Musical Genre Classification of Audio (Musikgattungszugehörigkeitserkennung)

Master Thesis Submitted to

Prof. Dr. Wolfgang K. Härdle
Prof. Dr. Cathy Yi-Hsuan Chen

School of Business and Economics
Ladislaus von Bortkiewicz Chair of Statistics

Humboldt-Universität zu Berlin

by

David Pollack

578001

in partial fulfillment of the requirements for the degree of

Master of Economics and Management Science

Berlin, June 18th, 2018

Acknowledgement

–

Abstract

We have been struggling with the problem of connecting a few seconds of music to a musical genre since the creation of music itself. Audio data presents a lot of unique problems, which in the past have been solved with handcrafted feature selection based on the plethora of domain knowledge in audio and signal processing. With ever increasing computing power, audio machine learning is moving away from these handcrafted features to a learned feature representation of the raw audio signal. We discuss what makes audio data unique, how one dealt with audio data in the past, and where the future of machine learning on audio is heading.

Specifically, we tackle the problem of audio classification, which attempts to determine the label (or labels) for an entire clip of audio. In the audio realm, other classes of problems include audio transcription, audio denoising / signal separation, and audio generation. Some of the techniques applied in this analysis were adapted from these other classes of problems to solve classification.

While our various network architectures showed promising results on the training data, we were unable to achieve the desired results on our validation and testing sets. We discuss many of the methods that we used to attempt to combat this overfitting, and conjecture why none of our attempts were successful. However our results did achieve an accuracy far better than random guessing, and many of the lessons learned can be applied to other audio machine learning tasks.

The codes used to obtain the results in this paper are available via www.quantlet.de

Keywords: machine learning, audio classification, musical genre, deep learning, dilated convolution

Contents

List of Figures

List of Tables

1 Introduction

The problem of correctly identifying a clip of audio, especially musical audio is a tale as old as time. One of the first successful smartphone apps was Shazam, which uses machine learning to identify the song title from a short recording of the song playing in a room. In December 2017, Shazam was acquired by Apple for a reported 400 million dollars Bloomberg (2017). Clearly, a market exists for audio recognition technology. We have chosen to tackle the problem of genre classification rather than song identification. For one, current copyright law would make it impossible to acquire enough data to build a reasonable classifier. Secondly, musical genre classification is part of a whole family of clustering problems. Similar problems in the field include language identification and keyword detection, although these tend to be precursors to full transcription and further natural language processing (NLP). These problems seek to find generic classes rather exactly matching one recording to another recording. Musical genre classification requires the model to generalize the features not only within each sample but also across the entire genre itself.

Classification of musical genre of an audio clip has many uses. Most obviously, one can tag an audio clip by musical genre to group similar songs together for a recommendation engine. Another application one could explore is finding a distance measure of the musical genres themselves, similar to what Mikolov et al. (2013) has done in NLP with word2vec. With a distance/similarity measure between genres, one could easily make recommendations across genres or use advanced searches such as those done with word2vec. Audio generation is another activate area of research. It presents it's own set issues including a qualitative objective (does the audio "sound natural/good") to unknown output length. Only in the past few months has voice generation gotten to near human quality by Skerry-Ryan et al. (2018). Several of

the most obvious applications of deep learning on audio, bring in a large component of NLP. We wanted to focus specifically on audio and the challenges it presents rather than have these challenges amplified or masked by NLP. Still many of the preprocessing, toolchains, and algorithms can be adapted to other areas of research on machine learning on audio data.

2 Background

2.1 Motivation

The genesis of this project stems from the simple question, what is audio? The explosion of machine learning has centered primarily around images. The digital representation of image data is often a 3-dimensional tensor of 8-bit unsigned integers (between 1 and 255) with one dimension representing 3 color channels (red-green-blue), and the other two dimensions representing height and width. Each data point represents the intensity of each color in each position of the image.

Conversely, audio data only has two dimensions, a variable sampling rate (commonly between 8hz and 44.1hz) and often higher fidelity (signed 16-bit integers). One dimension represents the audio channel (i.e. mono, stereo, etc) and the other dimension represents time. Sound itself is a continuous signal rather than a discrete signal. Digitalization converts this continuous signal to a discrete signal by taking samples of the continuous signal at set intervals and recording these discrete samples as a time-series. An example of digitalized audio comes from a time, long, long ago, when one could purchase audio on something called a compact disc (CD). On CDs, audio is represented as a 16-bit signed integers at 44,100 samples per second (hz or 44.1 khz), on the other hand traditional telephones transmit audio as an 8-bit integer at 8 khz. Thus one second of stereo CD audio would be a 2-channel tensor of length 44,100, whereas the exact same audio over the telephone would be a 1-channel tensor of length 8,000. Assuming the same original audio were to be encoded on a CD and through a landline phone, people would still be able to identify the two encodings as the same sample. Because many aspects of these two samples would be the same,

but the CD audio would objectively sound clearer. So with the problem of bit-length and sample rates, one must choose between fidelity and space/dimensionality, but which of these are better for machine learning? Importantly, how to humans hear sounds?

Going back to the original question, what is audio? We know now the digital representation of audio, but what about the general structure? With images for example, pixels tend to have high correlation with nearby pixels, so if one pixel is part of a red apple in a photo, the pixels next to it will probably be red as well. So what's the audio equivalent? Audio takes on a waveform and the length of the wave of the audio determines what we hear. It can be a single wave, such as a sine wave, but more likely, the audio will be many different waves overlapping each other to create what we perceive as sound. White noise, which has zero correlation, sounds like static, so what we perceive as sounds must have some correlation structure. In fact, what we hear actually depends on several samples that are temporally distant from each other. The earliest signal processing algorithms exploited this structure to create features from raw audio signals.

