A Comparative Analysis of LSTM and CNN for Music Classification

Kayla Wright, Matthew Guzman, Lauren Taylor

University of San Diego

AAI 511

Kahila Mokhtari

July 30, 2023

Abstract

Long-Short Term Memory (LSTM) and Convolutional Neural Network (CNN) structures have both been used for music classification tasks. However there is a lack of literature weighing the merits of both models relative to each other. In our project we aim to help bridge the gap in literature by comparing the two models. A dataset consisting of 369 MIDI files from 9 different classical composers was used. The LSTM model obtained an 89% accuracy while the CNN model achieved an 87% accuracy. These results stand in contrast to similar comparative work and raises important questions for future research.

Keywords: CNN, LSTM, music classification

Introduction

Music classification is a task that has garnered recent mainstream attention. One such example is the song identification app Shazam. The app uses mel spectrograms to differentiate songs, which are compressed representations of how the frequency of audio changes throughout time. This creates a sort of "music fingerprint" that can be used to categorize songs(Harris, 2018).

Deep learning methods have been applied to mel spectrograms with varying levels of success. Liu et al.(2017) utilized a convolutional neural network (CNN) in order to classify songs according to their emotional valence. A micro F1 measure of 70% was obtained and the authors expressed the need for finer tuning. However, they also explained that the CNN architecture was advantageous because of its ability to extract features from raw data, alleviating some of the burden of feature selection(Liu et al., 2017).

Palanisamy et al.(2020), used an Image-Net pre-trained deep CNN to categorize different environmental sounds by analyzing spectrograms. The authors evaluate the model using 2 datasets, ESC-50 and UrbanSound8K, obtaining 92.89% and 87.42% validation accuracy respectively (Palanisamy et al.,2020).

Another prominent solution that has emerged in the literature is the use of long short-term memory (LSTM) networks for the classification of music. LSTM networks are known for their ability to capture information in time series data. Music is in fact sequences of notes so it makes sense that this type of architecture would be utilized for analyzing music.

LSTMs have been implemented for music classification with different levels of success. Tang et al.(2018) implemented a multi-layer LSTM model in order to classify music according to

genre. The data set contained music from 10 genres. While the model only obtained a 50% accuracy, authors explain that it outperforms comparable CNN models (Tang et al.,2018).

Deepak and Prasad (2020) also used an LSTM structure to classify music by genre. The authors extracted Mel Frequency Cepstral Coefficient features, which represent the power spectrum of a signal. Two models were trained: LSTM and a combination of LSTM and a Support Vector Machine (SVM). While the combination of LSTM and SVM performed best (98.23% test accuracy), the LSTM alone had a respectable test accuracy of 95.83%(Deepak and Prasad,2020).

While there is substantial literature evaluating the performance of CNN and LSTM models, there is a lack of literature comparing the predictive power of the models in comparison to each other.

One comparison of CNN and LSTM models comes from Gessle and Åkesson (2019). The researchers trained LSTM and CNN models on two different datasets, GTZAN and FMA, which span 10 and 8 genres respectively. In the results, the CNN model (56.0% and 50.5% accuracy) outperformed the LSTM (42.0% and 33.5% accuracy) (Gessle and Åkesson, 2019).

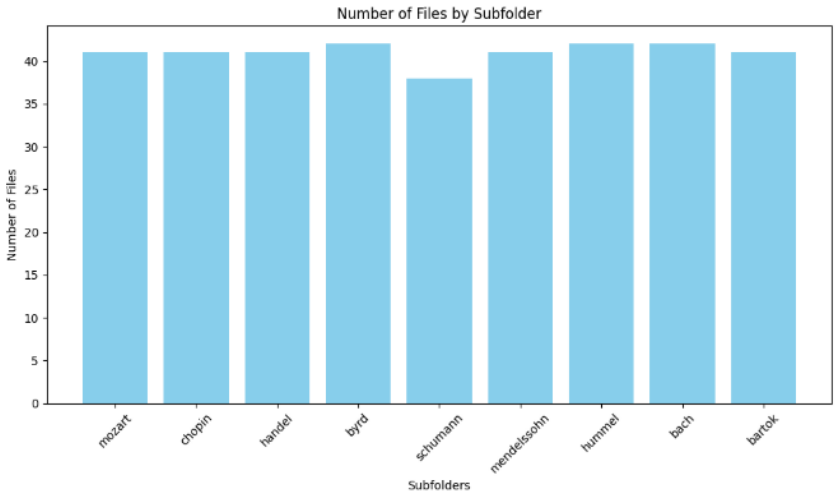
In our project, we wish to add to the body of comparative analysis, while evaluating the models' performance on an intra-genre classification task (eg. artists within a specific genre) as opposed to the more established inter-genre classification that dominates the literature.

