CIS 520 Fall 2022, Project Report

Understanding Classical Music with Machine Learning

Team Members:

- Xander Uyttendaele; Email: xanderu@seas.upenn.edu
- Ayanav Roy; Email: ayanav@seas.upenn.edu

home pod: lovable-lemur

Abstract

We explore the problem of pitch detection in audio files using machine learning. Given a 0.3-second audio snippet from the MusicNet dataset, our model attempts to detect which of 45 possible pitches are present at the midpoint of the audio clip. We thus formulate our problem as a multi-class, multi-label classification problem. Drawing on traditional techniques from signal processing, we try a variety of machine learning models on this problem, including a sequence-based model, and compare the results against a baseline approach which directly uses the results of the constant-Q transform in order to make its prediction. We evaluate our models using precision, recall, and AUC metrics, and find that all models except the RNN significantly beat the baseline, with our CNN performing the best.

1 Motivation

Music Information Retrieval (MIR) is the field of computer science dealing with processing and understanding music, and includes a variety of problems that are well-suited to be solved via machine learning methods. In this paper, we tackle the problem of pitch detection given an uncompressed audio file as input. That is, we try to determine which pitches are being played at each instant of time within an audio recording.

We believe that this problem is an ideal candidate for a machine learning based solution. First and foremost, accurate pitch detection is a surprisingly difficult task for computers to perform on real-world data, but that well-trained musicians can easily do. Moreover, noisy data is a key reason why this problem is hard for computers: if the input data were a MIDI track (a type of computer-generated music file), then extracting the pitches from the audio would be relatively straightforward; however, recordings of real instruments playing live music are significantly noisier than this, and thus a rule-based pitch-detection system works much less reliably. Finally, machine learning techniques are already used widely in the broader audio processing community (for instance, for use in applications that involve parsing and interpreting speech such as Alexa), and thus we see no reason why such methods would not find a similar level of success with music. For all of these reasons, we believe that machine learning methods are likely to perform well on the task of pitch detection.

We also believe that pitch detection is an important problem to solve, as it is a crucial stepping stone toward the solution of more difficult problems within MIR. For instance, the problem of automatic music transcription—where a recording of music is converted into human-readable sheet music representing that song—crucially requires pitch detection as a first step in its pipeline. On a personal note, being musicians ourselves, we also find this problem fascinating simply because it allows us to explore questions of music from a scientific perspective.

Traditionally, problems in MIR have been approached from a signal processing perspective, where windowed Fourier transforms such as the short time Fourier transform and the constant-Q transform are used to extract spectral information from an audio signal, and then this spectral information is processed according to some rule in order to extract information from the music. In this paper, we attempt to add machine learning onto this traditional framework, still allowing ourselves to use windowed Fourier transforms to initially process the data, but then solving the remainder of the problem using machine learning. We thus

attempt to understand the strengths and limitations of machine learning in this domain. For this reason, we try to solve the problem using a variety of different machine learning methods, comparing the results of our experiments against a baseline that uses the more traditional techniques of MIR.

While we focus primarily on supervised learning techniques within this paper since we are lucky enough to have access to a large dataset of pitch-annotated audio files, we also attempt a semi-supervised approach based on PCA. Perhaps surprisingly, this semi-supervised method turns out to be one of the better-performing and most interpretable models we train, although ultimately traditional deep learning based techniques such as convolutional neural networks do turn out to be the most successful at the problem of pitch recognition.

2 Music Information Retrieval Background

How precisely are audio files represented digitally? A .wav file is the simplest such type of digital audio file, being an uncompressed representation of an audio signal (in contrast to compressed audio files such as MP3s). Since uncompressed audio naturally contains the most information about a signal, and as good machine learning engineers our job is not to throw away any potentially relevant information, we use .wav files as the input to our machine learning models in this paper. A .wav file can be thought of as simply an array of values representing the amplitude of an audio signal at a set of discretized moments in time. The sampling rate f_s (included in the metadata of a .wav file) tells us the number of samples of audio taken per second, and thus $1/f_s$ is the time between samples. The standard choice for f_s is 44100 samples/second, since this is a sufficiently high sampling rate to represent any frequency of sound that the human ear can perceive.

