Music Classification: Predictive Models Using MEL Spectrograms and Convnets

W. Andy Holst

School of Professional Studies, Northwestern University

MSDS 458: Artificial Intelligence and Deep Learning

Dr. Syamala Srinivasan

August 28, 2021

Abstract

With the music industry creating a tremendous amount of new data daily, as well as offering plentiful business opportunities via an enormous total addressable market, machine learning techniques provide an opportunity to accelerate the digital transformation of streaming and online services. This paper addresses how Mel Spectrograms and Convolutional Neural Network (Convnet) models provide exceptional performance in multi-class classification of short audio clips of music recordings spanning ten different genres. In experiments I am able to achieve **88.5% Test Accuracy** with convolutional neural network techniques. This research represents an important first step towards organizing and optimizing online music catalogs through genre classification and building effective music recommendation engines for today’s online listeners.

Music Classification: Predictive Models Using MEL Spectrograms and Convnets

Convolutional neural networks (Convnets) are a subset of neural networks, and their technology utilizes automated feature engineering through the application of different self-learned “filters” across images. The rapid development of Convnets in the early 2010’s, led by Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton (University of Toronto), revolutionized the field of computer vision (Krizhevsky, 2017). Computer vision using Convnets, thanks to its power of embedded feature engineering, has extended into other fields of application as well, and this research paper addresses one such extension.

According to the International Federation of the Phonographic Industry (IFPI) Annual Global Music Report, the music industry produced $21.6 billion in total revenue in 2020 and has been growing at a positive rate for the past six years. Online music streaming represented 62.1% of the music industry’s total revenue and is growing much faster than the rest of the industry, growing 19.9% in 2020 alone (IFPI, 2021).

With the music industry creating a tremendous amount of new data daily, as well as offering plentiful business opportunities via its enormous total addressable market, an opportunity to accelerate the digital transformation of streaming and online services is possible through machine learning. One goal of organizing and optimizing online music catalogs is through genre classification, and machine learning can be used to automatically accomplish this. Additionally, multi-class classification machine learning models represent an important step towards building effective music recommendation engines for today’s online listeners. It is with these opportunities in mind that I conducted the research now presented in this paper. On a slightly more personal level, I grew up in a very musical family and in college I was an active member of multiple musical ensembles, everything from jazz to orchestra. To this day, I thoroughly enjoy music in live venues and outdoor music festivals. So, when I discovered the research of several Department of Electrical and Computer Engineering PhD students at University of California, San Diego, I decided to further explore music genre classification with Convnets.

Literature Review

Preparation for this research paper involved reviewing several different similar papers focused on the classification of sounds and, more specifically, on music genre classification. My literature review started at the University of California, San Diego (UCSD), which includes a group of researchers led by Peter Gerstoft called the “Noiselab” focusing on “physics-based machine learning and signal processing mostly using just weak noise as a source” (Gerstoft, n.d.). In 2019, UCSD Department of Electrical and Computer Engineering PhD students published a research paper entitled “*Audio Recognition using Mel Spectrograms and Convolution Neural Networks”*. The work focused on the recognition of 80 different classes sounds (Zhang et al., 2019). The UCSD team partially based its research on previous work by Google in 2017, wherein Shawn Hershey and his team reported that “derivatives of image classification networks do well on our audio classification task” (Hershey, et al, 2017). The same year, Google also released a large dataset called AudioSet, designed to advance research into audio event detection and recognition (Ellis, 2017).

In the UCSD students’ research, however, they faced a problem that many of the models used by Google are large and consist of many trainable parameters. Google conducted its research on 70,000,000 samples. But a more workable dataset for the UCSD team was on the order of thousands of samples rather than millions. I chose to follow the UCSD researchers’ approach and use a simple Convnet architecture in my research based on 8000 samples, rather than exploring massive amounts of data with pre-trained models.