So earlier, we learned that audio is represented digitally as a time-series, and this time-series is made up of a bunch of different waves. One can use the Fourier transform to go from the time domain into the frequency domain. However, we do not hear pitches in a linear fashion, but rather on a more logarithmic scale, which is why we bin our frequency domain representation of audio on the MEL scale. In 1937, S Stevens J Volkmann and E Newman (1937), created a scale of pitches called the MEL scale. Using this scale, the intensity of the various pitches is represented in a way similar to how we hear sounds. There are actually many MEL scales, but the most famous is that of O'Shaughnessy (1987), $m = 2595 \log_{10}\left(1 + \frac{f}{700}\right)$. However, we do not want to be purely in the time domain nor purely in the frequency domain. One can use a hybrid approach by doing many Fourier transforms across small overlapping windows of the signal and arrange these across time, which is called the short-time Fourier transform, or STFT, which is mathematically represented as:

$$\sum_{n=-\infty}^{\infty} x(n)w(n-m)e^{-i\omega n} \tag{2.1}$$

Early machine learning on audio was done using MEL-Frequency Cepstrum Coefficients (MFCCs), and prior to MFCCs, there were LPCs, Lyon (2013), and LPCCs, Lorenz and Meredith (1999). These had the advantage of taking the high dimensional audio and transforming it into a relatively low dimensional uncorrelated feature representation. The exact series of steps is to take the power of the STFT on the signal (square of the complex number resulting from the STFT), then take the log of this, then do a Discrete Cosine Transform (a type of Fourier transform) Makhoul (1980).

$$X_k = \sum_{n=0}^{N-1} x_n \cos\left\{\frac{\pi}{N}\left(n+\frac{1}{2}\right)k\right\} \tag{2.2}$$

Typically 25ms windows with 10ms hops and 14 MFCCs were employed for rudimentary speech recognition. The MFCCs are relatively inexpensive to calculate and could be utilized by the computing sources of the time. However, 14 features for 25ms of audio would mean that at 8khz, 200 samples are represented by these 14 features. But there are clearly more than 14 different pitches that we can distinguish from. Perhaps a higher dimensional representation of the audio could retain more of the audio information but still reduce the dimensionality to make machine learning feasible. Plus, we are doing two different Fourier transforms to calculate MFCCs. One could simplify the process by removing one of the Fourier transforms. So we turn to the STFT to create spectrograms. Spectrograms, also called periodograms, are simply a visual representation of the STFT of an audio signal often on the MEL scale. To create a spectrogram, one does the STFT to convert the signal into the time-frequency domain, but the standard STFT is not on the MEL scale. Thus one must use a filterbank to bin the resulting STFT from standard frequency bins into MEL frequency bins. These handcrafted features lay the foundation of modern audio analysis.

2.2 Related Works

In one of the seminal works in speech classification, Davis and Mermelstein (1980) discussed the need for a representation of audio that captures a large percentage of the variation in human speech, while maintaining a compact form. They compared MFCCs to linear-frequency coefficients (LFCCs), linear prediction coefficients (LPCs), reflection coefficients (RCs), and linear prediction ceptrum coefficients (LPCCs) and found MFCCs with a similar number of features (coefficients) outperformed the other forms between 2% and 20% in their recognition tests. Notably, Davis and Mermelstein (1980) utilize either 6 or 10 MFCCs, but later works such as Zheng et al. (2001) and Pearce and d'Esclercs (2003) use more coefficients, but not significantly more, with 13 being a commonly chosen number. In audio classification, MFCCs were often paired with hidden markov machine (HMMs) classifiers or gaussian mixture models (GMMs).

With the rise of deep learning, a less crafted set of features could be used as inputs into modern the neural networks, which can decorrelate the simpler feature set. As stated earlier, the MFCCs do a DCT on a filter-banked spectrogram, because the DCT removes correlation between the spectrogram bins, but also inherently destroys some information. A comparison of time frequency representations of audio by Huzaifah (2017) shows that on the task of classification, the same convolutional neural network with a spectrogram representation of audio was able to outperform an MFCC representation of the same sample by 2% to 20%. Intuitively this makes sense because the network can learn relationships that the DCT exemplified, but also could find new useful relationships that were destroyed in the DCT itself. The additional power of the spectrogram comes in an increased number of spectral bins. Spectrograms often have 40 or more bins, while utilizing similar window and hop lengths as MFCCs.

Most importantly, spectrograms are often viewed as images rather than linear input features, which logically leads to the use of image classification networks on audio feature representations.

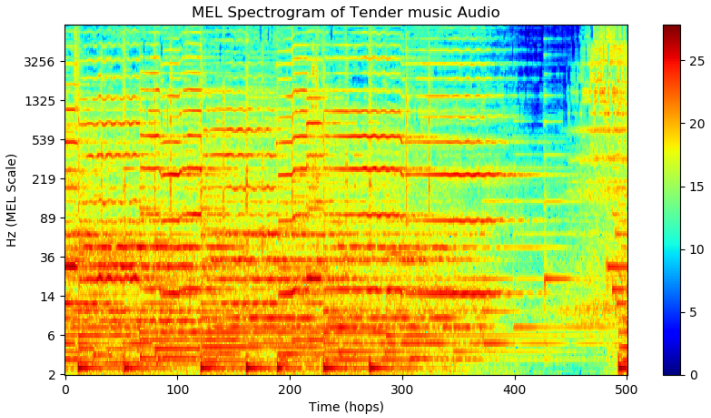

Figure 2.1: Spectrogram of a sample from Audioset

Image classification networks have ushered in a new era of deep learning where they have exhibited unparalleled performance in classification tasks. Furthermore, image classification networks are often trained on one task (ImageNet classification) and then fine-tuned to look for another class of images. One can use these pre-trained networks to do image classification on almost any sort of image, including spectrograms of audio clips. This transfer of domain knowledge could possibly allow one to do world class audio classification using a relatively simple preprocessing step (raw audio -> spectrograms) and a powerful pretrained image recognition network such as ResNet by He et al. (2015).

However, audio is sequence data and often times the desired output of the algorithm is also a sequence. These tasks are well suited to recurrent neural networks (RNNs) according to Härdle et al. (2013). Audio transcription (speech-to-text) and audio generation (text-to-speech) are the two most common tasks in this space. But the length of the raw audio presents problems for traditional neural networks. Spectrograms with their dual representation of frequency and time, and especially with a windowed representation of time, allow RNNs to not get bogged down in the time aspect of audio. While RNNs can remember previous inputs, these connections must be learned and the further away in time based on number of samples, the less likely these connections are to be made.

RNNs are often deployed as so called encoder-decoder frameworks as another solu-

tion to the alignment problem. One RNN will encode the input and then the output from this network will be used as the input for another network that will decode into the desired output. A special type of encoder-decoder framework are attention-based networks. In these frameworks the decoder takes a hidden context from the encoder network, the so-called memory, as an additional input. The information in this hidden context often encodes the exact portion of the unencoded input the encoder found important. The decoder can use this information to focus on the a specific portion of the input to make it's classification.

