In our previous research, we build a multilayer sequential model to classify the dataset and then solve the overfitting issue in that model. In this paper, we build a Convolutional Neural Network (CNN) and Recurrent Neural Network (RNN) with Long Short Time Memory (LSTM). Finally, we compared the results to know the capabilities and limitations of Deep Learning (DL). CNN outperformed the other models in terms of training and test accuracy, having 83.74% and 74%, respectively.

Keywords: Text Generation word2vec · Arabic poems · Music genresVecword2vec · Long Short Time Memory (LSTM)

1 Introduction

In 2012 NN has been discussed thoroughly in many other deep learning capabilities, e.g., Long Short Term Memory (LSTM), Deep-forward Neural Networks (DNN), and Convolutional Neural Networks (CNN) [1–6]. Many solutions exist to produce music based on using computers, e.g., neural networks, Markov chains Recombinancy, sampling, morphing, and rule-based methodologies [2, 3, 7, 8].

In [9] explored generating music by utilizing a statistical machine language model. Where a unit variable selection is then concatenating these units based in their variable ranking, this model inspired by the text to speech (TTS) technique and natural language processing (NLP) to label the units within. The authors firstly developed an autoencoder to encode input of the music spectrum. Then, to predict the following unit, a deep structured semantic model (DSSM) combined with a long-short memory model (LSTM).

The learning mechanism minimizes the perplexity of selecting the next unit based on the preceding one(s). The model is evaluated by calculating the cost based on rank accuracy, mean and subjective listening by some music experts to test the likability and naturalness of the output music. Moreover, the model was compared with another baseline system using trained LSTM to predict one subsequent unit.

In [10] studied music transcription composition and modeling by applying deep learning methods like LSTM networks. LSTM consists of three hidden layers with 512 blocks in every layer. LSTM trained and built using around 23 thousand musical transcriptions expressed with ABC notation and generated new transcriptions. Such transcriptions will be useful in some practical music contexts composition. Two approaches were conducted to train the model, i.e., character-based and token-based.

A neural network NN is a processing model of deep learning that consists of input and output layers that include several hidden layers; these layers consist of neurons (units), where input is transformed through them in a nonlinear process. RNN is an NN that processes the input from one unit to another close input in the next layer. However, deep RNN consists of several RNN layers where every hidden layer creates an output sequence used as input for deeper ones. Deeper the network architecture, it is expected to learn more complex input representations and their short and long-term relationship from each layer. RNN and LSTM proved efficient results when applied to domains where data is sequentially processed, e.g., recognizing speech and audio sounds [11].

Artificial neural networks (ANNs) can be learned to map audio features from low-level representation to a higher one. ANN is commonly used for regression and classification tasks such as instrument and genre classification, critical detecting, and identifying artists. [11] exploited the use of ANN for generating musical applications. The introduced model suggests reusing ANN autoencoder to synthesis musical audio. This can be accomplished using the ANN feedforward interactive system instead of regression or discriminative tasks. The proposed system implicitly trains ANN on the low-level representation of feature frames. However, high-level representation is learned explicitly through neural networks' autoencoding. This system allows direct interaction with model parameters and generates real-time musical audios. A machine learning library called Pylearn2 is used to train the proposed model [12], introduced WaveNet autoencoder, a new style model that applies an autoregressive type of decoding over learned raw waveform of audio. After that, the authors introduced the NSynth dataset, consists of high quality and large scale musical notes. This dataset demonstrated qualitative and quantitative improvements in WaveNet autoencoder performance over another type of well-tuned baseline autoencoder. Authors concluded that the proposed model learns from hidden representations allowing meaningful morphing and tones to playback within instruments that escalates in timbre creating new instrument sound types that are expressive and realistic using neural network model.

This paper is divided into eight sections. In the first one, we discussed some related work regarding applying different NN models over musical audio using a computer. Next, we discussed the classification models used in our research (i.e., CNN and RNN). Finally, in sections six through eight, we summarized models' results, conclusions, and limitations.

2 Methodology

In this paper, we used a preprocessed GTZAN[1] audio dataset. It consists of ten musical genres of total size 1 GB soundtracks split into 30 s audio files. These audio files are preprocessed before classification saved into Jason file that includes ten segments per track. Thus, we finally have 10000 segments to be split into training and test sets.