My literature review led me to conclude that the most common method employed in sound classification research is to convert sounds (in my case musical recordings) into visualizations by a process called Mel spectrogram representation, and I found a very informative tutorial by Kunal Vaidya to inform this process (Vaidya, 2020). A Mel spectrogram visually maps audio data (raw data is typically in a .wav format) into a 2-dimensional matrix “picture” format, with time corresponding to the x-axis and sound frequencies (in Hertz) corresponding to the y-axis. The spectrogram image is produced by representing sound magnitude (in decibels) across both time and frequency as a range of different colors (for this research the “viridis” colormap was used). An example of a Mel spectrogram created during my research, from a 3-second clip of the song “Lady Marmalade” (using the Librosa Python library) is shown in Figure 1 below.

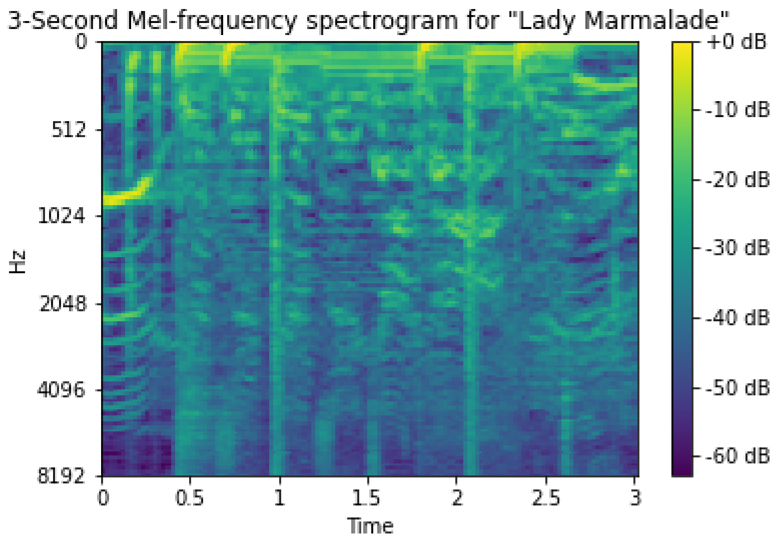

Figure 1: 3-Second Mel-frequency Spectrogram for “Lady Marmalade”

Lastly, I read the seminal work associated with the dataset used in my research, a 2002 paper by George Tzanetakis and Perry Cook, which introduced the GTZAN dataset to the world

(Tzanetakis & Cook, 2002). With this literature review now completed, I have developed a thorough understanding of how to prepare audio data for analysis and classification using convolutional neural networks.

Data

The dataset used for this research -- the GTZAN database of audio files -- is available in the Tensorflow datasets catalog (https://www.tensorflow.org/datasets/catalog/gtzan). The GTZAN database was originally compiled by George Tzanetakis in 2000-2001 from a variety of sources including personal CDs, radio, microphone recordings, in order to represent a variety of recording conditions (Marsyas, n.d.). The dataset consists of 1000 audio tracks each 30 seconds long. It contains 10 genres, each represented by 100 tracks. The tracks are all 22,050 Hertz Mono 16-bit audio files in .wav format. The data preparation, exploratory data analysis and visualization steps undertaken to prepare the dataset for subsequent research, analysis, and modeling were as follows:

- A Google Colaboratory Notebook (Python 3.7.11) was created, with GPU enabled.
- Python Packages imported as necessary for pre-processing and modeling, including: Pydub (0.25.1), Librosa (0.8.1), PIL (7.1.2), Numpy (1.19.5), Pandas (1.1.5), Tensorflow (2.5.0), Sci-kit Learn (0.22.2), Matplolib (3.2.2). Seaborn (0.11.1), and Time.
- Importing the GTZAN database, which is available using the Tensorflow API: `tfds.load('gtzan')`
- Splitting each of the 1000 audio tracks, which each are exactly 30 seconds long, into 3 second clips, thus increasing the total number of records to 10,000 total audio clips.
- Converting each of the 10,000 3 second audio clips into a Mel spectrogram using the Python library Librosa. It should be noted that this conversion took 2 full days to process.

The process was exacerbated by Google Colaboratory wanting “interactive” sessions and periodically displaying a popup to confirm presence of a person at the computer, so when I attempted to complete the conversions overnight, the processes failed to complete due to Google Colaboratory shutting down. It was a painstaking and time consuming process. In future research, improved methods for conversion of audio clips to Mel spectrograms should be explored.