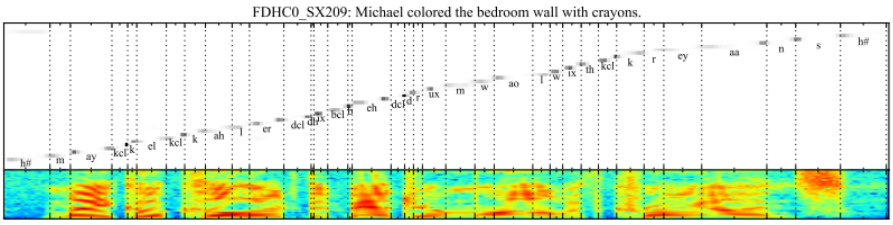

Figure 2.2: Attention-based Phoneme Detection - Luong et al. (2015)

In the audio generation realm, one of the most advanced networks is Tacotron by Skerry-Ryan et al. (2018). Tacotron uses spectrograms of audio as well as text and an encoded speaker identifier as input and outputs raw audio. The original Tacotron networks could create more natural sounding voices than previously possible and newer iterations of Tacotron can even mimic different intonations.

However, not all audio networks take spectrograms as input. Another of the most advanced audio generation networks is WaveNet by van den Oord et al. (2016). WaveNet uses raw audio as the input and encodes the raw audio using gated dilated convolutions. Dilated convolutions are similar to normal convolutions except there are gaps in the convolutional length. For example, a normal convolution with a size of 3 would do the convolution on the first, second, and third samples, whereas a dilated convolution with size 3 and dilation of 1 would do a convolution on the first, third, and fifth samples. WaveNet utilizes an exponentially increasing dilation length (1, 2, 4, 8, 16, ...) to increase the coverage of the feature maps without increasing the convolutional window size. This allows the network to learn from samples that are temporally far from each other, which is important in audio since

sound itself relies on the oscillations of distant samples to produce what we hear. Kalchbrenner et al. (2016) created a predecessor to WaveNet called ByteNet, which also uses dilated convolutions, but does not use the gated convolutions of WaveNet. ByteNet was first developed for neural translation, the task of translating text from one language to another.

Finally, there is Shazam. According to Wang and Th-Floor-Block-F (2003), one of the founders of Shazam, Shazam created fingerprints of each song using power peaks in spectrograms of each song in the database. A user recording of any audio is sent to Shazam, transformed into a spectrogram and then the peaks of these spectrograms are used as fingerprints to be matched against a database of fingerprints from recordings of songs.

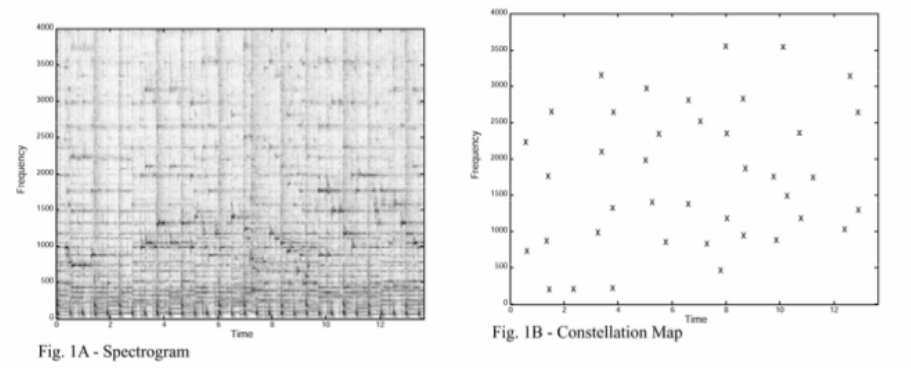

Figure 2.3: Shazam Spectrogram Peaks - Wang and Th-Floor-Block-F (2003)

The peaks of the spectrogram are resistant to noise, since theoretically the peaks will remain peaks in the presence of noise. The matching process is a combinatorial hashing of these peaks, which matches peaks relative to each other. The process is sensitive to millisecond temporal differences even among different version of the same song. Thus this technique would not translate to the process of musical genre classification, because the matching does not generalize to different versions of songs let alone the swath of combinations of peaks across different genres.

2.3 Dataset

We have chosen to use the Google Research (Gemmeke et al. (2017))- Audioset dataset (heretofore Audioset). Audioset is made up of 10 seconds clips of audio taken from Youtube videos with a total of 632 different classes. Each audio clip has 1 or more of these labels. Audioset contains three different subsets, a balanced train set (22,176 segments), an eval set (20,383 segments), and an unbalanced train set (2,042,985 segments). We chose 67 of the 632 labels to isolate only musical labels. Using these 67 musical labels, we were left with 3,146 segments (8.74 hours) in the balanced set, 2,574 segments (7.15 hours) in the eval set, and 224,155 segments (622.65 hours) in the unbalanced set. We attempted to train our networks with the balanced set, but found our algorithms did not generalize. Due to the massive size of the unbalance set, we took a random sample of the 224,155 segments and used 19,042 segments (52.89 hours).

Dataset Name	# of Samples	Total Length
Balanced	3,146	8.74 hours
Eval	2,574	7.15 hours
Unbalanced	224,155	622.65 hours
Unbalanced - Subset	19,042	52.89 hours

Table 2.1: Information on Google Research Audioset dataset

Each segment was labeled by a person. The labels correspond to ideas or categories that come to mind when hearing the clip. We included labels that were not strictly musical genres, but still pre-trained to musical themes such as "Theme music". Specifically, within the "Music" category, we kept "music genre", "music concepts", "music role" and "music mood". Below is a distribution of our subset of labels.