Data

The dataset used for this project is a compressed file containing 369 MIDI files (Fig. 1) of classical music from 9 different composers: (Bach, Bartok, Byrd, Chopin, Handel, Hummel, Mendelssohn, Mozart, Schumann).

Figure 1

Number of MIDI files by composer



Data Preprocessing

Spectrograms for CNN

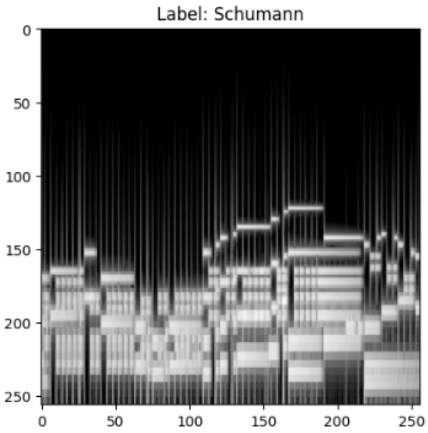
Using the Python os library, we traversed directories containing MIDI files sorted by composers by its respective subfolders. With the assistance of the pretty_midi library, each MIDI file is synthesized into audio, subsequently segmented into uniform 10-second durations to standardize the audio data for deeper analysis.

The segmented audio data is further transformed into a visual representation using the librosa library. By computing the Short-Time Fourier Transform (STFT), spectrograms emerge, shedding light on the audio's frequency components over time. With the aim of categorization, each generated spectrogram is saved as a JPEG image, and labeled according to filename patterns, indicating the respective composer. To ensure uniformity, the research ensures each composer-label category houses a maximum of 1,000 images.

Delving deeper into data quality, the spectrogram images are resized to 256x256 pixels using the cv2 library from OpenCV and are then converted into grayscale, offering consistent data for neural network training. Once preprocessed, the dataset finds its home on Google Drive, ensuring easy storage and accessibility for subsequent phases.

Figure 2

An example of the final spectrogram in the dataset (Label used for visualization purpose only)



These spectrograms were then labeled for the correct composer, one of the nine present in the dataset. A path of the spectrogram folder was made and a function was created to iterate through every item in the folder and label it. Labels and image paths were put in different lists. Length checking was used to ensure the correct number of labels and image paths were uniform. There were about 1000 spectrograms randomly selected for each composer from over 10,000 images. However, Bartok only had around 595 images due to that being the maximum number of spectrograms present for that composer.

After this, the labels were mapped based on the composer's name. These labels were then one-hot encoded so that the CNN model could classify them. Since the spectrograms were still paths to the original, they needed to be converted to 2D arrays. This is because the model uses the 2D array representation of an image to predict. Since there were a variety of images,

multithreading was used for this process to increase efficiency (joblib library was used. 2D array conversion was performed with the cv2 library. The data was then split into train, test, and validation sets with a 0.7/0.15/0.15 ratio.

Finally, the arrays were reshaped into size (-1, 256, 256, 1) so that data augmentation could be performed. The ImageDataGenerator was used to make variations in the spectrogram. We implemented this so that the model would have various different aspects of images to learn from. The parameters were: rotation range = 15, width shift range = 0.1, height shift range = 0.1, and horizontal shift = true.

This model took quite some time to optimize due to lack of data in the beginning. Originally we used only one spectrogram for each song, totaling in only 369 images that our CNN could learn from. We found this to not be an optimal way to achieve results, because after a certain point, our CNN stopped learning from the images.

Our team implemented a system where we took 10 second intervals from each song, labeled them, and then preprocessed this data. These songs are normalized to maintain consistent amplitude levels. This normalized audio data is segmented and transformed into a spectrogram, which is visualized and saved as JPEG images. The normalization process scales the audio waveform's amplitude to a range between -1 and 1 by adjusting it based on the waveform's maximum absolute value. The goal is to create standardized visual representations of audio segments for further use or analysis.Spectrograms show the loudness and change over time in hertz for each song. These patterns can potentially be identified and learned for each of our nine composers.

