Transfer Learning with Convolutional Neural Networks on Mel-Spectrograms for Accurate Song Genre Prediction

Jeffrey Kuo **Tim Tantivilaisin** **Bryan Wang** **Tessa Weiss**
University of California, Berkeley
{jrkuo2015,timt,bryan.wang,tweiss}@berkeley.edu

Abstract

This paper explores the use of transfer learning on Convolutional Neural Networks (CNNs) to predict the genre of a song using mel-spectrograms. We employed Google's VGGish CNN architecture, which was ported to PyTorch, and trained the model using the Free Music Archive (FMA) small dataset. The objective was to predict the genre of a song given a raw mp3 file and improve upon the baseline accuracies presented in the original FMA analysis. Our findings show that transfer learning with CNNs on mel-spectrograms can accurately predict song genre with a relatively high degree of accuracy, achieving an accuracy of 56.75%, a 7.75 percentage point improvement. The paper outlines the conceptual baseline of how the CNN utilizes the mel-spectrogram, data description, model architecture and training, results, and next steps.

Presentation Link: https://www.youtube.com/watch?v=0pU0BmEnB0w

1 Introduction

Automatic genre prediction is an important problem in the music information retrieval (MIR) space. Having an accurate genre prediction model is useful for building music recommender systems. On platforms such as Spotify and Apple Music, being able to accurately predict the genres that a user listens to allows us to recommend songs of the same type to the user.

However, this problem is complicated for two main reasons. The first is that a song can fit multiple genres, and the classification of a song can be subjective. The second is that there can be high variability within a genre. For example, the "Electronic" music genre contains high BPM party songs with straightforward beats, as well as more avante-garde, LoFi selections with extremely complex musical structure.

To mitigate these problems as much as possible, we decided to restrict the scope of our project to only predicting eight very distinct genres. We used the "small" partition of the FMA dataset, which contains exactly 8,000 songs. The genres present in this dataset are: "Hip-Hop", "Pop", "Folk", "Experimental", "Rock", "International", "Electronic", and "Instrumental".

The goal of our project was to reproduce state-of-the-art results in genre classification by utilizing convolutional neural networks (CNNs). We compared two different techniques to see which performed better: using a pretrained CNN (VGGish) as an embedding model for other classification models (SVM + XGBoost), and finetuning the pretrained CNN with additional layers for classifications.

The next section will introduce how to perform genre classifications with CNNs and provide some intuition on why this approach is state-of-the-art.

Preprint. Under review.

2 CNN + Mel-Spectrogram

CNNs are a class of deep neural networks that have been successfully applied to various computer vision tasks. CNNs are designed to automatically extract and learn spatial features from images by using convolutional layers, pooling layers, and fully connected layers. In a CNN, the convolutional layers apply a set of learnable filters to the input image, which detect different features such as edges, corners, and textures. The pooling layers downsample the feature maps by reducing their spatial resolution while retaining the most important information. The fully connected layers then process the high-level features and produce the final output of the network. By stacking multiple convolutional layers, pooling layers, and fully connected layers, a CNN can learn increasingly complex representations of the input image, which makes it suitable for a wide range of visual recognition tasks.

The reason why CNNs are so successful in music genre classification is because the audio recognition task is converted into a visual recognition task by utilizing mel-spectrograms.

A mel-spectrogram is essentially an image that is created from an audio file, which is is a one-dimensional array of floats. Each float represents a sample of the song at a particular time point. Every audio file also specifies a sample rate, which is the number of samples per second of audio. For example, Spotify uses mp3 files with a 44.1 kHz sample rate and 16-bit float precision. This means that there are 44,100 floats (in 16-bit precision) per second of audio in the song.

Thus, even 30 seconds of audio will constitute an array of about 1.3 million floats. This input is too high-dimensional for most conventional models to handle, including feedforward neural networks. Even models built to handle time dependency, such as transformers or recurrent neural networks, will perform sub-optimally because of the sheer length that possible dependencies can take. Figure 1 shows an example of a mel-spectrogram.