Next, it is important to understand what music looks like within an audio signal, so that we can structure our algorithms with this in mind. Each pitch on the scale is characterized by a unique fundamental frequency that represents the number of times the sound wave corresponding to that pitch oscillates per second. Our ears have evolved to extract these frequencies from sound waves, and thus hear music. Importantly, frequency scales exponentially with pitch (so for instance a C4 has twice the frequency of a C3). Moreover, humans perceive sound in decibels, which is a similarly logarithmic representation of the loudness of a signal. Beyond this fundamental frequency, the only reason that a violin sounds different when playing an A4 versus a piano is due to the fact that these instruments have different overtone profiles, which correspond to higher-frequency components present in the signal. However, importantly these overtones will only occur at integer multiples of the fundamental frequency. We keep this physical structure of music in mind as we design our algorithms in this paper, starting with a discussion of how to extract frequency information from an audio signal below.

Once we have a .wav file, the traditional first step in most MIR applications involves first running the audio file through some sort of windowed Fourier transform in order to extract relevant spectral information from the signal. In this paper, we also take this as a first step in preprocessing our data, and thus we now briefly describe how precisely windowed Fourier transforms work. For a more detailed description, see the excellent book on MIR by Muller [5].

The discrete Fourier transform is the simplest type of spectral transform that one can apply to an audio signal, converting a time-domain audio signal into an equivalent spectral-domain representation. Given an audio signal, it determines how much of each frequency is present in the signal. However, the discrete Fourier transform operates on the entire audio signal at once (thus giving a spectral representation that only tells us the average amount of each frequency present across, say, an entire 7-minute-long song). For applications that want more localized information about the frequencies present, we thus need something more sophisticated.

In order to deal with this problem, the short time Fourier transform (or STFT) was developed. The STFT is the simplest example of a windowed Fourier transform: instead of operating on an entire audio signal at once, it instead splits the signal up into a series of smaller signals, and performs the discrete Fourier transform on each of these shorter signals. The output of the STFT is called spectrogram, and can be visualized as an image with time on the x-axis, and frequency information on the y-axis.

The key hyperparameter of the STFT is the length of window we consider. Due to the properties of the discrete Fourier transform, the shorter the length of input, the less fine-grained the resulting frequency-domain output will be (more formally, this is because the Fourier transform is a unitary operator, and thus conserves information). However, for pitch information that scales exponentially in the frequency domain,

we need a higher resolution of frequency information for lower pitches (whose frequencies lie closer together in absolute distance) than we do for high pitches (whose frequencies lie farther apart). To deal with this issue, we use what is called the constant-Q transform, which was originally introduced in [1]. The constant-Q transform works by performing a series of STFTs with different window lengths, then combining the results of these STFTs into a single spectrogram that has a higher resolution of frequency information for lower frequencies. For the reasons discussed above, the constant-Q transform is ideally suited to dealing with audio signals containing music, and thus we choose to use it in this paper.

3 Related Work

With this background in mind, we now survey a number of previous attempts to solve the problem of pitch detection. As we mentioned above, the traditional approach to pitch detection involves first running the audio signal through a windowed Fourier transform such as the STFT or constant-Q transform, then interpreting this spectral data directly in order to predict the presence or absence of various pitches. For instance, [4] outlines a number of these traditional techniques, many of which roughly amount to simply predicting pitches to be at the locations of the maxima of the spectrogram at each point in time.

It is worth mentioning that some traditional techniques also operate solely in the time domain (for instance the super simple method of simply counting the number of zero crossings of a signal and using that as the fundamental frequency works surprisingly well), however those that operate in the frequency domain have been found to work better, especially on signals where multiple pitches may be playing at the same time. Of course, the primary benefit of these traditional methods is their conceptual simplicity and the fact that they do not require a large dataset in order to train. The disadvantage of this approach, however, is that these techniques tend to not be particularly robust to noisy data, and that these baseline methods do not tend to account for overtones within the data, and thus may be prone to falsely detecting pitches that are really just overtones.