Sequences for LSTM

Leveraging the Google Colab environment, we established a connection to the original data within the Google Drive environment. Various libraries were incorporated, such pretty_midi and music21 for MIDI file processing, and numpy and pandas for data manipulations. Data was transformed with a customized function aimed towards extracting each MIDI file's base file name and their musical notes. The extracted musical notes contained its own tuples of three values: pitch, velocity, and duration: Each of these values were then normalized to fall within the range [0, 1]. List of notes were further split into overlapping sequences, each having a specified length. The MIDI file name is also returned. Each file in the directory is iterated through and if a MIDI file is found, the 'preprocess_midi' function is called to get the sequences and file name. Results are added to a master list and then appended to a dataframe.Processed data was manually run for each composer as each data frame was exported to an HDF5 file for further analysis.

The created data files are reloaded to the notebook and are further split into segments based on unique values and constructs a new dataframe containing segments based on unique values in the 'File Name' column. It then extracts the first 1,000 rows from each segment and returns a new dataframe. The resulting data frame returned 368,000 rows and for columns X_ data frame is further segmented so that each input size is 100x3 to match the y target shape.

Labels are then created given the file name column and grabs the composer's name for each row. These labels are separated into the 'y' list and used for the target. The other columns (Pitch, Velocity, and Duration) are used as the predictors and stored in the 'X' data frame. Train & Test data was split by 80% and 20% respectively. The 'X' train and test data was converted to an array data type while the 'y_train' data was shaped using Keras 'to_categorical' function to

reshape. The Final train set results ended up being 2,944 instances while the final test set contains 736.

Model Training/Building/Optimization

Convolutional Neural Network

The spectrograms, now as 2D arrays, were placed through a variety of different architectures to determine which was the best to predict the composer it came from. As stated above, the spectrograms were now 10-second intervals from each song and randomly selected due to hardware constraints for each composer. There were a total of 8,572 spectrograms out of the over 10,000 we created across all composers.

The splitting of data was discussed in the previous section with training images/labels having 6,000 items, validation images/labels having 1286 items, and test images/labels having 1286 items. The data was stratified based on the one-hot labels spoken about in the previous section. This entire process was carried out by using the train_test_split library from sklearn.model_selection. As stated before, augmented images were fed through the CNN model to make predictions.

Our final and best-performing model utilized a VGG19 block, a ResNet block, and a DenseNet block. VGG19 was created by the Visual Geometry Group. It consists of 16 convolutional layers, four fully connected layers, and ends with the softmax activation function. This architecture was created for the ImageNet Large Scale Visual Recognition challenge and outperformed a great deal of competitors (Bardhi, 2021).

Resnet, created by AlexNet, has been paving the way for advances in computer vision. ResNet helps eliminate the vanishing gradient problem by learning the difference between the output and input layers. This causes the deep network to learn the transformation being done

entirely which helps the gradients flow as easily as possible to flow through the neural network with backpropagation. A large network of layers can be made with this method, since the vanishing gradient is not as large, which allows complex patterns to be learned in data (Feng, 2022).

DenseNet was created by Cornwell University, Tsinghua University, and FacebookAI Research. Each layer of this allows inputs from layers before it and causes concatenation. For each composition layer pre-activation batch norm, ReLU, and convolution is used. There is then a bottleneck layer used to reduce complexity. Then, there are several dense blocks with transition layers consisting of different versions of convolution to transition between dense layers. Finally, a global average pooling is done with the softmax activation function (Tsang, 2018).

The basic architecture of the model takes the following form:

- The regularization strength for the L2 regularizer is set at 0.0001
- A function consisting of the VGG19 blocks with several convolutional layers stacked together with a fixed 3x3 kernel size. There is an optional 2x2 max-pooling operation that reduces spatial dimensions.
- A function consisting of ResNet addressing the vanishing gradient problem and learning complex patterns of spectrograms. The residual block function starts by creating a "shortcut" connection to bypass the main operations. The 3x3 convolutions are performed with batch normalization applied to them. The results of these operations are then added to the input, forming the "residual connection." Activation is applied after the addition.
- A function for the DenseNet operations in this architecture. It starts with batch normalization and activation, followed by a 1x1 convolution. This operation compresses the channels before expanding them again. Another batch normalization and activation is

applied followed by a 3x3 convolution. Output is then concatenated with the input, leading to increased channel dimensions.