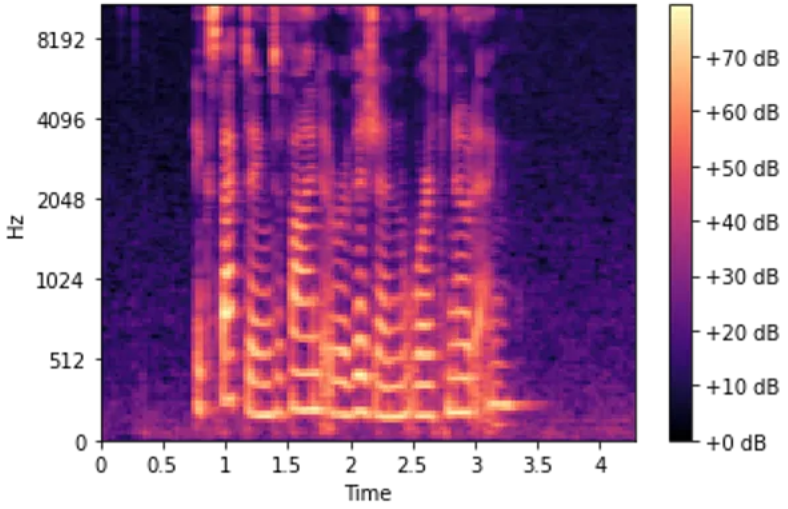

Figure 1: Example of mel-spectrogram (not from our dataset)

Mel-spectrograms allow the user to condense a long audio vector into a compact two-dimensional representation with spatial dependency. This allows CNNs to perform well on the data. To compute a mel-spectrogram, the audio signal is first divided into short overlapping frames, and then each frame is transformed into the frequency domain using the Fourier transform. This technique is often called Short-Time Fourier Transform (STFT). The resulting spectrum is then converted into a mel-scaled spectrum using a bank of triangular filters that emphasize certain frequency ranges that are more important in the human perception of sound. Finally, the mel-scaled spectrum is converted into a logarithmic scale and displayed as an image, where the x-axis represents time and the y-axis represents frequency.

In the following section, we will expand on the data that we used to construct the mel-spectrograms as well as how we implemented the computations.

3 Data and Pipeline

The Free Music Archive (FMA) is a large and diverse dataset of high-quality, legal audio recordings, which is freely available for research and educational purposes. It contains over 106,000 tracks from various genres and cultures, ranging from classical to experimental to hip-hop, and is organized into various subsets and metadata that allow for easy access and exploration.

For this project, we used the "small" partition of the FMA dataset, which contains exactly 8,000 songs. The genres present in this dataset are: "Hip-Hop", "Pop", "Folk", "Experimental", "Rock", "International", "Electronic", and "Instrumental". Each song constitutes 30 seconds of audio sampled at 44.1 kHz. There were six files that were corrupted and removed from dataset, leaving us with exactly 7,994 files. Figure 2 shows the number of times each genre is present in the dataset.

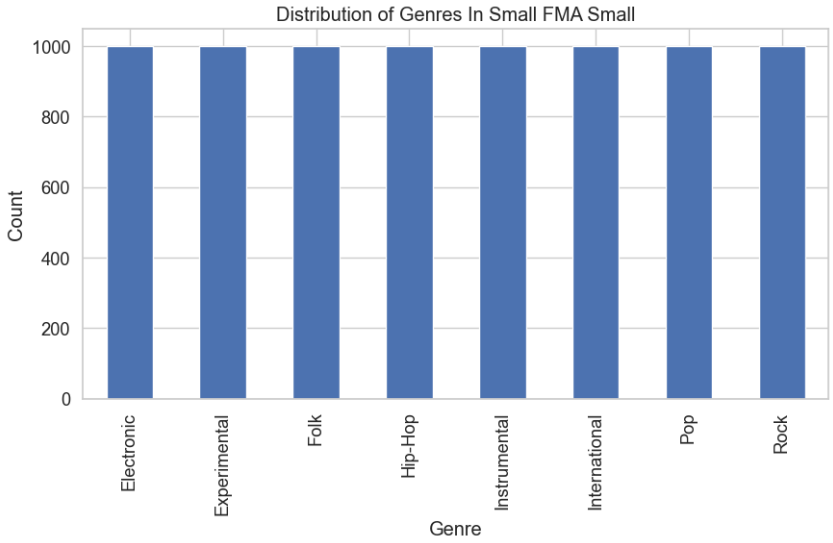

Figure 2: Class balance of genres present in dataset.

We used the preprocessing functions from the sklearn-audio-transfer-learning repository `https://github.com/jordipons/sklearn-audio-transfer-learning` to create the mel-spectrograms. The repository implements standard functions for framing the data, applying a periodic Hann window to each frame and producing a mel-spectrogram. One difference between the implementation of mel-spectrogram in this repository versus the standard implementation of the HTK algorithm is that they represent the conversion from STFT to mel-spectrogram as a matrix multiplication. This increases the number of computations from num_fft multiplies per frame to num_fft^2 multiplies per frame, but has the attraction of being more general and easier to read.

