Monophonic Pitch Tracking Using a Convolutional Neural Network

Koye Alagbe
Computer Science Student
Stanford University

Abstract

Pitch tracking is the ability to identify the frequency of a sound signal in real time. Being able to do this quickly and accurately has many uses across different areas of both speech recognition and music processing. In music, each note is determined by the frequency of its sound, so identifying frequencies digitally allows computer programs to identify notes. This has many applications, such as in digital tuners and in musical transcription.

People with chromesthesia map sounds, such as musical notes, to colors. For example, an A on a piano might "sound" blue. These color mappings are involuntary, consistent, and unique, meaning the same note always gets mapped to the same color, and two people with chromesthesia would likely not have the same color mappings. In this project, I attempt to use monophonic pitch tracking to build an application that allows people to visualize the experience of having chromesthesia. The color mappings used in the application were obtained from a person with chromesthesia. The pitch tracking is handled by a convolutional neural network that operates on fast Fourier transforms of sound data, trained using examples from various instruments of all 88 musical notes on a piano.

1 Introduction

The word monophonic refers to sounds with only one frequency being played at a time. Monophonic pitch tracking is quite different from polyphonic pitch tracking, or tracking multiple pitches at once. Monophonic pitch tracking plays more of a role in speech recognition, in applications where one voice is being tracked. In music, monophonic pitch tracking is widely used in transcription or melody extraction applications. The ability to read a sound signal and identify the pitch has many uses, and it is still being explored today.

Chromesthesia is a rare condition, affecting only about 1 in 3,000 people. For this project, I worked with a subject with chromesthesia, and I was told that for someone with it, seeing colors is a regular part of the experience of listening to music. The specific colors that are seen seem arbitrary, and they differ from person to person. For this project, I wanted to create a way both for synesthetes to be able to play a sound and see what they already see in their minds shown back to them, and for non-synesthetes to visualize what having chromesthesia might be like.

To do this, I split the process of pitch tracking into two parts: segmentation and identification. A model that tracks pitches needs to be able to both "know" when one note ends and a new one is played, and be able to identify that note. Since these are two

different problems, I address pitch identification in this project, and I circumvent the need for segmentation by treating every sound signal as a series of small (64ms) slices. I then treat every slice as a single note and use the model to identify its pitch. In the demo application accompanying this project, these slices are taken in through the microphone, and by using the model on the audio slices one after another, I am able to approximately track the pitch of the incoming signal.

2 Related Work

The model in this project is in part based on the CREPE pitch tracking model (https://github.com/marl/crepe). CREPE stands for convolutional representation for pitch estimation, and it uses a convolutional neural network that operates directly on waveform audio data to produce its output. The inputs to the CREPE model are 64ms sound slices and the output of the model is a one-hot encoded vector with 360 entries. Each of the 360 entries corresponds to a specific pitch, with possible pitches spanning the six octaves from C1 to B6. The CREPE model outputs pitches with 20-cent intervals between notes. A cent is one hundredth of a semitone, which is the interval between two adjacent notes on a piano. Therefore, in the output of the CREPE model, five output nodes span a single semitone.

The model developed in this project differs from the CREPE model in some significant ways. Firstly, this model is trained with examples covering all 88 notes on a piano, from A0 to C8. This corresponds to the frequencies 27.50 Hz and 4186.01 Hz. For the sake of simplicity, this model also does not include ouput classes at 20-cent intervals between notes. Instead, it has just 89 output classes, one for each of the 88 notes, and an additional class for silence. The purpose of this choice was to increase efficiency and reduce overall model size and training time, since I did not need that level of precision for my intended application.

Secondly, this model does not operate directly on sound data. Instead, it uses fast Fourier transforms of the sound data. Fourier transforms decompose time=series data into its frequency-series data. This means that if the original graph of the sound plots the amplitude at different times, the Fourier transform will plot amplitude at different frequencies. The peaks of this graph occur at the frequencies that are most prominent in the original sound. Because this data provides more information about the frequency of the inputted sound than the raw sound data, using a Fourier transform as input greatly reduced training time and improved performance.