- GlobalAveragePooling2D to reduce dimensions of spectrograms.
- Dense fully connected layer of size 64, ReLU activation, and L2 regularization of 0.0001.
- Dropout layer of 0.5.
- Dense fully connected layer of size 9 and softmax activation.
- A custom loss function for weights and penalties. When the model was run originally: Mozart, Hummel, Chopin, and Bartok songs were having issues being classified at different rates. This function allows the penalty for different songs for misclassified songs to be stronger and at different rates from others. A value of 1.0 is used for a normal penalty in misclassification. The following weights were changed from 1.0: Mozart = 1.5, Hummel = 2.0, Chopin = 1.5, and Bartok = 1.5.

The optimizer used was Adamax with a learning rate of 0.0001. The model was compiled with this optimizer and our custom weighted loss function. Early stopping was used to monitor validation loss, patience set at 25, and set to alert the team if this was implemented. The model was then fit to iterate over 100 epochs with a batch size of 8. Higher batch sizes did not significantly improve our model.

Long Short-Term Memory Network

The data used was time series, containing pitch and tempo information extracted from the music files. In order to increase the size of the available training data each song was broken into smaller segments of length 100. Labels for classification, which were numbers ranging (1-9), representing the 9 composers. Training and test data were allocated using an 80/20 split,

resulting in a training set of length 2944, and a test set of length 736. In order to prepare the training labels for the model were converted into a binary class matrix of size (2944,10).

The basic architecture of the model takes the following form:

-Dense input layer

- Intermediate layer(s) with dropout (Number of layers vary)

-Dense layer

-Dense output layer(10, activation = softmax)

In order to tune model hyperparameters Keras tuner architecture was used. These hyperparameters were optimized:

-Learning Rate: (0.0001, 0.001, 0.01)

-Input Layer Nodes: (32, 64, 100, 120, 240)

-Number of intermediate layers(1 or 2)

-Intermediate layer(s) nodes (32, 64, 100, 120, 240)

-Dropout : (0.2 or 0.4)

-Dense layer Nodes(32, 64, 100, 120, 240)

After tuning the hyperparameters the optimal values were:

-Learning rate = 0.0001

- Input layer nodes = 32

-Number of intermediate layers = 2

- Layer 1 nodes =120

-Layer 2 nodes = 240

-Dropout = 0.2

-Dense layer nodes = 120

The model was then fit using the training set over the course of 300 epochs.

Model Evaluation

CNN

The combination of VGG19, ResNet, DenseNet, and early stopping in one model yielded decent metrics. Our accuracy was 87% overall, precision was 87%, recall 87%, and F1 Score 87%. We think using spectrograms was a great way to classify these nine composers.

Figure 3

The final metrics of the CNN model.

	precision	recall	f1-score	support
0	1.00	0.99	1.00	150
1	0.78	0.80	0.79	90
2	1.00	1.00	1.00	147
3	0.86	0.80	0.83	150
4	0.93	0.93	0.93	149
5	0.88	0.84	0.86	150
6	0.78	0.77	0.78	150
7	0.87	0.87	0.87	150
8	0.73	0.80	0.76	150
accuracy			0.87	1286
macro avg	0.87	0.87	0.87	1286
weighted avg	0.87	0.87	0.87	1286

Figure 4

The confusion matrix for the CNN model.

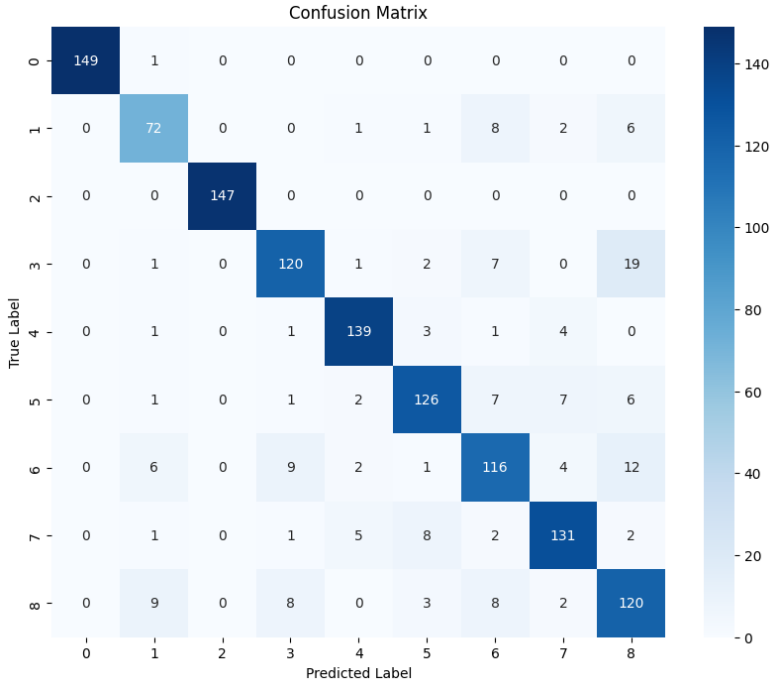


Figure 4 shows that some composer songs are still much more misclassified than others. This is very much the case for composer labels 3 (Hummel), 6 (Byrd), and 8 (Bartok). Bartok's misclassification is understandable because it had the least spectrograms present.