To make inputs compatible with the VGGish model, each song is downsampled to 16 kHz. In addition, each 0.96 seconds of audio is converted into one mel-spectrogram of size 96 by 64. Thus, for this dataset, each 30 second song is converted to a 3D tensor of mel-spectrograms of shape 31 x 96 x 64. The next section will discuss how the data is processed by the VGGish model as well as the two additional approaches we tried on top of the pretrained VGGish model.

4 VGGish Model Architecture and Training

The VGGish model architecture is as follows:

```
VGGish(
  (features): Sequential(
    (0): Conv2d(1, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU(inplace=True)
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (3): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (4): ReLU(inplace=True)
    (5): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (6): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (7): ReLU(inplace=True)
    (8): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (9): ReLU(inplace=True)
    (10): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (11): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (12): ReLU(inplace=True)
    (13): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (14): ReLU(inplace=True)
    (15): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (embeddings): Sequential(
    (0): Linear(in_features=12288, out_features=4096, bias=True)
    (1): ReLU(inplace=True)
    (2): Linear(in_features=4096, out_features=4096, bias=True)
    (3): ReLU(inplace=True)
    (4): Linear(in_features=4096, out_features=128, bias=True)
    (5): ReLU(inplace=True)
  )
  (pproc): Postprocessor()
)
```

Figure 3: VGGish Model Architecture

Essentially, the input passes through a series of convolutional layers, which create a feature vector of size 12,288. This feature vector is then passed through a vanilla feed-forward neural network to produce embedding vectors of size 128. Since one embedding vector per 0.96 seconds of audio is produced, the final output per song in the dataset is 31 x 128.

The VGGish network was originally trained on Audioset, an extremely large dataset containing 70M training videos (5.24 million hours of audio) with 30,871 labels. Because the network was trained on such a rich collection of audio, we hypothesize that it will generalize well to our much smaller task of genre classification on the FMA small dataset.

For both of the following approaches we tried, we split the data into training, validation, and test sets. We used the 80-10-10 split that the FMA team created, so that we could compare our approaches to their baseline results.

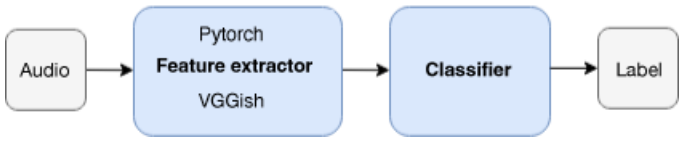

Figure 4: Pretrained model as feature extractor. Altered figure from S. Hershey et. al

Figure 5: Fine tuning VGGish for classification. Altered figure from S. Hershey et. al

The first (Figure 4) approach we tried was directly using VGGish as an embedding model. We embedded all of the songs in the dataset as size $31 \times 128 = 3,968$ vectors. We then tried a few models on the embedded vectors a three of which were: SVM, RandomForest, and XGBoost. The results are shown in the next section. We also tried a PCA of size 128 on the embedded vectors first, before feeding them to a classifier. We hypothesized that the embedding model would help classification accuracy by reducing the dimension of the original input from over 1.3 million to 3,968, which is much more manageable for classification algorithms. We hypothesized that because VGGish is trained on audio, it will find an efficient latent representation of our data that would be better than just naively doing a PCA and then running a classification algorithm. Indeed, the results in the next section show that this hypothesis was correct.

The second (Figure 5) approach that we tried was finetuning the VGGish model to our dataset. We added one hidden layer of shape (3968, 256), and an output layer of shape (256, 8). We computed the softmax of the output layer to get a probability distribution over the 8 genres, and took the argmax as our prediction.

We trained this neural network for 11 epochs, using a batch size of 8. We used the ADAM optimizer with default parameters and vanilla cross-entropy loss. Fine-tuning the VGGish PyTorch port for our task required significant programming effort. The port lacked native support for our necessary steps, particularly batch training. Originally designed for retrieving audio embeddings, it was not optimized for fine-tuning. To address this, we defined a custom data loader that handled the batch preprocessing steps. Additionally, we implemented a training loop to perform the fine-tuning of our model. These steps allowed us to overcome the limitations of the PyTorch port and successfully fine-tune the VGGish architecture for our specific task.