Moving past these traditional methods, now, we next consider work that is been done in applying machine learning to the problem of pitch detection. Incidentally, the paper [7] that originally introduced the MusicNet dataset that we use in this paper also attempts to solve the problem of pitch detection using machine learning. In particular, [7] treats the problem as a multi-label classification task in which a given segment of audio is associated with a vector of labels indicating which pitches are present in the audio segment. The authors try a ridge regression, a multi-layer perceptron, and a convolutional neural network on the raw audio data, finding respectable but far from perfect results (the best result they achieve is a precision of 60.5% and a recall of 71.9%).

Interestingly, although they compare their results against a spectrogram-based baseline, all of the machine learning methods used in [7] operate exclusively in the time domain. Although presumably given the right model architecture, such time-domain techniques can produce good results, we had significant trouble replicating these time-domain models in this paper, while we had relatively little trouble achieving comparable results using machine learning methods in the frequency domain. Coupled with the success of spectrogram-based machine learning techniques for other applications of deep learning to audio signals [6], we thus focus on machine learning models trained in the frequency domain in this paper.

The benefit of the time-domain approach presented in [7] is that it does not rely on any sort of audio preprocessing, and is this conceptually simpler than our method. However, one limitation of the approach taken in [7] is that it does not take into account the sequential nature of the dataset, treating adjacent audio clips (which are likely to be highly correlated in pitches) as independent. In this paper, we tried to exploit this sequential information by trying an RNN-based model.

Of course, moving beyond pitch detection, machine learning has been used to make strides on many different problems within MIR. For instance, [2] uses a transformer-based approach on the MusicNet dataset in order to train a model that performs automatic music transcription, achieving state of the art performance. The details of these more complicated models are, however, safely outside the scope of this paper.

4 Data Set

In this paper, we use the MusicNet dataset in order to train our models. The MusicNet dataset[1] contains 330 live recordings of classical music pieces from 8 different composers, along with detailed labellings of what notes are being played in the recording at each moment in time. The dataset was originally introduced in [7], with the proposed use cases of identifying notes, classifying instruments or composers, or predicting the next note given the previous notes. The note labels were found by performing a dynamic time warping algorithm [5] in order to align the live recording against a known MIDI file for each piece, which thus gives a mapping from time in the original recording to what notes are being played at that time. Certain sections of the dataset were then verified by professional musicians in order to be sure that these labellings were correct, ultimately estimating an error rate of only 4%. In total, the dataset contains over 1 million labeled notes and is 33 GB in size.

We choose a train-test split of 320/10 songs, as was done in the original MusicNet paper. Although for small datasets such a skewed train-test split may be a bad idea, the MusicNet dataset is sufficiently large that we may do this. Importantly, we split the dataset according to songs, as opposed to mixing examples from the same song in both the training and test set, since there is likely to be significant correlations between pitches at different times in the song. For example, the songs in MusicNet often have a constant key signature throughout that implies only a subset of the notes will consistently be played.

Processing a dataset of this size turned out to be a key challenge for this project. Initial processing of the dataset was relatively straightforward, consisting of simply loading each song, slicing it into appropriate-length pieces, and then creating a multi-hot encoding label vector corresponding to the pitches being played during each of these slices. However, the dataset is sufficiently large that it cannot be loaded into our 12GB RAM at once, and thus training models on the time domain data proved to be especially challenging. In particular, training data had to be loaded into memory sequentially with the help of a custom PyTorch dataloader class we created. However, this dataloader turned out to be extremely slow, making even training a logistic regression intractable. Ultimately, we were forced to only consider a subset of our dataset that could be loaded into RAM at once in order to train these time domain models.

In order to train spectral domain models, we next ran each of our songs through a constant-Q transform, then sliced the resulting spectrogram into 0.3-second chunks just as we did for the time domain data above. We then additionally converted the spectrogram values to a decibel scale, and saved this transformed dataset as a .npy so that the models we trained could load this preprocessed dataset directly. Training models on the constant-Q-transformed data proved easier, since the constant-Q transform also served as a large dimensionality reduction on the data, and thus the transformed dataset could be loaded into our RAM all at once.