In order to get the best results, we have we noticed that the split of the dataset into (Training/Test 80/20) percentage gains better training accuracy and less testing error results than the split (Training/Test 20/80). Therefore, we will consider the first split of data (Training/Test 80/20) for the next classification process using CNN and RNN models.

3 Experiment

3.1 Implementing CNN with Max-Pooling for Music Genre Classification

CNN are special types of neural networks that are mainly used for image processing. They have been found to outperform the classical multilayer architecture, and they have fewer parameters. The difference in using CNN over the multilayer architecture is that processing images with CNN is applied on a structured data, so the pixels represent more than a simple dots but a more emergent structure like edges, shapes, and translation and scale invariance. Using CNN, we try to extract different basic features using convolutional or pooling components. In the convolution component, a filter is applied on the image as an overlay layer, and the convolutional value in the output image is the dot product of each value from the input image with the filter (kernel) layer. We slide the filter (kernel) layer all over the input image pixels. The kernel is a feature detector where different kernels detect different features like vertical or oblique lines; the network here, rather than finding these features, learns them.

When building CNN, we should look at different parameters to set up a CNN model (e.g., the grid size, stride, depth, and several filters). It is advised to use a grid size with odd numbers since it will have a central value as a reference in the middle. Strides are the number of steps in pixels to slide the kernel over the input image. For example, depth for a gray image equals one, but it is three for a colored image (three channels), a grid for each channel. The convolutional layer can have as many 2D arrays as the number of kernels.

The other component is pooling, it tries to downsampling the original image using max or average pooling functions, and pooling does not have any parameters. We build our CNN model using three CNN layers with Max-Pooling function with stride kernel (2×2), then the output of these layers is flattened into a dense layer. Here, we performed feature learning by extracting basic features, and while moving deeper into the network, it will leverage the features into a more meaningful abstract presentation like shapes.

[1] GTZAN dataset can be freely downloaded through the following link: https://www.kaggle.com/carlthome/gtzan-genre-collection/data.

For audio, we can use different visual pins of Time and Frequency axes in a 2D array that is quite similar to an image with pixel values as Amplitudes of the audio. To build a CNN model, we created three sets of data, e.g., training, validation, and testing sets. We split a validation dataset from the training dataset for tweaking the hyper-parameters to evaluate our model and see the effectiveness of the chosen setting. Therefore, we trained and evaluated CNN over the test set. Finally, we made predication on some samples.

Table 1. Accuracy and error results of applying the CNN model with Max-Pool function over 30 epochs.

Number of epochs = 30	Training accuracy	Testing accuracy	Training error	Testing error
(Training, Validation)/Test 80 (80, 20)/20	0.7455	0.6844	0.7253	0.8919

Table 1 shows the Accuracy and Error results for implementing a CNN model with max-pooling function over 50 epochs. We run the model over one dataset split, with ((Training, Validation)/Test 80 (80, 20)/20) split percentages. Results show promising results for both training and test Accuracy, which equals 74.55% and 68.44%, respectively. As well for training and testing, error equals 72.53% and 89.19%, respectively.

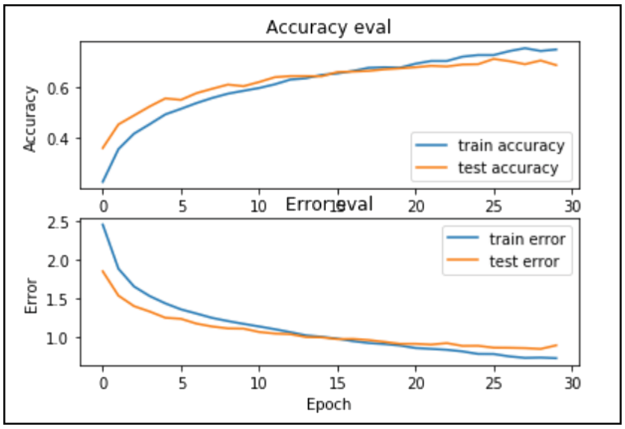

Fig. 1. Accuracy and error plot after applying the CNN model with Max-Pool function over 30 epochs.