- Each Mel spectrogram representation initially was a color image of size `(432,288,3)` depicting sounds using the “viridis” colormap. However, I recognized that these images included a border of white around the entire perimeter of each image.
- So, in order to improve the speed of modeling, I removed the white borders from all 10,000 images. Each resulting image, with only the Mel spectrogram data remaining and no borders, was reduced size of `(220,216,3)`
- To further improve modeling speed, I resized each image to `(128,128,3)` using the Python Library PIL. Reducing image size did not appear to visually have any impact on the quality of the image, and modeling results were actually better using this smaller image size (see Experiment 8 and 10).
- Using conventions suggested by Jason Brownlee in his *Deep Learning for Computer Vision*, I then stored and structured this image dataset on disk within folders labeled with each genre of music, in order to make it fast and efficient to load. To feed images into each model, I used Keras `ImageDataGenerator` class to progressively load the train, test, and validation datasets (Brownlee, 2021).
- After all preprocessing was completed, the final result was `8,000` training images, `1,000` validation images, and `1,000` test images.

Methods: Research Design and Modeling

Over the course of the last three weeks of the Summer Term 2021, I conducted ten (10) experiments on the GTZAN dataset, preprocessed as described above, all of which consisted of modeling a multiclass classification problem. The sound clips are labeled as one of 10 different types: `['blues', 'classical', 'country', 'disco', 'pop', 'hiphop', 'jazz', 'metal', 'reggae', 'rock']`, and are all evenly distributed in count. The neural network models explored in these experiments consisted initially of 2-hidden-layer and 3-hidden-layer deep neural networks (Experiments 1 and 2). Later models utilized the Convolutional Neural Network (Convnet) concept, and different hyperparameters explored included varying number of Convnets layers, consolidating images via a process called max-pooling, using regularization to avoid overfitting, and changing the size of kernels and use of padding in the filtering process. Following the Convnet architecture, a final, fully connected Dense neural network layer provides the interpretation of the convoluted layers and filters that are applied in the modeling process. All models were constructed in a `Sequential` layer-fashion using Keras. The input layer consists of input in the shape of `(128, 128, 3)`. The single hidden layers are all fully connected (`Dense`) layers with `relu` activation function, each with `128` nodes. The last layer uses a `softmax` activation with `10` nodes, outputting a probability distribution over the 10 different output classes. The loss function used in these experiments was `categorical_crossentropy`, and the optimization algorithm used was `Adam`.

Because this problem ultimately involves identification of features present in each Mel spectrogram, and the evolution of this type of data is expected to be frequently updated as more and more music proliferates the Internet, it is expected that this model will require periodic updating. However, as Chollet outlines in his text *Deep Learning with Python,* there are several means of “freezing” portions of computer vision models and utilizing them for “transfer learning.” Thus, updating of these types of image recognition models is expected to be infrequent and will only enhance performance of future Convnets.

Experiments 1 and 2

Our very first deep neural network consisted of a first flatten layer which flattens each `(128x128x3)` image array into a `(49,152)` length linear array. Next are two hidden layers of `128` input nodes each, and then a final output layer of `10` output nodes (corresponding to the 10 genre categories). Regularization is explored for the hidden layers of the model, by first compiling and fitting the model without regularization and then compiling and fitting the model with L2 Regularization (with `L2=0.003`) This model consists of a total of `6,309,386` parameters (including weights and bias). After training the model using `20` Epochs and Batch Size of `64`, the test loss and accuracy, and model build time, are recorded and the `confusion_matrix`; is reviewed. Visual lineplots of accuracy and loss over time (ie. each epoch) are created for the training and validation datasets.