Each of these preprocessing steps took a significant amount of time to complete (on the order of 15 minutes to an hour in Colab), and thus progress was very slow through this part of the project.

Since we perform pitch detection on 0.3-second audio clips (as described in more detail in the Problem Formulation section below), once we split each song into pieces of this length we are left with $n = 326580$ training examples and $n = 3981$ testing examples. For methods that operate in the time domain, each of these clips has $p = 16384$ features (since our sampling rate is 44100). However, if we instead choose to work in the spectral domain, applying a constant-Q transform to the data, then we are left with the same ns but a much smaller $p = 2688$ features (assuming default parameters on the constant-Q transform).

Importantly, as we can see in Figure 1, the distribution of note occurrences in MusicNet is far from uniform. Unsurprisingly, middle C is the most common note, and chromatic notes are less common than diatonic notes (since the pieces of classical music in MusicNet are almost all written in "normal" key signatures like C major that do not use these chromatic notes as often). Very low notes and very high notes never occur in MusicNet, and outside of the range of MIDI notes between 40 and 84 or so, occurrences in general are very low. Since we are unlikely to successfully train a classifier to detect a certain pitch with no training data at that pitch, we choose to only train models that attempt to detect pitches within this range of sufficiently frequently occurring notes, and thus our label vectors are ultimately only 45-dimensional.

[1] https://www.kaggle.com/datasets/imsparsh/musicnet-dataset?datasetId=1167622

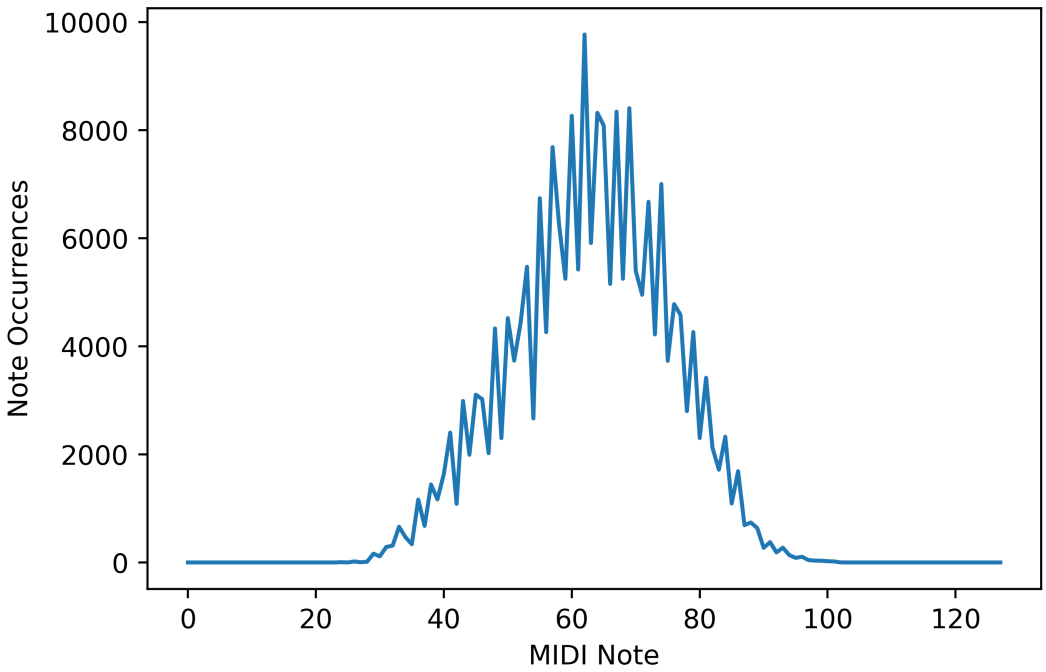

Figure 1: Note occurrences versus pitch in the MusicNet dataset. Interestingly, the distribution is far from uniform, with significant differences in occurrence number even for adjacent pitches.