We encountered several challenges during the model fine tuning process, particularly with regard to the limitations of GPU memory. The available GPU memory was insufficient for the model and the batch sizes required for efficient training, necessitating the use of smaller batch sizes. While this helped address the GPU memory issue, it also increased the time required to compute each epoch. Balancing the GPU memory and batch size proved to be a complex issue, with trade-offs between memory usage and training speed. Ultimately, we optimized the batch size to a value that balanced both the memory usage and computational efficiency.

In the following section, we will discuss how our approach improved upon the results achieved in the original FMA analysis. Specifically, we will compare the performance of the original CNN without fine-tuning using classifiers for genre prediction, and the fine-tuned CNN approach. Additionally, we will outline our general areas to expand upon, including potential avenues for future research and further improvements to our model.

5 Results and Next Steps

The following figure summarizes our results with using VGGish as an embedding model. There are three columns in the figure. The first column denotes results with embedding, but without PCA. The second column denotes results with embedding and PCA. The third column denotes baseline results using only raw MFCC features. Note that the first row denotes the dimensions of the features.

Feature Set	Embedding	Embedding + PCA	MFCC (baseline)
Dimension	3968.00000	128.00000	6144.0000
LR	0.47500	0.54750	0.4212
kNN	0.50000	0.49625	0.3688
SVCrbf	0.54750	0.56750	0.4638
SVCpoly1	0.53625	0.54875	0.4250
linSVC1	0.47625	0.54000	0.4162
linSVC2	0.43625	0.46125	0.4300
DT	0.38000	0.41000	0.2925
RF	0.52000	0.52500	0.3550
AdaBoost	0.45375	0.46625	0.2925
MLP1	0.50250	0.47000	0.3875
MLP2	0.50500	0.47500	0.4050
NB	0.50125	0.42625	0.3600
QDA	0.13000	0.48000	0.3950

Figure 6: Results using VGGish as an embedding model vs. baseline

Based on the figure, we see that using embeddings allowed us to outperform all of the baseline models by a substantial margin (with the exception of QDA). However, using embeddings and PCA boosted performance even more and allowed us to outperform all of the baseline models by a significant amount. Overall, this method improved upon FMA's prediction accuracy by 7.75 percentaege points.

For our second experiment, we finetuned the VGGish model directly. Our fine-tuned CNN (Figure 7) achieved a maximum validation accuracy of 61.75% at five epochs. These weights achieved a test accuracy of 53.4%, representing a 4.4 percentage point increase over the FMA baseline.

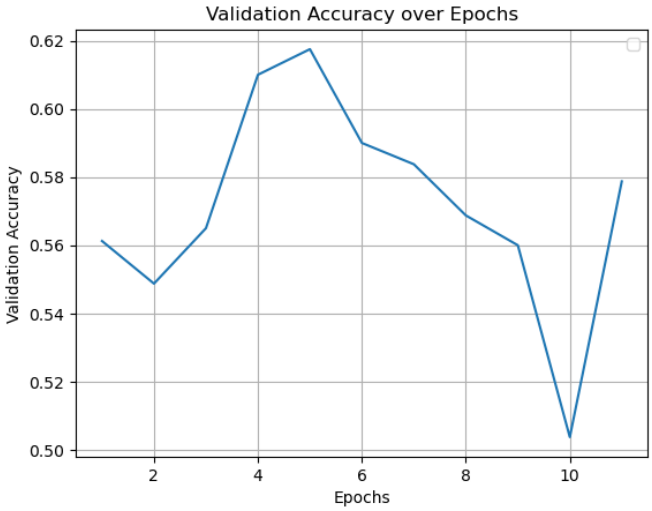

Figure 7: Validation accuracy over 11 epochs.

However, overfitting became evident after the fifth epoch, with the validation accuracy decreasing beyond this point. Despite this limitation, our approach provides an improvement in accuracy over the FMA analysis. In our next steps, we plan to leverage the fine-tuned model as a feature extractor, following figure 1. Additionally, we will experiment with other model parameters, including potentially using fewer channels from the audio file. It is also possible that our small dataset size may have limited the performance of our model, and we aim to explore this further in future research, in addition to classifying additional genres.