Experiment 3

Experiment 3 consists of a convolutional neural network consisting of two “layer pairs” of Convnet Layer + Max Pool layer. The 2D Convnet Layer contains `128` filters, uses a filter size of `(3x3)` pixels, with a stride of `(1x1)`, and no padding (thus reducing each array from `(128x128)` down to `(126x126)`. Following the 2D Convnet Layer is a 2D Max-Pool Layer which uses a max-pool of `(2x2)` and a stride of 1 in order to reduce the array size from `(126x126)` down to `(63x63)`. A second Convnet Layer + May Pool Layer is added with the same specifications. Next comes a flatten layer which flattens the final `(30x30x256)` filter array into a `(115200)` linear array. A fully-connected Dense layer of `128` nodes processes the filter outputs. A final output layer of `10` output nodes (corresponding to the 10 genre categories). Regularization is explored for the hidden layers of the model, by first compiling and fitting the model without regularization and then compiling and fitting the model with L2 Regularization (with `L2=0.003`) This model consists of a total of `14,898,186` parameters (including weights and bias). After training the model using `200` Epochs, Batch Size of `64`, and Early Stopping with `Patience=3`. The test loss and accuracy, and model build time, are recorded and the `confusion_matrix`; is reviewed. Visual lineplots of accuracy and loss over time (ie. each epoch) are created for the training and validation datasets.

Experiment 4

Experiment 4 consists of a convolutional neural network consisting of **three** “layer pairs” of Convnet Layer + Max Pool layer. The 2D Convnet Layer + Max Pool layer “pair” matches the specifications of Experiment 3. A second and a third Convnet Layer + Max Pool Layer are added with the same specifications. Next comes a flatten layer which flattens the final `(14x14x128)` filter array into a `(25088)` linear array. A fully-connected Dense layer of `128` nodes processes the filter outputs. A final output layer of `10` output nodes (corresponding to the 10 genre categories). Regularization is explored for the hidden layers of the model, by first compiling and fitting the model without regularization and then compiling and fitting the model with L2 Regularization (with `L2=0.003`). **In addition a 20% Drop Out Layer is added between each Convnet + Max Pool pairing.** This model consists of a total of `3,511,434` parameters (including weights and bias). After training the model using `200` Epochs, Batch Size of `64`, and Early Stopping with `Patience=3`. The test loss and accuracy, and model build time, are recorded and the `confusion_matrix`; is reviewed. Visual lineplots of accuracy and loss over time (ie. each epoch) are created for the training and validation datasets.

Experiment 5

Experiment 5 consists of a convolutional neural network consisting of **three** “layer pairs” of Convnet Layer + Max Pool layer, with the same specifications as **Experiment 4**, ***except*** the number of filters is ***increased*** from `128` filters in the First Convnet, to `256` filters in the Second Convnet, and `512` filters in the Third Convnet. A fully-connected Dense layer of `128` nodes processes the filter outputs. This model consists of a total of `14,325,386` parameters (including weights and bias). The test loss and accuracy, and model build time, are recorded and the `confusion_matrix`; is reviewed. Visual lineplots of accuracy and loss over time (ie. each epoch) are created for the training and validation datasets.

Experiment 6

Experiment 6 consists of a convolutional neural network consisting of **three** “layer pairs” of Convnet Layer + Max Pool layer, with the same specifications as **Experiment 4**, ***except*** the option of `padding = ‘same’` is added to each Convnet Layer. This adds a border around each image to ensure that the image size is not reduced in the Convnet Layer. This model consists of a total of `4,494,474` parameters (including weights and bias). The test loss and accuracy, and model build time, are recorded and the `confusion_matrix`; is reviewed. Visual lineplots of accuracy and loss over time (ie. each epoch) are created for the training and validation datasets.

Experiment 7

Experiment 7 consists of a convolutional neural network consisting of **six** “layer pairs” of Convnet Layer + Max Pool layer, with the same specifications as **Experiment 6** (including `padding = ‘same’` but two more Convnet Layer + Max Pool pairs are added. Adding two Convnet layer pairs means the final flatten layer now flattens a `(2x2x128)` filter array into a `(512)` linear array. This model consists of a total of `808,458` parameters (including weights and bias). The test loss and accuracy, and model build time, are recorded and the `confusion_matrix`; is reviewed. Visual lineplots of accuracy and loss over time (ie. each epoch) are created for the training and validation datasets.