LSTM

The LSTM model performed slightly better compared to the CNN model, obtaining 89% for accuracy, recall, F1 score, and 90% for precision. Figures 5 and 6 are a classification report and confusion matrix, summarizing the LSTM results.

Figure 5

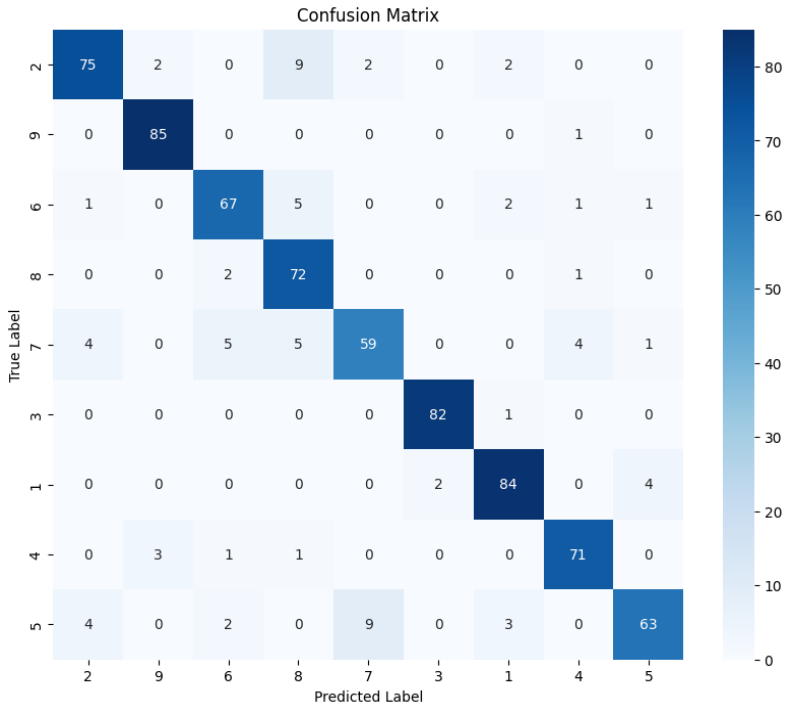
The final metrics of the LSTM model.

Classification Report:

	precision	recall	f1-score	support
2	0.89	0.83	0.86	90
9	0.94	0.99	0.97	86
6	0.87	0.87	0.87	77
8	0.78	0.96	0.86	75
7	0.84	0.76	0.80	78
3	0.98	0.99	0.98	83
1	0.91	0.93	0.92	90
4	0.91	0.93	0.92	76
5	0.91	0.78	0.84	81
accuracy			0.89	736
macro avg	0.89	0.89	0.89	736
weighted avg	0.90	0.89	0.89	736

Figure 6

The confusion matrix for the LSTM model.



While the model performed well, in some categories the model had significant errors. The artists that were misclassified most often were Handel (5), Bartok (2), and Mendelssohn (7).

Figure 7

Summary of model performances.

Model	Accuracy	Precision	Recall	F1
CNN v2	**0.87**	**0.87**	**0.87**	**0.87**
LSTM v1	**0.89**	**0.90**	**0.89**	**0.89**

Discussion

At first, we found that creating a CNN that classifies spectrograms was a daunting task. Our team found overfitting, small data size, different architectures, and hardware constraints to be big issues when it came to making this a reality. Tempo and loudness was different for each song, so great care was taken in normalizing spectrograms to ensure that the model learns all aspects evenly. Our team found that creating more data was the key to increasing performance, as with only 369 spectrograms only gave so much from out CNN to learn from. Once we implemented the 10-second intervals, the model did not stop learning and became more able to identify composers. Audio recordings may not always be free of noise such as background noise and more faint audio. Our team believes that we mitigated much of this noise to allow our predictions to be more accurate.

Overall the LSTM was slightly better than the CNN model for classification. Despite the negligible difference between the performances, the LSTM emerges as the preferable model. This is due to the lower computational complexity of the LSTM model relative to the CNN model.