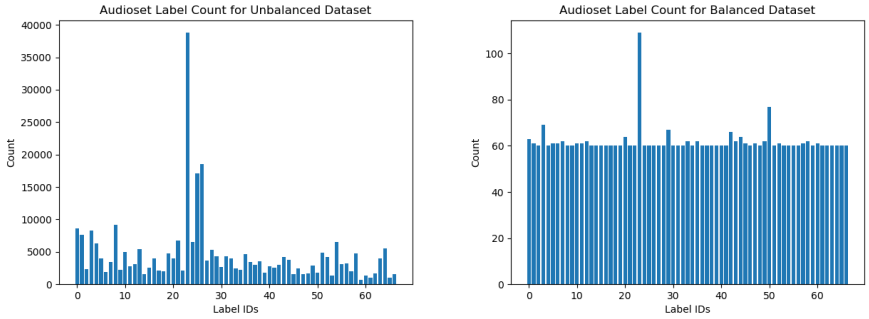

Figure 2.4: Distribution of Audioset Labels

The segments come in different encodings and formats. We converted every clip to 16khz mono wav files. We chose this sampling rate because it should capture most of the frequency range that the human ear is capable of hearing. Higher sampling frequencies would require more samples to fill the same temporal window and require significantly more resources to process. Additionally, most humans can distinguish small differences in audio even at much lower sampling frequencies. In fact one can look to the first modern digital audio encoding for GSM cell phones by Lorenz and Meredith (1999) which produced 8khz audio, and while considered poor quality by modern standards, this ushered in the modern cellular communications era.

Since each sample potentially contained more than one label, we created a vector of binary values of length 67, where 1 represents the presence of a label and 0 the absence of this label. We used the binary cross entropy loss function to train our network with a sigmoid layer on the network outputs to coerce the outputs to be between 0 and 1.

The authors of Audioset used 1 second cuts of each segment on a feature representation of the audio as input into a fully connected neural network classifier and took the average of the 10 predictions to make their final prediction for each clip.

3 Methodology

3.1 Procedure

The goal of this paper is to compare the relative performance of different encoding and decoding methods on audio data for the task of classification. Encoding of the audio took place in either one or two stages. The one stage encoding processes were raw audio to either MFCCs, spectrograms, or learned feature maps with dilated convolutions and the only two stage encoding process was an attention-based RNN encoder on spectrograms, which were themselves from the raw audio. The decoders used were primarily ResNets pre-trained on ImageNet and then fine tuned for the audio classification test, but we also utilized an GRU-based RNN network and a Bytenet-style decoder network. Overall, we tested 6 different encoding-decoding combinations: ResNet34 with MFCCs, ResNet34 with spectrograms, ResNet101 with MFCCs, ResNet101 with spectrograms, Attention-based RNN with spectrograms, and a Bytenet encoder-decoder.

3.1.1 ResNet34

For our baseline encoder-decoder setup, we used MFCC features as the input for a ResNet34 network. This ResNet34 network was pre-trained on the ImageNet dataset on 224x224 images. Thus for MFCCs and spectrograms, we resized these feature representations using bilinear interpolation and used a 1x1 convolution to produce 3 channels from the single channel outputs from the MFCCs and spectrograms. The ResNet34 network consists of an initial 7x7 convolutional layer from the 3 channels to 64 feature maps. Next, four (4) sets (denoted by different colors in the figure below)

of 3x3 convolutional layers with batch normalization between each convolution and a downsample layer for the residual if necessary. The number of feature maps increases upon entering the successive set from 64 to 128 to 256 to 512. Finally, the network has a 7x7 average pooling layer followed by a dense layer with our desired number of outputs, 67. We used rectified linear units as the non-linear activation function throughout the network. A visualization of this network is shown below.

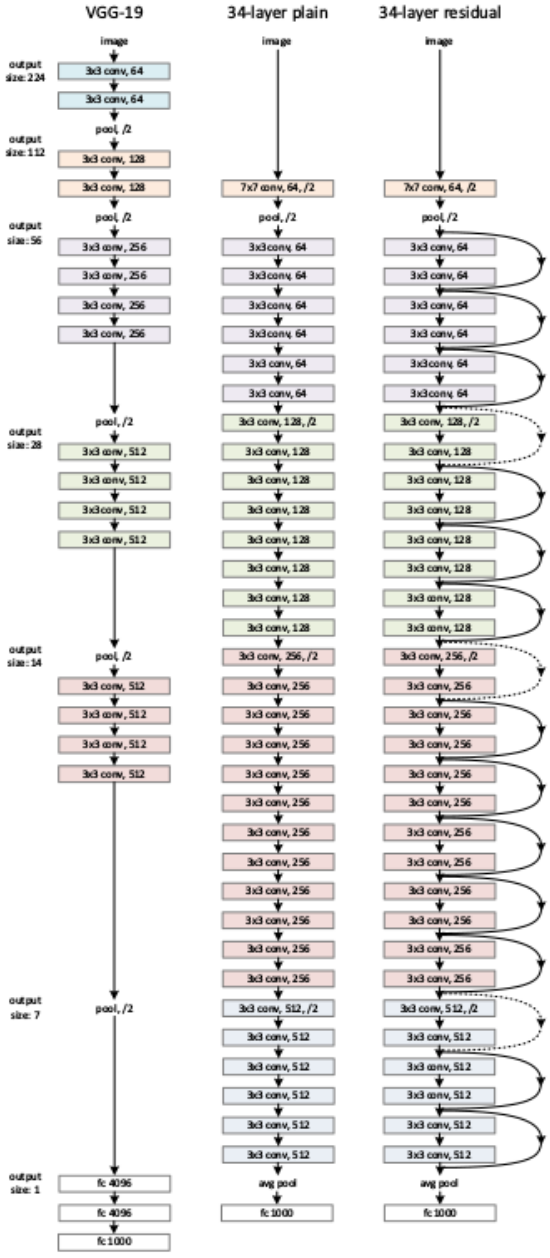

Figure 3.1: ResNet34 - He et al. (2015)

We utilized either a binary cross entropy loss preceded by a sigmoid layer that allows us to use the log-sum-exp trick for numerical stability or a cross entropy loss preceded by a softmax layer. Binary cross entropy allows for the possibility of multi-labels per sample. The loss is calculated by taking the sum of the derivative of the binomial prediction and it's targets, $\hat{y}, y \in \mathcal{R}^C$. Mathematically we can express this as: $\ell(\hat{y}, y) = \sum_{i=1}^{C} \left\{ y_i \cdot \log\left(\frac{1}{\hat{y}_i}\right) + (1 - y_i) \cdot \log\left(\frac{1}{1-\hat{y}_i}\right) \right\}$. On the other hand, cross entropy only optimizes the outputs for one label and importantly the softmax function normalizes the scores of the outputs relative to all the sum of all the outputs. Cross entropy is a simpler metric as can be seen in the formulation of the loss, $-y \log \hat{y}$. We optimize the network using the Adam AMSgrad stochastic optimization method Reddi et al. (2018). We begin with a learning rate of 1e-4 and multiply this by 0.4 at epochs 10, 25 and 40. For the first 10 epochs, we fine-tuned only the fully connected layer, while freezing weights in the rest of the network. After this, we unfroze all of the weights in the network for the remainder of the training procedure.