Experiment 8 "Best"

Experiment 8 consists of a convolutional neural network consisting of **six** "layer pairs" of Convnet Layer + Max Pool layer, with the same specifications as **Experiment 7** (including `padding = 'same'` but this time **the kernel size of each convolution layer is increased from `(3,3)` to `(5,5)`. In addition the Drop out Layer between each Convnet + Max Pool pairing is increased to 30%.**This model consists of a total of `2,125,322` parameters (including weights and bias). The test loss and accuracy, and model build time, are recorded and the `confusion_matrix`; is reviewed. Visual lineplots of accuracy and loss over time (ie. each epoch) are created for the training and validation datasets.

Experiment 9

Experiment 9 consists of a convolutional neural network consisting of **six** "layer pairs" of Convnet Layer + Max Pool layer, with the same specifications as **Experiment 8** (including `padding = 'same'` but this time **the drop out layer is increased to 50%**. This model consists of a total of `2,125,322` parameters (including weights and bias). The test loss and accuracy, and model build time, are recorded and the `confusion_matrix`; is reviewed. Visual lineplots of accuracy and loss over time (ie. each epoch) are created for the training and validation datasets.

Experiment 10

Experiment 10 consists of a convolutional neural network consisting of **six** "layer pairs" of Convnet Layer + Max Pool layer, with the same specifications as **Experiment 8** (including `padding = 'same'` but this time **the input size of each image is increased from**

`(128x128x3)` to the original size of `(220x216x3)`. This model consists of a total of `2,207,242` parameters (including weights and bias). The test loss and accuracy, and model build time, are recorded and the `confusion_matrix`; is reviewed. Visual lineplots of accuracy and loss over time (ie. each epoch) are created for the training and validation datasets.

Results

Experiments 1 and 2

In Experiment 1, it was determined that the **Test Accuracy = 58.1% but then reduced to 54.3% when correcting for overfitting**. This is better than a "random" guessing (which has an expected accuracy of 10%). But this is certainly not good enough to place into production.

Experiment 3

It was expected and found that the Test Accuracy and model performance increases rather dramatically with the introduction of Convolutional Neural Networks. In Experiment 3 it was determined that the **Test Accuracy = 67.3% and then increased slightly to 68.6% after correcting for overfitting**.

Experiment 4

In Experiment 4 it was determined that the **Test Accuracy = 65.2%** and then increased slightly to 67.4% after correcting for overfitting using L2 regularization. The most significant increase in test accuracy occurred by introducing 20% Drop out layers, **Test Accuracy increased significantly to 73.9% after adding the 20% Drop out layers**.

Experiment 5

In Experiment 5 it was determined that the **Test Accuracy = 65.2%**. Therefore, changing the number of nodes in the Convnet Layers does not help or improve the performance on this set of images.

Experiment 6

In Experiment 6 it was determined that the **Test Accuracy = 73.1%**. Padding around the edges did not improve the overall accuracy of the model, however it did allow for the final pairing of Convnet + Max Pooling to have a dimension of `16` rather than `14`, which allows for an two additional pairs of layers to be added to the network (see Experiments 7 and 8)

Experiment 7

In Experiment 7 it was determined that the **Test Accuracy = 86%**. Improvement over previous models was accomplished by introducing a setting of `padding = 'same'` within each Convnet layer and then adding a fourth, fifth and sixth pair of Convnet+Max Pool layers. Padding corrects for loss of information around the edges of each image by adding a border prior to applying the filters, so that the filters can capture characteristics all the way to the edge of each image. It also maintains the size of the matrix at each layer (prior to max pooling), which allows for two additional pairs of layers to be added to the network.

Experiment 8

In Experiment 8 it was determined that the **Test Accuracy = 88.5%**. Therefore, changing **the kernel size of each convolution layer from `(3,3)` to `(5,5)`** resulted in a noticeable improvement in accuracy.

This model performed the best of all models in this research paper. Since this is the best performing model, I present in Figure 2 and Training / Validation Loss and Accuracy curves and in Figure 3 I present the Confusion Matrix which displays very few misrepresented values.