The higher performance of the LSTM model stands in contrast to Gessle and Åkesson (2019), who found the CNN to be superior. One possible explanation is the difference between the tasks performed. The current project involves classification within a genre while Gessle and Åkesson did classification between genres (Gessle and Åkesson, 2019). It is plausible that LSTM might be more amenable to discriminating within genre differences while CNN might be more adept at recognizing between genre differences.

While generalizations cannot be made due to the small body of research, this project provides some insight. More research is needed to better ascertain the strengths and weaknesses of LSTM and CNN architectures relative to each other. In particular, it would be fruitful to explore differences in performance based on specific music classification tasks. While this project could be improved by having more composers to draw data from, there are notable hardware constraints that our team had difficulties overcoming for both models. It may be ideal, given the hardware, to add more composers to diversify the dataset. This can help predict more diverse sounds and differentiate between artists.

References

Bardhi, M. (2021, February 25). Image Detection using Convolutional Neural Networks. Medium. https://medium.com/mlearning-ai/image-detection-using-convolutional-neural-networks-89c9e21fffa3

Deepak, S., & Prasad, B. G. (2020, July 1). *Music Classification based on Genre using LSTM*. IEEE Xplore. https://doi.org/10.1109/ICIRCA48905.2020.9182850

Feng, Z. (2022, September 16). *An overview of ResNet Architecture and its Variants*. Built In. https://builtin.com/artificial-intelligence/resnet-architecture

Gessle, G., & Åkesson, S. (2019). A comparative analysis of CNN and LSTM for music genre classification.

Harris, A. (2019, January 30). *How does Shazam work?* Medium. https://medium.com/@anaharris/how-does-shazam-work-d38f74e41359

Kamalesh Palanisamy, Dipika Singhania, & Yao, A. (2020). Rethinking CNN Models for Audio Classification. *ArXiv (Cornell University)*.

Liu, X., Chen, Q., Wu, X.-P., Liu, Y., & Liu, Y. (2017). CNN based music emotion classification. *ArXiv (Cornell University)*.

Tang, C., Ka Lung Chui, Yu, Y., Zeng, Z., & Kin. (2018). Music genre classification using a hierarchical long short term memory (LSTM) model. *Third International Workshop on Pattern Recognition*. https://doi.org/10.1117/12.2501763

Tsang, S.-H. (2019, March 20). *Review: DenseNet - Dense Convolutional Network (Image Classification)*. Medium. https://towardsdatascience.com/review-densenet-image-classification-b6631a8ef803

Libraries Used

Library Name	URL	Purpose
Collections	https://docs.python.org/3/library/collections.html	To count instances of composers
CV2	https://pypi.org/project/opencv-python/	Provides tools and functions for image and video manipulation
CSV	https://docs.python.org/3/library/csv.html	Reading Comma-Separated value files
Glob	https://docs.python.org/3/library/glob.html	Pathnames iterator
Google Colab	https://colab.research.google.com/?utm_source=scs-index	To import shared datasets into singular notebooks
H5py	https://docs.h5py.org/en/stable/	Exporting file for preprocessing
IPython	https://ipython.org/documentation.html	To display a random image from folder path
Joblib	https://joblib.readthedocs.io/en/stable/	To utilize multithreading when there were hardware constraints
Keras	https://keras.io/	To add layers to models, to create a custom class weights function, and to use DenseNet, ResNet, and VGG19 architectures.
Keras tuner	https://keras.io/keras_tuner/	To do hyperparameter tuning

Librosa	https://librosa.org/	Analyzing and processing audio and music signals
Matplotlib	https://matplotlib.org/	Plot various graphs to understand our data
Numpy	https://numpy.org/	Numerical and Scientific computing
Pandas	https://pandas.pydata.org/docs/	Load in dataset and adjust any needed changes in the dataframe
PIL	https://pillow.readthedocs.io/en/stable/	Opening various image file formats
Pretty MIDI	https://craffel.github.io/pretty-midi/	Creating, manipulating, and analyzing MIDI files with musical notations, events, and instruments
Random	https://docs.python.org/3/library/random.html	To generate random grayscale images and to get random spectrograms
Seaborn	https://seaborn.pydata.org/	To map confusion matrices
Sklearn	https://scikit-learn.org/stable/#	Statistics and train test split
Tensorflow	https://www.tensorflow.org/resources/libraries-extensions	To build architectures for both models