In Fig. 1, for the accuracy plot, we can notice that the training and test sets tend to linearly increased then sort of stabilized until epoch number 15 with a slight increase for the training set afterward. The same thing is noticed for Error plot where testing error where Error rate tends to keep decreasing until epoch 15 where testing error sort of stabilize but training error slightly reduced more than the testing error. Next, we applied the same CNN model, but over 50 epochs are noticing if any improvements will occur.

Table 2. Accuracy and error results of applying the CNN model with Max-Pool function over 50 epochs.

Number of epochs = 50	Training accuracy	Testing accuracy	Training error	Testing error
(Training, Validation)/Test 80 (80,20)/20	0.8374	0.7400	0.4715	0.7658

Table 2 shows the Accuracy and Error results for implementing a CNN model with a max-pooling function over 50 epochs. We run the model over one dataset split, with ((Training, Validation)/Test 80 (80, 20)/20) split percentages. Results show promising results for both training and test Accuracy equals to 83.74% and 74%, respectively. As well for training and testing, error equals 47.15% and 76.58%, respectively.

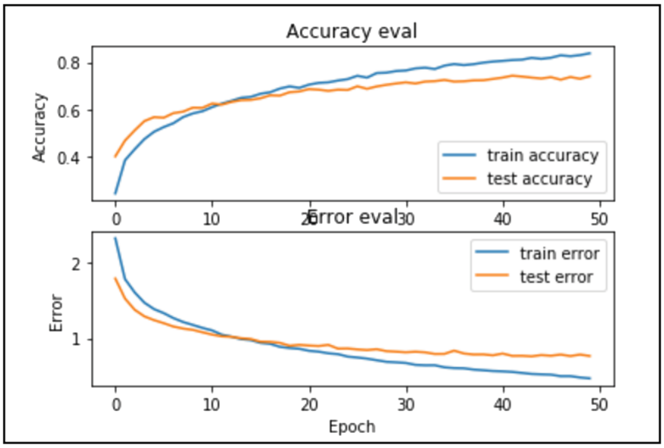

Fig. 2. Accuracy and error plot after applying the CNN model with Max-Pool function over 50 epochs.

In Fig. 2, for the accuracy plot, we can notice that the training and test sets tend to linearly increased then sort of stabilized until epoch number 11 with a slight increase for the training set afterward. The same thing is noticed for Error plot where testing error where Error rate tends to keep decreasing until epoch 13 where testing error sort of stabilize but training error slightly reduced more than the testing error.

In summary, Table 3 shows that applying the CNN model with the Max-Pool function over 50 epochs tend to perform better than the same model with 30 epochs in terms of accuracy and error for both training and testing percentages. This means the model learns better through iterating more epochs.

Table 3. Accuracy and error results of applying the CNN model with Max-Pool function over 30 and 50 epochs.

Number of epochs	Training accuracy	Testing accuracy	Training error	Testing error
30	0.7455	0.6844	0.7253	0.8919
50	0.8374	0.7400	0.4715	0.7658

3.2 Implementing RNN with LSTM for Music Genre Classification

With RNN, the order of data is essential, like in time series or a piece of music with a melody, where a melody can have a variable number of notes or data points. We want a network that is able to process data regardless number of notes or data points. So RNN tries to address these types of sequential data where each data point is processed in a context that looks back to know what to predict. Therefore, RNN is considered ideal for processing audio music because we can think of audio as a time series with ordered points over time.

A special type of time series is Univariate time series that means we have only one value that takes each interval. MFCCs are considered a multivariate time series. In our case for MFCC intervals, each interval has 13 values, and each value represents a coefficient to its MFCC. The relative dimension overall can be measured by the following equation [sample-rate/hope-length $\times$ 9, number of MFCC] so for our example, we have [22016/512 $\times$ 9, 13] that is equal to [387, 13], so we have 387 intervals each interval has 13 values. These time series are fed one data point at a time into our proposed network; this allows the network to predict the next step depending on the previous step given the current step in a specific context. Data points are fed into a recurrent layer as an inputs. Then, they are passed into a dense layer that process classification on these data points using Softmax function. The Shape of the input takes three parameters e.g. the batch size which is the number of samples in a window we are passing into a network, number of steps we have in a sequence, and number of different measures we have in a time series.