5 Problem Formulation

We formulate our problem of pitch detection as a multi-label, multi-class supervised learning task. In particular, for ease of comparison we will formulate the problem as in [7], dividing each audio file into 0.3-second segments and labelling a pitch as present if it is being played at the center of the clip. Our labels thus look like 45-dimensional multi-hot vectors. Regardless of the details of our models, the final layer always consists of 45 outputs, representing a vector of probabilities for which pitches are being played in that audio segment. Since this is a multi-label classification task, this output vector need not be normalized. And, as we discuss in our later Experiments and Results section, there's a heavy class imbalance in this problem, meaning that simply using accuracy isn't enough. Metrics like precision and recall are crucial to evaluate how well our models learned certain pitches.

With this formulation, along with the additional constant-Q transform preprocessing described in the previous section, we then trained a variety of machine learning models on our dataset. As is standard for multi-class, multi-label problems, for our loss function we use the binary cross-entropy (i.e., the KL-divergence between the true labels and the predicted labels for a given pitch). This choice is so standard (and theoretically well-grounded due to the properties of KL-divergence) that we do not consider other options for loss function in this paper. We tried using both normal stochastic gradient descent and the Adam optimizer in order to train our models.

In addition to our basic formulation treating the problem as a simple classification problem, we also formulate the problem in a slightly different way in order to train an RNN to detect pitch. In this framework, we arrange the data in sets of 20 contiguous constant-Q transformed 0.3-second snippets within a song, and attempt to predict the pitches playing in the final of these 20 clips using an RNN. Thus, our problem is still a multi-class, multi-label classification problem, just with a slightly different dataset structure. Formulating our problem in this particular way makes intuitive sense, since it mimics how humans naturally listen to music (i.e., we don't normally try to detect pitch by simply listening to a single 0.3-second clip of audio, but instead a sequence of chords). Moreover, since adjacent notes in a piece of classical music are likely to be highly correlated, we expect this method to work slightly better than our methods which do not use

sequential information in their predictions.

6 Methods

Since our problem is naturally a supervised learning problem on very high dimensional noisy data, deep learning techniques make the most sense to try. In addition to our baseline model, we tried a variety of machine learning methods with different model complexities in order to get a better sense of how hard this problem really is and what works best.

6.1 Baseline Method

For our baseline method, we used the constant-Q transform directly, predicting a pitch as present if the relevant frequency level in the transformed audio signal is sufficiently energetic. That is, given a constant-Q spectrogram representing an audio clip, we take the sum across each frequency band to calculate the amount of signal at each frequency, then mark a pitch as present if this summed signal strength at the frequency corresponding to that pitch is greater than some threshold. We implement the constant-Q transform in Python using the Librosa package, and compute all the other relevant array operations using Numpy.

We believe that our choice of baseline model makes sense as a comparison against our machine learning models, because it adopts the approach typically used within MIR before the advent of machine learning algorithms (as detailed in the MIR Background section). Moreover, although our baseline is quite simple, since the constant-Q transform encodes lots of spectral information with no further processing, our baseline is reasonably successful on the problem of pitch detection, as we see in the Results section.

6.2 Time Domain Logistic Regression

As the simplest possible approach to this problem, we attempt to fit a logistic regression in scikit-learn directly on our 0.3-second audio clips. As we mentioned above, this uncompressed form of our dataset is too large to fit into our RAM all at once (and scikit-learn's logistic regression function cannot handle datasets more than a GB or so in size without crashing), and thus we were forced to train the logistic regression on a dataset with only a tenth of the original data points. Presumably with a sufficiently powerful computer and enough RAM, one could learn Fourier-transform-esque functions on this raw data that compute pitch information. However, with our more limited capabilities, we were unable to fit a logistic regression that performed any better than randomly on our test set. Thus, we include this discussion only for completeness.

Nevertheless, trying to fit a model on the time domain data was worthwhile, as it gave us a sense of the difficulty of the problem without using such tools as the constant-Q transform.