6 Appendix

	dim	LR	kNN	SVCrbf	SVCpoly1	linSVC1	linSVC2	DT	RF	AdaBoost	MLP1	MLP2	NB	QDA
chroma_cens	84.000000	25.00%	22.88%	32.00%	26.25%	26.25%	25.25%	18.25%	23.12%	18.62%	24.12%	26.88%	18.62%	23.75%
chroma_cqt	84.000000	27.38%	22.25%	29.25%	26.88%	28.62%	27.12%	22.38%	21.25%	22.50%	28.62%	24.62%	17.25%	13.88%
chroma_stft	84.000000	33.25%	30.88%	37.62%	32.75%	32.75%	34.75%	26.50%	28.25%	27.00%	30.38%	30.38%	16.00%	17.00%
mfcc	140.000000	42.12%	36.88%	46.38%	42.50%	41.62%	43.00%	29.25%	35.50%	29.25%	38.75%	40.50%	36.00%	39.50%
rmse	7.000000	21.12%	21.00%	23.12%	21.75%	21.75%	23.38%	25.37%	22.25%	22.00%	22.88%	20.12%	19.50%	17.75%
spectral_bandwidth	7.000000	31.87%	30.50%	31.37%	31.75%	32.00%	31.87%	28.50%	29.25%	26.25%	32.38%	27.12%	29.00%	28.00%
spectral_centroid	7.000000	30.88%	30.63%	33.12%	32.25%	32.25%	31.50%	29.75%	31.62%	36.88%	31.25%	28.00%	25.75%	26.00%
spectral_contrast	49.000000	36.12%	34.75%	40.00%	37.00%	37.38%	33.75%	26.25%	30.00%	30.25%	32.75%	31.75%	35.12%	33.88%
spectral_rolloff	7.000000	28.38%	30.50%	31.50%	31.37%	31.87%	29.88%	28.62%	31.50%	29.12%	31.25%	29.38%	24.62%	23.88%
tonnetz	42.000000	26.75%	21.75%	27.50%	27.00%	26.25%	27.62%	21.12%	22.75%	24.50%	25.75%	21.25%	22.88%	22.50%
zcr	7.000000	26.75%	22.75%	25.75%	26.25%	24.88%	26.12%	23.88%	23.88%	27.50%	29.12%	25.37%	23.00%	22.25%
mfcc/contrast	189.000000	43.25%	38.50%	48.00%	45.25%	43.00%	42.75%	31.87%	38.12%	32.50%	40.25%	40.75%	39.50%	40.75%
mfcc/contrast/chroma	273.000000	42.38%	35.25%	47.00%	45.75%	41.88%	41.75%	31.62%	34.75%	33.00%	42.00%	43.12%	38.75%	40.62%
mfcc/contrast/centroid	196.000000	43.25%	38.25%	49.00%	45.62%	43.12%	44.62%	32.50%	36.00%	30.50%	43.75%	40.62%	39.62%	40.75%
mfcc/contrast/chroma /centroid	280.000000	43.38%	34.62%	47.12%	46.25%	42.62%	42.12%	32.62%	35.50%	34.62%	39.62%	44.62%	38.50%	41.62%
mfcc/contrast/chroma /centroid/tonnetz	322.000000	42.88%	33.75%	47.75%	47.25%	43.38%	44.50%	32.50%	32.75%	36.38%	40.50%	43.12%	41.00%	40.38%
mfcc/contrast/chroma /centroid/zcr	287.000000	42.62%	33.25%	46.88%	46.88%	42.75%	41.62%	33.12%	34.75%	34.62%	40.00%	41.88%	39.25%	41.62%
all_non-echonest	518.000000	44.25%	32.38%	47.88%	47.75%	40.38%	42.00%	37.38%	35.12%	32.25%	43.62%	46.25%	26.88%	32.62%

Figure 8: Results from FMA's baseline study on the small dataset.

References

[1] S. Hershey, S. Chaudhuri, D. P. W. Ellis, J. F. Gemmeke, A. Jansen, C. Moore, M. Plakal, D. Platt, R. A. Saurous, B. Seybold, M. Slaney, R. Weiss, and K. Wilson. (2017), "Deep clustering: Discriminative embeddings for segmentation and separation," in *International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, IEEE.

[2] Taylor, H. (2019), "torchvggish", *GitHub repository*, https://github.com/harritaylor/torchvggish

[3] Defferrard, M. and Benzi, K and Vandergheynst, P and Bresson, X. (2017), "FMA: A Dataset for Music Analysis", *18th International Society for Music Information Retrieval Conference (ISMIR)*, arXiv.

[4] Jordi P. (2019), "Audio Transfer Learning with Scikit-learn and Tensorflow", *GitHub Repository*, https://github.com/jordipons/sklearn-audio-transfer-learning

[5] Ketan D. (2021), "Audio Deep Learning Made Simple (Part 2): Why Mel Spectrograms perform better", *towardsdatascience*, https://towardsdatascience.com/audio-deep-learning-made-simple-part-2-why-mel-spectrograms-perform-better-aad889a93505