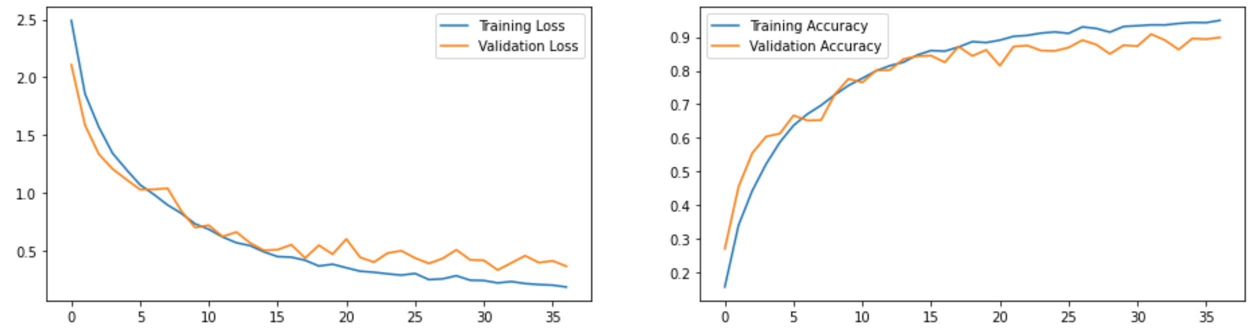

Figure 2 - Training / Validation Loss and Accuracy Curves for Experiment 8

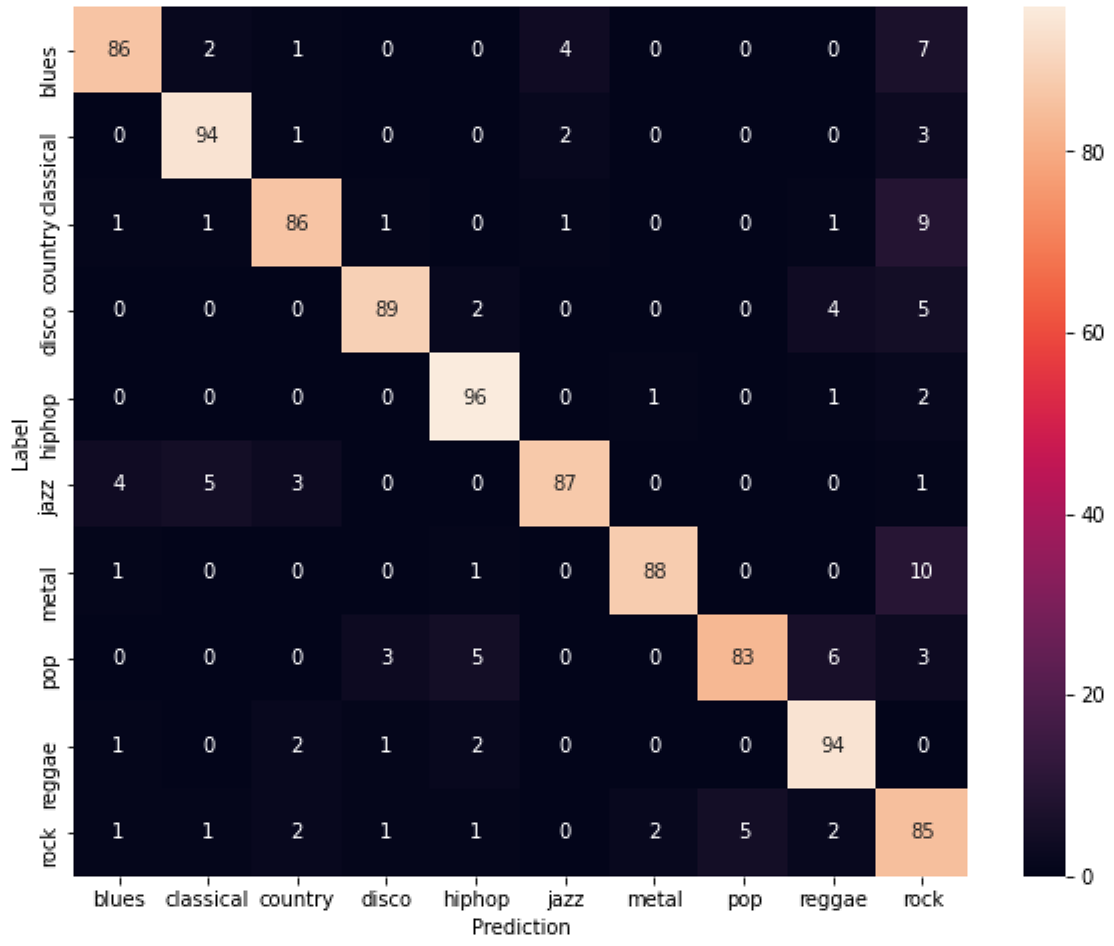

Figure 3 - Final Confusion Matrix for Experiment 8

Experiment 9

In Experiment 9 it was determined that the **Test Accuracy = 77.7%**. Therefore, increasing drop out from 30% to 50% does not help or improve the performance on this set of images.

Experiment 10

In Experiment 10 it was determined that the **Test Accuracy = 83%**. Therefore, increasing the size of each image does not necessarily help or improve the performance on this set of images, and it dramatically increased model build time.

Analysis and Interpretation

The following Table produces final results for all ten (10) Experiments:

Experiment	Build Time (sec)	Train Accuracy (%)	Train Loss	Valid. Accuracy (%)	Valid. Loss	Test Accuracy (%)	Test Loss
1: DNN	537	63.3	1.02	58.0	1.21	58.1	1.17
2: DNN w/Reg	876	52.1	1.49	51.6	1.49	54.3	1.48
3: 2CNN	516	98.2	0.07	67.1	1.61	67.3	1.48
3R: 2CNN w/Reg	394	83.9	0.74	66.7	1.27	68.6	1.18
4: 3CNN	374	94.9	0.16	67.7	1.33	65.2	1.46
4R: 3CNN w/Reg	868	92.9	0.51	76.0	1.10	73.9	1.23
5: 3CNN (incr nodes)	577	81.7	0.75	64.8	1.23	65.2	1.20
6: 3CNN (padding)	597	91.5	0.49	74.8	1.17	73.1	1.17
7: 6CNN	638	89.2	0.36	84.6	0.49	86.0	0.51
8: 6CNN "Best" (incr kernel size to 5x5)	**1343**	**95.0**	**0.19**	**89.9**	**0.37**	**88.5**	**0.37**
9: 6CNN (0.5 dropout)	903	80.9	0.61	78.0	0.69	77.7	0.71
10: 6CNN (large image)	2026	88.2	0.42	84.2	0.56	83.0	0.63