6.3 Spectral Domain Logistic Regression

With the failure of our time domain logistic regression in mind, we pivoted to trying to fit models only after first preprocessing our data with the constant-Q transform. Thus, given an 84x32-dimensional spectrogram array representing a 0.3-second audio clip, we fit a logistic regression to detect pitches. Since even with this dimensionality reduction scikit-learn's logistic regression function could not handle a dataset of this size, we fit the logistic regression using stochastic gradient descent in PyTorch.

With minimal training, we saw strong performance of our logistic regression model on the test set, beating the baseline and thus justifying our use of machine learning on spectral domain data.

6.4 Spectral Domain CNN

With the success of our spectral domain logistic regression in mind, and seeing no signs of overfitting on the test set, we thus increased our model complexity and attempted to train a CNN on our constant-Q transformed data. A CNN in particular makes sense as a model architecture on this data (as opposed to a multi-layer perceptron, say) because spectrograms are invariant to shifts in time and frequency, and thus models that use parameter sharing are likely to perform well. Said another way, a model trained to detect the presence of a C4 given a spectrogram is likely to look very similar to a model trained to detect an A4,

just with each weight translated by the same amount along the frequency axis. Moreover, a model trained to detect a C4 at time t_1 is likely to look very similar to a model trained to detect a C4 at t_2. Thus, the parameter sharing used by a CNN makes sense.

We trained a CNN in PyTorch on our dataset, and once again saw significant improvements in performance on the test set, giving comparable results to the MusicNet paper [7] with few signs of overfitting, suggesting we could in theory increase the model complexity even more, although the training time required to train a model of this size was already quite significant, and thus we did not have time to train a larger CNN.

6.5 Spectral Domain RNN

Finally, in order to exploit the sequential structure of our audio data, we trained an LSTM-based RNN on our dataset, restructured as described in the Problem Formulation section. This didn't go as planned. No matter the learning rate of the model, if given enough epochs, it would always converge to concluding that there were no pitches at all for any of the samples. This occured with the LSTM and a vanilla RNN we also tried, which both gave us 0 precision and recall with this model. And an early stopping caused the model's accuracy to plummet, while not matching the precision or recall of our less complicated models.

We discuss possibilities for these disappointing results and possible next steps in the later Conclusion and Discussion section.

6.6 Time Domain PCA

Having now completed our experiments using traditional deep learning methods, we attempted to try something more novel by performing PCA on our time domain audio data, then training the resulting embeddings in a semi-supervised manner.

As with our time domain logistic regression, semi-supervised learning did not work very well. However, the "eigen-audios" learned from PCA to create the embeddings were quite interesting and interpretable, as we discuss in the Conclusion and Discussion section and demo notebook. We performed all of this in scikit-learn, and thus once again were forced to operate on a dataset of reduced size to avoid running out of memory.

6.7 Spectral Domain PCA

As a final experiment, we attempted to perform PCA on our constant-Q transformed data, and then train a model on the resulting embeddings in a semi-supervised manner. Although the resulting "eigen-CQTs" that were learned from PCA were significantly less interpretable than for our time domain embeddings, semi-supervised learning performed much better in this case, even beating our logistic regression model on the test set.

7 Experiments and Results

Since this is a multi-label classification problem on unbalanced data, we must consider our evaluation metrics carefully. If we were to simply use accuracy, then since any given note is not being played the vast majority of the time within a song (and thus the vast majority of the time within a given audio clip) we would achieve an accuracy of the order of 90-95% (with fluctuations depending on how common the pitch is) simply by predicting that the note is never being played. Thus, it makes sense to instead calculate precision and recall for each note on the test set. Moreover, since precision and recall are dependent upon the (somewhat arbitrary) threshold value chosen for classification, we compute the pitch-wise AUC on the test set as well. Since this results in 45 different precisions, recalls, and AUCs, we report our summary statistics for each model as the average precision, recall, and AUC over all the notes on the test set, which we believe gives a strong indication of the model's overall performance. The average AUC is ultimately what we use to decide whether one model is better than another, since we believe this is the best single-valued statistic for assessing model quality on an unbalanced dataset. We also compute ROC curves on the test set for middle C in particular for visualization purposes, as we will see in the Results section.