3 Dataset and Features

This project makes use of the NSynth dataset. This dataset contains synthesized audio files of 305,979 musical notes, spanning all 128 MIDI pitches and 1,006 unique instruments across 11 instrument families, namely bass, brass, flute, guitar, keyboard, mallet, organ, reed, string, synth lead, and vocal. Each audio file, stored as a waveform audio file (.wav), lasts for 4 seconds at a sampling rate of 16 kHz. This means that each file contains 64,000 individual audio samples, stored as floating-point numbers that represent amplitude over time. The dataset is split into training, cross validation, and test sets with 289,205, 12,678, and 4,096 samples respectively, and each of these datasets contains an accompanying JSON file that lists information such as pitch, instrument family, and qualities for each audio file.

Not every file from the NSynth dataset was used for this project. Audio files were selected such that roughly the same number

of training examples was chosen from each pitch and from each of the 11 instrument families. Preprocessing involved randomly selecting audio files from the set and obtaining random 1,024-sample slices from them. Note that at a sampling rate of 16 kHz, this is equivalent to 64ms of audio. After each slice was selected, Fourier transforms were applied to each one, cutting the input length in half. The final length of each input example was 512 values, representing the Fourier transform of a 64ms sound slice.

4 Methods

The final model used for this project is a convolutional network consisting of an input layer with 512 neurons, a one-dimensional convolutional layer with 256 filters, a kernel size of 32, and a stride of 2, with the input to this layer padded so that the output has the same dimensions as the input. This layer uses the ReLU activation function and is followed by a one-dimensional max pooling layer with a pool size of 2, also padding so that the input and output match dimensions. The next layers are two fully connected layers of 1,024 neurons each, both with ReLU activation. Finally, the output layer consists of 89 neurons and uses the softmax activation function. During training, each layer other than the output was followed by a dropout layer with a dropout rate of 0.15. The optimizer used in training was the Adam optimizer with learning rate decay and a batch size of 50, optimizing the categorical cross entropy loss function. Lastly, the input to the model was normalized to be in the range (-1, 1), and all of the weights were initialized with He initialization.

Over the course of this project, this pitch identification model went through many iterations. At the start of the training process, I did not apply Fourier transforms to the sound data, so the raw waveform data was the direct input to the network. My original idea was that this might help the network "form its own conclusions." I did not want to lose too much information right away.

At the start, the model was a fully connected network with just two layers of 128 neurons each. This model quickly ran into problems with underfitting. I experimented with increasing both the number of layers and the number of units per layer, which slightly decreased validation error. However, the increase in performance was relatively small, never breaking 50% validation accuracy, especially considering the large increase in training time caused by the increase in model size.

From here, I switched to a convolutional model, more similar to the CREPE model. Once again, the model started out small, and I added layers and neurons to try and combat underfitting. In this stage, I first tried to make the model overfit the training set, which it was eventually able to do. I then introduced regularization in the form of dropout to the model to try and improve performance on the validation set. The recurring problem was that the validation performance would eventually stop improving during training. When that happened, I tried to add layers and neurons to increase the learning capacity of the network. I experimented with adding convolutional layers, fully connected layers, or both to see what yielded the best performance. When I started to notice overfitting again, I also experimented with both L1 and L2 regularization with different regularization parameters, and with changing the batch size. None of this seemed to make a significant difference to validation performance, I eventually kept the batch size at 50 and removed the L1 and L2 regularizaiton. At a certain time, I was using a convolutional model with six convolutional layers (each followed by a max pooling layer)

and two fully connected layers, and achieving about 75% validation accuracy. At this point, training was very slow, and I noticed that the further addition and removal of layers did not improve validation performance. I began to investigate the input to the model.