The following steps were instrumental towards increasing model performance: First of all, switching from Deep Fully Connected Neural Networks to Convolution Neural Networks with Max-Pooling results in a significant increase in accuracy, jumping approximately 10%. Next, use of Drop out layers “tames” the validation accuracy and produces better results that are less susceptible to overfitting. Next, moving from two (2) Convnet + Max Pool “layer pairs” to three (3) Convnet + Max Pool “layer pairs” increases the accuracy by another ~4-5%. Then, with padding turned on and adding a fourth, fifth and sixth Convnet + Pool pair, the accuracy increases yet again, topping 86% accuracy. The best model performance was achieved in Experiment 8 by increasing the kernel size from `(3,3)` **to** `(5,5)` with a **Best Test Accuracy of 88.5%**. This final improvement of an additional 2.5% in accuracy is most likely due to the fact the data have “larger” features so increasing the size of the window captures more of the variation in these features.

As a final analysis, I utilized t-Distributed Stochastic Neighbor Embedding (t-SNE) to visualize Experiment 8’s activation features, which are represented by the final fully connected dense layer in the model. Using t-SNE, the 128 activation values are reduced down to 2 dimensions. The first 250 music clips in the test dataset are visualized in Figure 4 below. Visualization was limited to 250 music clips, due to memory constraints. However, regardless of the memory constraint, this visualization powerfully shows how each of the Activation Values are contributing to the model prediction results.