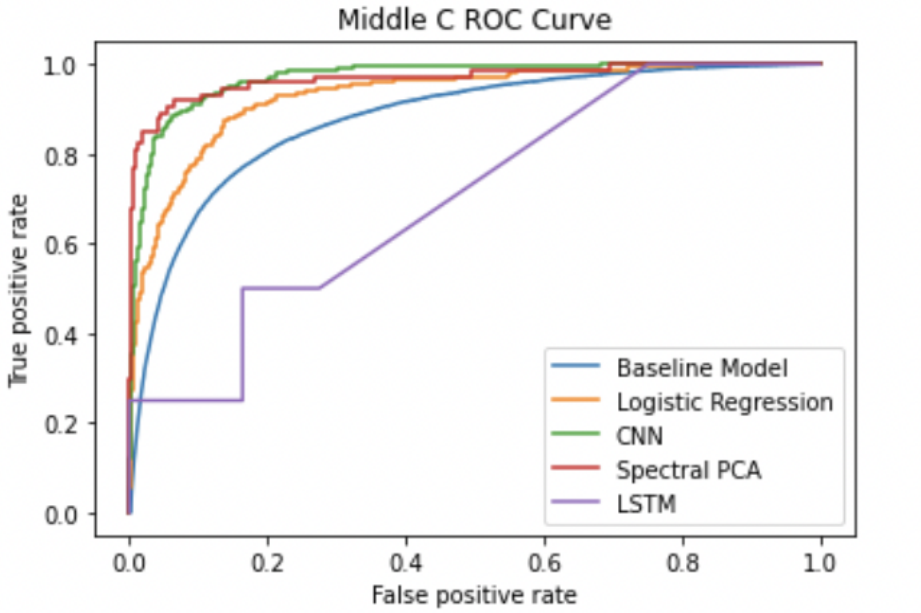

Figure 2: The ROC curve on Middle C for all of our methods, compared against our baseline. Except for our failed LSTM attempt, all machine learning methods considerably beat the baseline, with the CNN performing the best.

With these performance metrics chosen, we train all our models using our 320-song training set, then evaluate the performance on our 10-song test set. Since the amount of hyperparameter tuning we perform is relatively minimal, we do not feel the need to add an additional validation set to avoid overfitting on the test set.

We now give a brief summary of our exact model architecture for each method we tried. For our baseline model, the only hyperparameter was the threshold, for which we found that a value of -10 gave a good balance of precision and recall. For our spectral domain logistic regression, we found the best performance with 30 epochs of training using an Adam optimizer with learning rate 1e-5. For our CNN, we trained for 5 epochs using an Adam optimizer with a learning rate of 1e-3 (please see our source code for the layer-by-layer structure of the CNN). For our RNN, we tried multiple different epoch and learning rate combinations, but the one shown here had a lr of 1e-5 trained using an Adam optimizer for 5 epochs. Finally, for our spectral domain PCA, we ran semi-supervised learning on only the first 128 principal components of the dataset by training a logistic regression on these embeddings in scikit-learn with a max_iter parameter of 500. The results of our experiments using these methods can be found summarized in Table 7, and the ROC curves for middle C are plotted in Figure 2.

	Average Precision	Average Recall	Average AUC
Baseline	0.353	0.296	0.834
Spectral Logistic Regression	0.393	0.702	0.929
CNN	0.522	0.718	0.960
RNN	0	0	0.512
Spectral PCA	0.715	0.380	0.936

Table 1: Experimental results of our methods on the test set, using the hyperparameters given in the text body.

8 Conclusion and Discussion

As we can see from Table 7, our best performing model was our CNN, with an AUC of 0.960. This more or less matches our expectation, and provides additional evidence for our above argument that parameter sharing across a spectrogram makes sense. The relative ordering in performance of our other non-sequential models also mostly matches our intuitive expectations, with more complex models generally performing better. The only exception is our spectral PCA, which performs surprisingly well (slightly outperforming our logistic regression) despite being a strictly simpler model. This highlights the power of PCA to find structure within data, and underscores the utility of semi-supervised learning in general.