During each epoch, we used a minibatch size of 100 and recorded the training loss for each minibatch. Additionally we ran our network on a validation set of 100 audio clips and recorded the validation loss and the accuracy on correct positive predictions. After our training procedure had completed, we saved the weights in our network and used these weights on an evaluation set of 2,474 audio clips in the testing (eval set - 100 validation samples) set and recorded the accuracy. During validation and testing, we ran the output through a sigmoid layer to coerce the outputs to values between 0 and 1.

For the ResNet34 network with spectrograms, we created spectrograms with a window size of 600 and a hop size of 300 and 112 filter banks. We then once again did a bilinear interpolation to coerce the output into a 224 x 224 matrix, which is the input height and width expected by the ResNet family of networks. Additionally, we scaled the spectrogram from power to decibels and then rescaled everything to be between [0, 1], because the ResNet networks were trained with such inputs. All other configuration options were the same as the ResNet34 with MFCCs network.

3.1.2 ResNet101

The ResNet101 networks used a 101-layer ResNet-based network. The primary difference between the ResNet101 and ResNet34 networks is the type of blocks used within each set in the network. The ResNet34 utilizes a block as described above, while the ResNet101 utilizes bottleneck blocks. These bottleneck blocks use a 1x1 convolution to reduce the number of feature maps going into the 3x3 convolutions followed by another 1x1 convolution to restore the original number of feature maps. This is done to reduce computing complexity during the 3x3 convolutions, which allows the network depth to increase without having the number of parameters explode. During our experiments, we found this network also seemed to converge faster than the ResNet34 networks. Thus we lowered the total number of epochs to 33 and changed the learning rate schedule decreases to the 10th, 20th, and 28rd epochs, respectively. Other than the bottleneck blocks and the sped up schedule, the configurations of these networks were the same as their respective ResNet34 networks.

3.1.3 Attention-based RNN

The next configuration that we tried was an attention-based RNN encoder-decoder network. We used spectrograms as described above as the input for the encoder network. The encoder network was a simple GRU-based RNN. We used unidirectional 1-layer GRUs with a hidden size of 2500. GRUs were chosen instead of LSTMs to reduce computational complexity. We used the final hidden input from the decoder as the initial hidden decoder input and the spectrogram as the normal input of the decoder network. For this attention network, we calculated the attention score using the "general" method, which does a fully connected multiplication on the encoder outputs to get the attention "energy". Since we are not calculated a sequence, we have also modified the final layers of our network. We transposed the time and features dimensions and then do a 1x1 convolution on the features to reduce the number of features from number of hops * 2500 to number of hops * 250. We flatten the tensor and use a fully connected layer to output the correct number of classes.

For this network, we used a stochastic gradient descent optimization scheme with momentum. The training schedule lasted for 40 epochs with a single learning rate adjustment after the 25th epoch. This network was not pre-trained so we have done an end-to-end training for this network.

3.1.4 ByteNet

For the dilated convolutional network, we used Bytenet encoder network with 6 sets of blocks exponential dilations to 16, 224 hidden features, and a kernel size of three. The Bytenet encoder also used strided convolutions of length 8, 4, 2, and 4 after the 1st, 2nd, 4th and 6th sets respectively to reduce the dimensionality of the encoding. The decoder network was a similar Bytenet style network with a fully connected layer as the classification layer. We used an Adam optimizer with amsgrad similar to the ResNet networks for training rather than SGD but the same annealing schedule as the attention networks.

3.2 Discussion

Unfortunately, while many of the networks converged on the training set, they tended to overfit and not generalize well to the testing set. At first, we experimented with network and training hyperparameters such as epochs, annealing schedule, type of optimizer, learning rates, network depths, number of filter banks, etc, etc. We described the set above that we finally settled on because our networks were performant on the training data and other methods tended to not converge on the training data nor did they validate well during the training procedure. Our best results is a 27.32% accuracy rate with a ResNet101 network on spectrograms trained with the cross entropy loss. Below is a table of our results.

We began our experiments using the binary cross entropy loss, which should have allowed our networks to learn our multi-label dataset. Using the balanced training set, the loss for most of our networks fell throughout training, but the validation set loss and accuracy plateaued.

Network Type	Accuracy
Resnet34, MFCC, BCE, Balanced, No Noise, w/Cache	19.25%
Resnet34, Spectrogram, BCE, Balanced, No Noise, w/Cache	25.81%
Resnet34, MFCC, BCE, Unbalanced, No Noise, w/Cache	18.99%
Resnet34, Spectrogram, BCE, Unbalanced, No Noise, w/Cache	23.62%
Resnet34, Spectrogram, Cross Entropy, Unbalanced, No Noise, w/Cache	26.38%
Resnet34, Spectrogram, BCE, Balanced, Noise Added, w/Cache	25.08%
Resnet34, Spectrogram, Cross Entropy, Balanced, Noise Added, w/Cache	23.49%
Resnet101, MFCC, BCE, Balanced, No Noise, w/Cache	20.23%
Resnet101, Spectrogram, BCE, Balanced, No Noise, w/Cache	26.69%
Resnet101, MFCC, BCE, Unbalanced, No Noise, w/Cache	23.44%
Resnet101, Spectrogram, BCE, Unbalanced, No Noise, w/Cache	25.86%
Resnet101, Spectrogram, Cross Entropy, Unbalanced, No Noise, w/Cache	**27.32%**
Resnet101, Spectrogram, BCE, Balanced, Noise Added, w/Cache	25.46%
Resnet101, Spectrogram, Cross Entropy, Balanced, Noise Added, w/Cache	25.27%
Attn, BCE, Balanced, No Noise, w/Cache	7.35%
Attn, BCE, Unbalanced, No Noise, w/Cache	6.26%
Attn, Cross Entropy, Balanced, No Noise, w/Cache	14.42%
Attn, BCE, Balanced, Noise Added, w/Cache	7.46%
Bytenet, Cross Entropy, Balanced, No Noise, w/Cache	2.45%