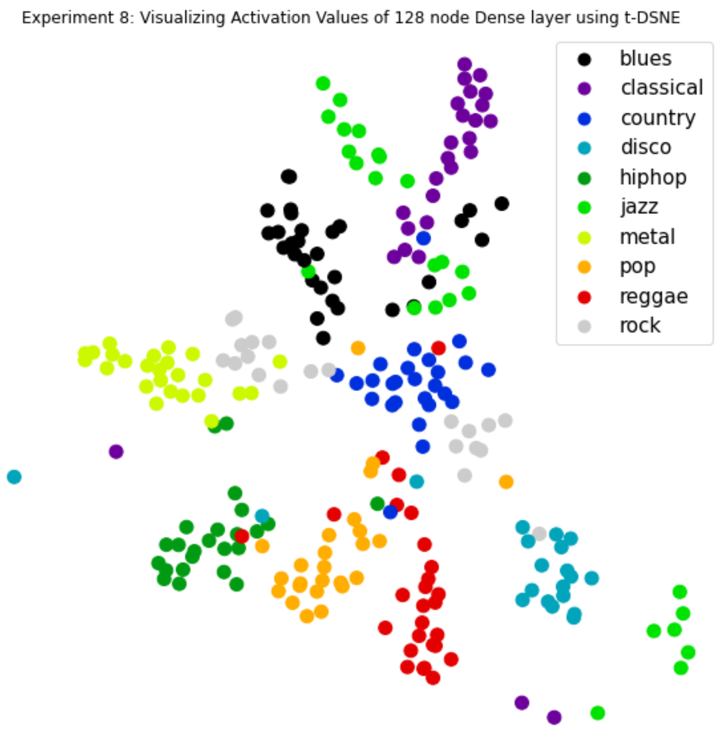

Figure 4 - Visualizing Activation Values of 128-node Dense Layer using t-DSNE

Conclusions

This paper addressed how Mel Spectrograms and Convolutional Neural Networks (Convnet) provide exceptional performance in computer vision multi-class classification of short audio clips of music recordings spanning ten different genres. In our experiments we were able to achieve **88.5% Test Accuracy** with convolutional neural network techniques. This research represents an important step towards organizing and optimizing online music catalogs through genre classification and building effective music recommendation engines for today's online listeners. As someone who really enjoys listening to and enjoying music, this research was extremely rewarding and I appreciate having the opportunity to present these results.

References

Ahmed,M. S., Mahmud, M. Z., & Akhter, S. (2020) Musical Genre Classification on the Marsyas Audio Data Using Convolution NN, *2020 23rd International Conference on Computer and Information Technology (ICCIT)*, 2020, pp. 1-6, https://doi.org/10.1109/ICCIT51783.2020.9392737

Brownlee, J. (2021). *Deep Learning for Computer Vision*. Machine Learning Mastery.

Gerstoft, P. (n.d.). *Welcome to Noiselab*. NoiseLab. http://noiselab.ucsd.edu/index.html.

Hershey, S., *et al* (2017). "CNN architectures for large-scale audio classification," *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 131-135, IEEE, https://doi.org/10.1109/ICASSP.2017.7952132.

Ellis, D. (2017, March 30). *Announcing AudioSet: A dataset for Audio Event Research*. Google AI Blog. https://ai.googleblog.com/2017/03/.

IFPI. (2021, March 25). *IFPI issues global Music REPORT 2021*. IFPI. https://www.ifpi.org/ifpi-issues-annual-global-music-report-2021/.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2017). ImageNet classification with deep convolutional neural networks. *Communications of the ACM*, *60*(6), 84–90. https://doi.org/10.1145/3065386.

Marsyas. (n.d.). *Data sets*. marsyas.info. http://marsyas.info/downloads/datasets.html.

Tzanetakis, G., & Cook, P. (2002, July). *Musical genre classification of audio signals*. IEEE Transactions on Speech and Audio Processing, Vol. 10, No. 5. https://doi.org/10.1109/TSA.2002.800560.

Vaidya, K. (2020, December 9). *Music genre recognition using convolutional neural networks (CNN) - Part 1*. Medium. https://towardsdatascience.com/music-genre-recognition-using-convolutional-neural-networks-cnn-part-1-212c6b93da76.

Zhang, B., Leitner, J., & Thornton, S. (2019). *Audio Recognition using Mel Spectrograms and Convolution Neural Networks.* Noiselab. http://noiselab.ucsd.edu/ECE228_2019/Reports/Report38.pdf.