We were surprised at our RNN model's inability to learn pitch detection. We have a few guesses as to the reason. One is that our larger sequence length of 20 for our data caused there to be not enough samples for our model to train on, and perhaps a smaller sequence length of 8 might have performed better. Another possibility is that the heavy class imbalance is something that affects RNNs more than CNNs. Perhaps only training for pitch detection of a certain note and oversampling our training set, would give us better results. Examining these possibilities would be interesting extensions for the future. RNNs are often used for music composition [3], so we still believe that there is potential to make this model work.

Although we did not achieve strong performance on the time domain PCA, the resulting principal components learned from this method were perhaps the result that provided the most insight into our problem. In particular, when we ran PCA on a subset of the data consisting only of songs in the key of A (since PCA runs out of memory if run on our entire dataset at once), we found that our principal components—which have the shape of the original 0.3-second audio data and thus can be played as an audio file—corresponded to chords within the key of A (the first principal component being a fifth, the second being an octave, and so on). We found this to be a very interesting result, and an excellent example of PCA extracting meaningful structure from raw data in an unsupervised manner. To listen to these eigen-audios, see our code notebook demo.

The primary lesson we learned from this project was that processing large datasets can be extremely difficult and time-consuming. Even simply managing a dataset taken directly from Kaggle, we faced significant challenges processing and loading this dataset into memory in a way that allowed for efficient model training. Ultimately, the solution to this problem came in the form of the dimensionality reduction provided by the constant-Q transform. However, we tried many other techniques to try and make the time domain more tractable before resorting to the constant-Q transform. For instance, we built a custom PyTorch Dataset class meant to load our time domain data dynamically, but this turned out to be too slow for our purpose. This project gave us a much deeper appreciation for the data scientists whose job it is to assemble these large high-quality datasets for the machine learning engineer to work with.

We also became significantly more familiar with the inner-workings of PyTorch through this project, as well as the importance of the correct data preprocessing methods when handling audio data.

Much remains to be done in applying machine learning to the field of MIR. In addition to trying even more complex models such as transformers on the problem of pitch detection and refining the ones we tried that don't quite work yet like RNNs, there are many additional problems within MIR that would surely benefit from a machine learning approach. The MusicNet dataset could easily be adapted to the train models on the problems of composer detection, note prediction, instrument detection, music generation, and more. All of this remains as exciting future work within this field.

Acknowledgments

Thank you to Matthew Pressimone for advising our group on this project.

References

[1] Judith C Brown. Calculation of a constant q spectral transform. *The Journal of the Acoustical Society of America*, 89(1):425–434, 1991.

[2] Andis Draguns, Emīls Ozoliņš, Agris Šostaks, Matīss Apinis, and Karlis Freivalds. Residual shuffle-exchange networks for fast processing of long sequences. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 7245–7253, 2021.

[3] Douglas Eck and Juergen Schmidhuber. A first look at music composition using lstm recurrent neural networks. *Istituto Dalle Molle Di Studi Sull Intelligenza Artificiale*, 103:48, 2002.

[4] David Gerhard et al. *Pitch extraction and fundamental frequency: History and current techniques*. Department of Computer Science, University of Regina Regina, SK, Canada, 2003.

[5] Meinard Müller. *Fundamentals of music processing: Audio, analysis, algorithms, applications*, volume 5. Springer, 2015.

[6] Hendrik Purwins, Bo Li, Tuomas Virtanen, Jan Schlüter, Shuo-Yiin Chang, and Tara Sainath. Deep learning for audio signal processing. *IEEE Journal of Selected Topics in Signal Processing*, 13(2):206–219, 2019.

[7] John Thickstun, Zaid Harchaoui, and Sham Kakade. Learning features of music from scratch. *arXiv preprint arXiv:1611.09827*, 2016.