Table 3.1: Results Table

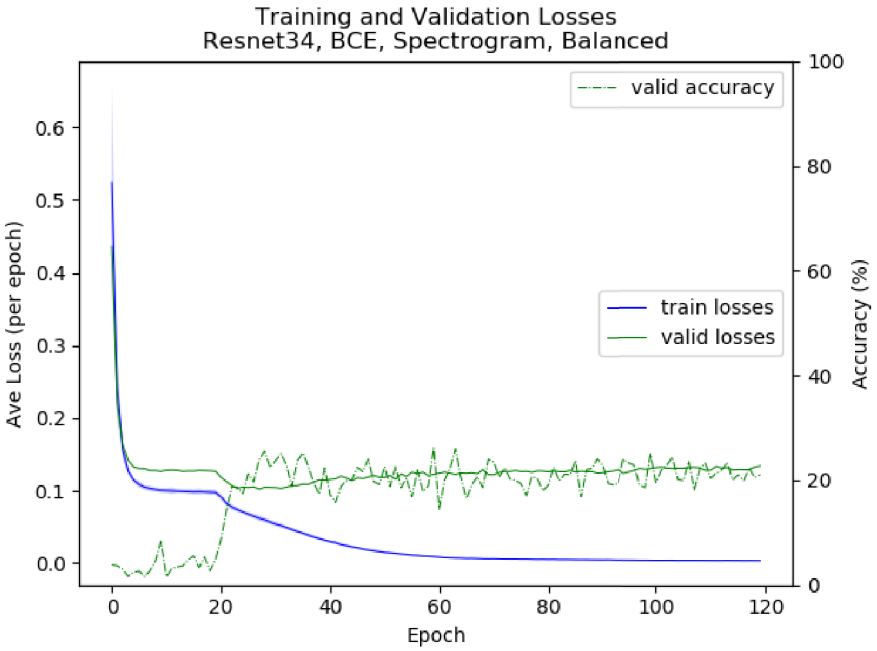

Figure 3.2: ResNet34 Training and Validation Graphs on Balanced Set

MGC

This is a classical sign of overfitting. The naive answer to any overfitting problem is "find more data". Often times this is not an option, but in this case, we had access to the unbalanced set. With more data, we would have more samples per class so our networks could see more examples of a particular genre of music. The hope being with more examples, an intra-genre pattern across samples would develop. We retrained our networks on the larger unbalanced dataset. However, our networks were still not generalizing well to the test set.

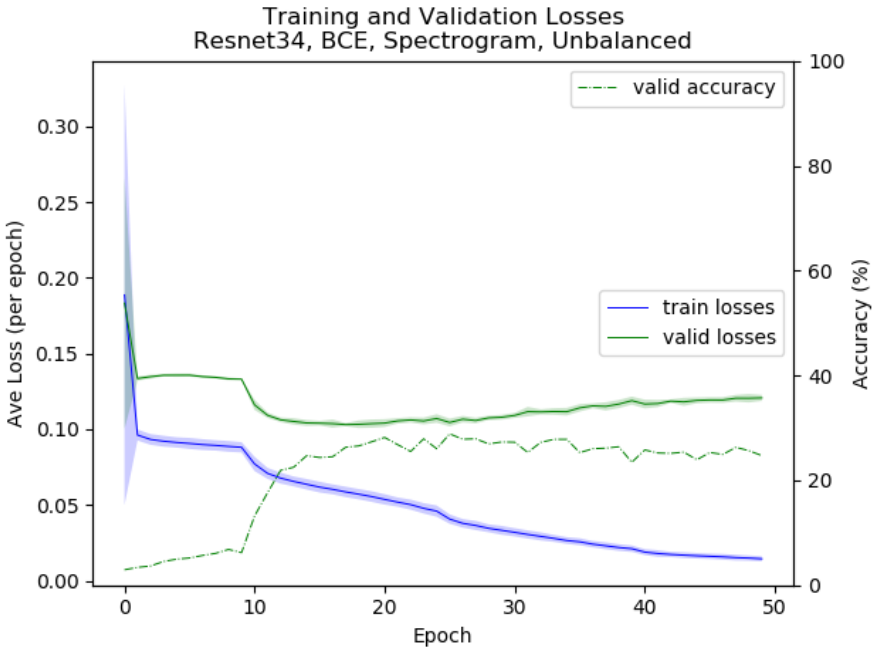

Figure 3.3: ResNet34 Training and Validation Graphs on Unbalanced Set

When "find more data" doesn't work, what does one do? Make more data! We found a small noise dataset from Hu (2015) of 100 different non-speech noises. Since a genre of music remains the same genre regardless of the environment it's played in, we kept the same labels for a sample with or without noise. The Shazam paper also shows evidence that noise does not affect the peaks of a spectrogram, so added noise may not mask important features of our samples, while simultaneously preventing our models from memorizing each sample. We probabilistically mixed one of these noises into our original samples at a low enough volume to not overpower the original sample. This could have increased our dataset 100-fold or combinatorially higher if we had decided to mix more than a single noise into a single sample. However, our results were not promising. Also our results were not as good as when noise was not added to our inputs. For example, our ResNet34 network using the BCE loss on the balanced dataset achieved an accuracy of 25.81%, while the same model with noise added achieved an accuracy of 25.08%. We could have deterministically added each sound to each sample, but this would have increased the run time and memory usage linearly. With the unbalanced dataset, we already were running into memory limits on our system so creating an augmented dataset 100 times larger

than our original dataset was not realistic. It should be noted that to decrease training times, we precomputed and cached the spectrograms. Especially for the Resnet34 network and attention-based network, we noticed that our GPUs were not fully utilized. Thus when adding noises, we recomputed and cached each sample after 10 epochs, but always used unaltered samples for the first ten epochs. Our results did not vary much using caching or no caching. However the training graphs, when utilizing caching, spike after epochs where the cache was refreshed such as in the figure below.