The use of Fourier transforms on the data made a significant difference. With a much smaller convolutional model than the one without Fourier transformed inputs, I was able to achieve the same validation performance in only a fraction of the training time. Once again, after a process of adding and removing layers and neurons, as well as changing dropout rates, I was able to achieve the final model for this project, which was able to get an accuracy of about 80% on the validation set.

5 Results

The final model achieves a loss of 0.83 and an accuracy of 81.42% on the test set. A confusion matrix of the final result shows that the model frequently misclassifies notes at the highest and lowest ends of the input range. One reason for this could be that very high notes do not resonate as much as lower notes, and low notes resonate too much. Musical notes consist of a fundamental frequency and other frequencies that are integer multiples of that fundamental frequency playing simultaneously. When a note resonates very much, it is more likely for these other frequencies to be heard. This is why I believe lower notes can be harder to classify. This belief is corroborated by the subject with chromesthesia who helped me with this project, who also has perfect pitch. They said that the lowest notes on the piano are harder to classify, and it is harder to decide what color they see most clearly when these notes are played. This is one difference between the model in this project and the CREPE model, since the latter only uses input notes between C1 and B6, which I believe after seeing the results would improve the performance of the model.

With the final model, I was able to build the demo application to accompany this project. This application takes in input from the microphone and tracks the incoming signal in real time. It uses the pitch identification model to determine the pitch coming in and update the color of a circle every 64ms according to the specific color mappings of the subject I worked with. When I showed the demo to them, they became visibly excited, saying things like “This is exactly what I see in my head!” The subject could play, hum, or sing notes into the microphone and watch the color of the circle change accordingly. I consider this outcome a successful one, because one of the goals of this project was to provide an interesting way for people with chromesthesia to experience music. However, improvements can still be made to the model to improve its accuracy.

6 Conclusion

At the start of this project, the goal was to build a pitch identification model that could be used to track the pitch of a sound in real time and display colors according to the color mappings of a person with chromesthesia. The demo application was able to do that to a significant degree. One way to improve the accuracy could be to limit the input range to a set of notes more common in the real world since the notes at the lowest and highest ends of the range are harder to classify. One further application of this pitch identification model could be in an instrument tuner application that, similar to the demo application, would track the pitch of audio coming in through the microphone. This application would most likely require more precise output than simply the note, so the model could be changed to include pitches at

20-cent or 10-cent intervals between semitones, like the CREPE model. Monophonic pitch tracking has many more applications beyond chromesthesia and instrument tuning, and a model like this one can definitely be improved on and put to many different uses.

7 Contributions

I would like to give a very special thanks to my sister, Lota Alagbe, who has both perfect pitch and chromesthesia and agreed to help me with this project. She provided me with her color mappings for all 88 notes on the piano, and she gave me a lot of information on her experience with chromesthesia.

8 References

[1] Jong Wook Kim, Justin Salamon, Peter Li, and Juan Pablo Bello. "CREPE: A Convolutional Representation for Pitch Estimation." In 2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). IEEE, 2018.
The original paper on the CREPE model. A GitHub repository with a published version of the model can be found at https://github.com/marl/crepe.

[2] Jesse Engel, Cinjon Resnick, Adam Roberts, Sander Dieleman, Douglas Eck, Karen Simonyan, and Mohammad Norouzi. "Neural Audio Synthesis of Musical Notes with WaveNet Autoencoders." 2017.
The paper where the NSynth dataset was introduced.

[3] Palmer, Stephen E. "What Color is this Song?" Nautilus. 16 Jul 2015, < https://nautil.us/issue/26/color/what-color-is-this-song#:~:text=Such%20individuals%20have%20a%20neurological,hearing%20music%20and%20other%20sounds.&text=Chromesthesia%20is%20relatively%20rare%2C%20occurring,about%201%20in%203%2C000%20individuals>. Accessed 16 Mar 2021.
[4] Libraries and frameworks: Tensorflow (https://www.tensorflow.org/), Keras (https://keras.io/).