Calling a recurrent network is recursively using the same cell repeatedly in a sequential way. RNN solves Vaniting Gradient, which can explode when using ReLU layer because it is not bounded, unlike RNN, which has limited values between $[-1, 1]$ by using "tanh" function. After that, we learn the network using the backpropagation method that passes error through time by unrolling RNN steps considering each time step as a layer in a feedforward network—calculating the error and back-propagate it through the early stages using the sequence-to-sequence approach. By doing so, we stabilize the training process and speed up the training process. One issue with RNN is it cannot be used to look to the distant past. It has no long-term memory. It cannot learn long-term patterns with long dependencies like in long audio music type of data. RNN is not capable of understanding and makes a prediction that keeps track of the context. Therefore, the LSTM network a special successful type of RNN network. LSTMs can learn long-term patterns with up to 100 steps but start struggling to deal with more steps than that. As in the previous CNN model, we first prepared datasets (train, validation, and test sets). To build an LSTM model, we drop all previous CNN layers, and we add two LSTM layers instead and one Dense layer after that with the ReLU activation function. Then we also added a dropout layer to avoid the overfitting issue discussed above.

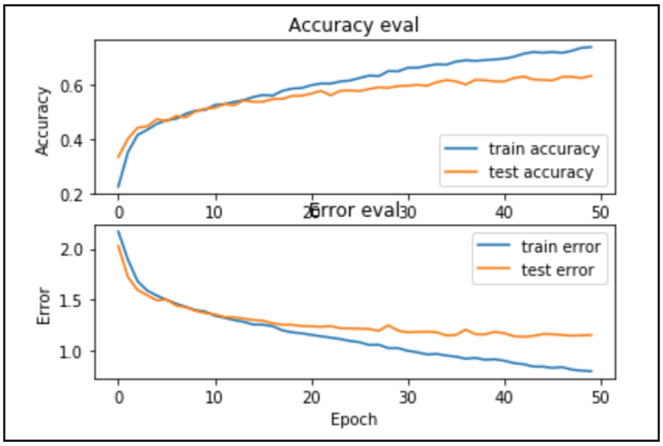

Fig. 3. Accuracy and error plot after applying the RNN model with LSTM.

In Fig. 3, for the accuracy plot, we can notice that the training and test sets tend to linearly increased then sort of stabilized until epoch number 8 with a slight increase for the training set afterward. The same thing is noticed for Error plot where testing error where Error rate tends to keep decreasing until epoch 10 where testing error sort of stabilize but training error slightly reduced more than the testing error.

Table 4. Accuracy and error results of applying the RNN model with LSTM.

Number of epochs = 50	Training accuracy	Testing accuracy	Training error	Testing error
(Training, Validation)/Test 80 (80,20)/20	0.7411	0.6344	0.7925	1.1487

Table 4 shows the Accuracy and Error results for implementing an RNN model with LSTM over 50 epochs. We run the model over one dataset split, with ((Training, Validation)/Test 80 (80, 20)/20) split percentages. Results show that relatively good results for both training and test Accuracy equal 74.11% and 63.44%, respectively. As well as for training and testing Error.

3.3 Models Summary

Table 5. Accuracy and error results over the different proposed models with (Training/Test 80/20) percentages.

Training/Test 80/20				
Evaluation metric	Model			
	Multi-sequential (three HL)	Multi-sequential with overfitting issue solved	CNN with Max-Pooling	RNN with LSTM
Training accuracy	0.9393	0.7424	0.8374	0.7411
Testing accuracy	0.5890	0.6055	0.7400	0.6344
Training error	0.1813	1.0073	0.4715	0.7925
Testing error	2.8012	1.6286	0.7658	1.1487

Table 5 shows the Accuracy and Error results for implementing the different proposed models with over 50 epochs. We run the model over one dataset split, with (Training/Test 80, 20) split percentages. Results show that our proposed CNN with Max-Pooling model tends to outperform over all the other proposed models. With the Max-Pooling model, CNN has the highest Testing Accuracy equals to 74% and has the least Testing Error equals to 76.58%.