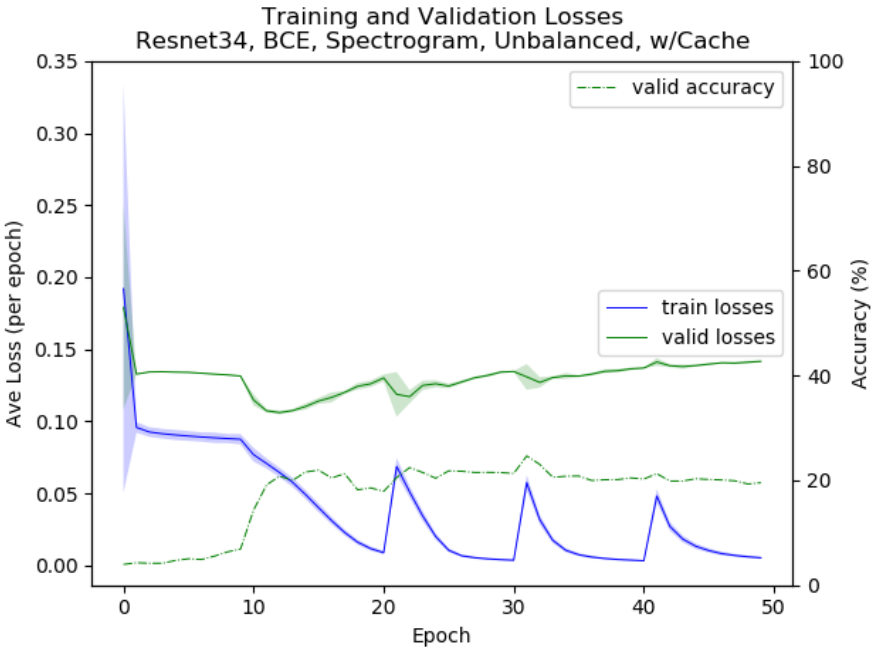

Figure 3.4: ResNet34 Training and Validation Graphs on Unbalanced Set

Finding data didn't work. Making data didn't work. Now what? Creating data from the data we already have. Instead of using the full 10-seconds for each sample, we randomly took a 3- or 5-second section for each iteration or caching operation. Although each sample could overlap across epochs, the features will be temporally displaced so during training our network will have to generalize not only across samples, but also within samples. Perhaps repeating patterns that defined a genre would be found within the samples and then generalized to the genre; however, this was not the case. We found the training loss would still settle in a range similar to the 10 second samples. Interestingly, the training curve declines more slowly with

these sub-samples. We suspect this was due to the network effectively seeing a larger dataset; yet, still not large and diverse enough to overcome overfitting.

More data was used, more data was made, the data was split to get more samples out of the same data, and still the problem of classifying our multi-label dataset remains. So we simplified the problem. Instead of trying to classify multi-labels for each sample, we choose one label for each sample. We then used the cross entropy loss preceded by a softmax activation. Theoretically, if each sample only has one label, then our network shouldn't have to learn if there are multiple labels for a sample or just one. Thus our network only needs to learn learn the correct class and not how many classes there might be and the correct classes for that number of classes. The downside to switching to a single label problem is that perhaps our network would not make connections between similar genres of music. Attacking our new problem, we once again repeated the above process. Unfortunately, the results were the same. The network did not properly generalize to the validation set.

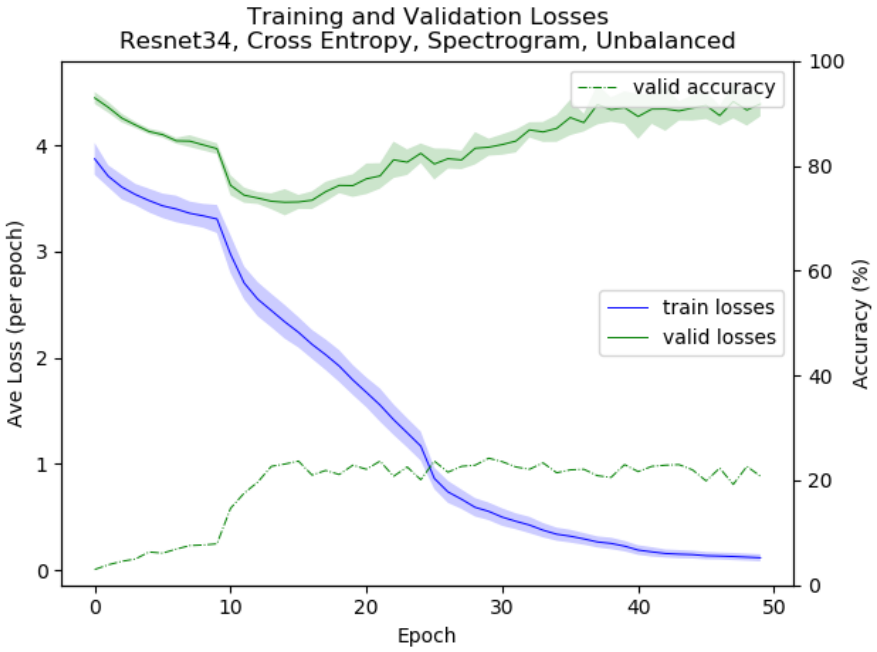

Figure 3.5: ResNet34 Training and Validation Graphs on Unbalanced Set

Finally, we should note that both the Attention-based RNN and Bytenet networks performed rather poorly. In fact, the Bytenet encoder-decoder network never converged. We suspect the end-to-end training process and the relatively small dataset

size contributed to this issue for both networks. Our pre-trained ResNet networks were trained on millions of images and then fine-tuned to our dataset. Perhaps with a larger, cleaner dataset these network would have been able to generalize on the dataset. For the ByteNet network, we also suspect that the dilated convolutions may not have been large enough. WaveNet uses dilations up to 512 with a kernel size of two, which would be equivalent of a dilation of 256 with our kernel size of 3. However, when we tried to increase the dilation to these levels, we ran out of memory on our GPUs.

4 Conclusion

Although we did not get the results that we wanted, our networks did classify the musical genres at a better than random rate. There are many possible reasons for the misclassification. The most obvious is the problem itself. Musical genres are difficult for humans to classify because of their inherent ambiguity. What is dance music or pop music? Is drum and bass a subclass of techno or it's own category altogether? Questions such as these do not have a correct answer to them. Thus, although our data was labeled by humans, it could be that the labels themselves are misleading. Especially with some of the less specific categories, perhaps the dataset does not contain the breadth of feature variation for our networks to properly generalize each class.

Speaking of not having enough samples, even with the larger unbalanced dataset, we were only able to train our models on 19,000 samples for 67 classes. For neural networks, this is a relatively paltry amount of data. Resource constraints restricted us from using the entire unbalanced dataset, but even doing so would introduce additional problems. Specifically, the unbalanced nature of the dataset could skew our network to choose the genres that appear in the dataset most often. There are techniques to combat unbalanced datasets, but many require removing samples in the classes with more samples or duplicating samples in the classes with a lower number of samples. Both solutions are not ideal.

Additionally, the samples themselves come from Youtube videos and the quality of the audio varies greatly. The recording environment, the quality of the recording, background noise and other factors contribute to relatively low quality dataset. It could be that our network memorized these features rather than focusing on the

music itself. We listened to a random selection of the samples and often times had trouble connecting the genre to the music.

However, there are additional steps we did not attempt, which may have given us better results. For one, we could have reduced the number of genres. Specifically, we could have eliminated overlapping genres ("techno" / "drum and bass") or overarching musical themes ("video game music"). We decided against this because we wanted to avoid cherry-picking. Also as stated earlier, the problem of musical genre classification is inherently difficult because of the nature of how people classify musical genres themselves. We did not want to remove that element of difficulty from our problem by hand-picking genres. While our accuracy may have increased to the desired levels, we would still have been plagued by the generalization problems stated above when expanding to the full set of genres. Those problems would have only been masked by our preselected classes. Another solution would have been to visualize the spectrograms as actual images and then take the image representations of the spectrograms as inputs. This method feels convoluted and requires a relatively computationally intensive step of saving an intermediary step to disk. Plus this would have been pure image classification rather than audio classification.

In the end, science is not about attaining a predetermined result, but rather developing processes to solve problems. The lessons learned from so called failure can further one's knowledge in the subject matter as much or more than successful experiments. So while our networks were not able to achieve the desired accuracy, we did achieve many good results. The training loss generally fell to near zero, which means that our network designs did learn the training data. Also, although not at the levels desired, our networks did identify genres at a level far better than random (27% vs 1.5%). There were also a lot of decisions made about the data that furthered our fundamental understanding of audio data and the unique problems it presents. This project was born from a sister project to identify languages in speech samples and we can bring the lessons learned back to that task to improve our results there.

Bibliography

Bloomberg (2017). Apple buys shazam to boost apple music. https://www.bloomberg.com/news/articles/2017-12-11/apple-buys-early-iphone-app-hit-shazam-to-boost-apple-music. [Online; accessed 15-Mar-2018].

Davis, S. and P. Mermelstein (1980, Aug). Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences. *IEEE Transactions on Acoustics, Speech, and Signal Processing 28*(4), 357–366.

Gemmeke, J. F., D. P. W. Ellis, D. Freedman, A. Jansen, W. Lawrence, R. C. Moore, M. Plakal, and M. Ritter (2017). Audio set: An ontology and human-labeled dataset for audio events. In *Proc. IEEE ICASSP 2017*, New Orleans, LA.

Härdle, W., S. Borak, and B. López-Cabrera (2013). *Statistics of Financial Markets: Exercises and Solutions*. Universitext. Springer Berlin Heidelberg.

He, K., X. Zhang, S. Ren, and J. Sun (2015). Deep residual learning for image recognition. *CoRR abs/1512.03385*.

Hu, G. (2015). A corpus of non-speech sounds. http://web.cse.ohio-state.edu/pnl/corpus/HuNonspeech/HuCorpus.html. [Online; accessed 03-Jun-2017].

Huzaifah, M. (2017). Comparison of time-frequency representations for environmental sound classification using convolutional neural networks. *CoRR abs/1706.07156*.

Kalchbrenner, N., L. Espeholt, K. Simonyan, A. van den Oord, A. Graves,

and K. Kavukcuoglu (2016). Neural machine translation in linear time. *CoRR abs/1610.10099*.

Lorenz, D. and J. M. Meredith (1999). Digital cellular telecommunications system (Phase 2+) (GSM); Full rate speech; Transcoding (GSM 06.10 version 8.1.1 Release 1999). https://portal.etsi.org/webapp/workprogram/Report_WorkItem.asp?WKI_ID=11074. [Online; accessed 25-Jun-2017].

Luong, M., H. Pham, and C. D. Manning (2015). Effective approaches to attention-based neural machine translation. *CoRR abs/1508.04025*.

Lyon, J. (2013). Mel Frequency Cepstral Coefficient (MFCC) tutorial. http://practicalcryptography.com/miscellaneous/machine-learning/guide-mel-frequency-cepstral-coefficients-mfccs/. [Online; accessed 10-Jan-2018].

Makhoul, J. (1980, February). A fast cosine transform in one and two dimensions. *IEEE Transactions on Acoustics, Speech, and Signal Processing 28*(1), 27–34.

Mikolov, T., K. Chen, G. Corrado, and J. Dean (2013). Efficient estimation of word representations in vector space. *CoRR abs/1301.3781*.

O'Shaughnessy, D. (1987). *Speech communication: human and machine*. Addison-Wesley series in electrical engineering. Addison-Wesley Pub. Co.

Pearce, D. and C. d'Esclercs (2003). Speech Processing, Transmission and Quality Aspects (STQ); Distributed speech recognition; Front-end feature extraction algorithm; Compression algorithms). https://portal.etsi.org/webapp/workprogram/Report_WorkItem.asp?wki_id=18820.

Reddi, S. J., S. Kale, and S. Kumar (2018). On the convergence of adam and beyond. In *International Conference on Learning Representations*.

S Stevens J Volkmann and E Newman (1937). A scale for the measurement of the psychological magnitude pitch. *Journal of the Acoustical Society of America 8*(3), 185 – 190.

Skerry-Ryan, R. J., E. Battenberg, Y. Xiao, Y. Wang, D. Stanton, J. Shor, R. J. Weiss, R. Clark, and R. A. Saurous (2018). Towards end-to-end prosody transfer for expressive speech synthesis with tacotron. *CoRR abs/1803.09047*.

van den Oord, A., S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. W. Senior, and K. Kavukcuoglu (2016). Wavenet: A generative model for raw audio. *CoRR abs/1609.03499*.

Wang, A. and Th-Floor-Block-F (2003). An industrial-strength audio search algorithm. In *Proceedings of the 4 th International Conference on Music Information Retrieval*.

Zheng, F., G. Zhang, and Z. Song (2001, Nov). Comparison of different implementations of mfcc. *Journal of Computer Science and Technology 16*(6), "582–589".

Declaration of Authorship

I hereby confirm that I have authored this master thesis independently and without use of others than the indicated sources. Where I have consulted the published work of others, in any form (e.g. ideas, equations, figures, text, tables), this is always explicitly attributed.

Berlin, June 18, 2018

David Pollack