4 Conclusion and Future Work

Using CNN with the Max-Pooling function outperformed both RNN with the LSTM model and the classical multi-sequential layer model with three hidden layers even after handling the overfitting issues. Our experiment is applied on a laptop with intel-core i7-8550U CPU@1.80 GHz 1.99 GHz, 8 GB RAM, and 64 bit windows 10 Pro Operating System. One drawback of CNN in comparison with the multilayer architecture is that CNN is expensive. We perform the model over 50 epochs; it took 18 s and 3 ms each, thus around 15 min. It took 1 s to perform each epoch in the multilayer model we have implemented above in a total of 50 s. Therefore, the proposed multilayer model (one input, three hidden layers, and one output) is faster 18 times the CNN model. We run RNN with the LSTM model over 50 epochs and take much longer to compile each epoch than any previous model. It took around 28 s to run each epoch with a total of 23.3 min that is 64% much slower than the CNN model and 28 times slower than the multi sequential layer model.

Acknowledgment. This work is a part of a project undertaken at the British University in Dubai.

References

1. AlGhanem, H., Shanaa, M., Salloum, S., Shaalan, K.: The role of KM in enhancing AI algorithms and systems. Adv. Sci. Technol. Eng. Syst. J. **5**(4), 388–396 (2020)
2. Salloum, S.A., Alshurideh, M., Elnagar, A., Shaalan, K.: Machine Learning and Deep Learning Techniques for Cybersecurity: A Review. In: Hassanien, A.E., Azar, A., Gaber, T., Oliva, D., Tolba, F. (eds.) Joint European-US Workshop on Applications of Invariance in Computer Vision, pp. 50–57. Springer, Cham (2020)
3. Salloum, S.A., Alshurideh, M., Elnagar, A., Shaalan, K.: Mining in educational data: review and future directions. In: Hassanien, A.E., Azar, A., Gaber, T., Oliva, D., Tolba, F. (eds.) Joint European-US Workshop on Applications of Invariance in Computer Vision, pp. 92–102. Springer, Cham (2020)
4. AlShamsi, M., Salloum, S.A., Alshurideh, M., Abdallah, S.: Artificial intelligence and blockchain for transparency in governance. In: Hassanien, A., Bhatnagar, R., Darwish, A. (eds.) Artificial Intelligence for Sustainable Development: Theory, pp. 219–230. Cham, Practice and Future Applications, Springer (2021)
5. Yousuf, H., Zainal, A.Y., Alshurideh, M., Salloum, S.A.: Artificial intelligence models in power system analysis. In: Hassanien, A., Bhatnagar, R., Darwish, A. (eds.) Artificial Intelligence for Sustainable Development: Theory, pp. 231–242. Cham, Practice and Future Applications, Springer (2021)
6. Alomari, K.M., Alhamad, A.Q., Mbaidin, H.O., Salloum, S.: Prediction of the digital game rating systems based on the ESRB. Opcion **35**(19), 1368–1393 (2019)
7. Salloum, S.A., Khan, R., Shaalan, K.: A survey of semantic analysis approaches. In: Hassanien, A.E., Azar, A., Gaber, T., Oliva, D., Tolba, F. (eds.) Joint European-US Workshop on Applications of Invariance in Computer Vision, pp. 61–70. Springer, Cham (2020)
8. Wahdan, K.S.A., Hantoobi, S., Salloum, S.A., Shaalan, K.: A systematic review of text classification research based ondeep learning models in Arabic language. Int. J. Electr. Comput. Eng. **10**(6), 6629–6643 (2020)
9. Bretan, M., Weinberg, G., Heck, L.: A unit selection methodology for music generation using deep neural networks. arXiv Prepr. arXiv1612.03789 (2016)
10. Sturm, B.L., Santos, J.F., Ben-Tal, O., Korshunova, I.: Music transcription modelling and composition using deep learning. arXiv Prepr. arXiv1604.08723 (2016)
11. Sarroff, A., Casey, M.A.: Musical audio synthesis using autoencoding neural nets. In: Proceedings of the International Society for Music Information Retrieval Conference (ISMIR2014) (2014)
12. Engel, J., et al.: Neural audio synthesis of musical notes with wavenet autoencoders. In: International Conference on Machine Learning, pp. 1068–1077 (2017)