Introduction

Guitar notes classifiers have been around since the 1960s as tuners, that use direct signals from electric guitars, microphone signals or clamp-on vibration detectors. They traditionally use the Harmonic Product Spectrum (HPS) method to estimate the pitch.

However, this method wasn't originally intended to also predict the octave of the musical note and its accuracy is inconsistent when dealing with low notes down to A0 = 27.50 Hz, which have become part of an electric guitar's range in the last decade.

In this project, I will describe an extensible neural network-based approach to octave-aware, single chromatic note classification, and leveraging it in real-time operation.

Data Description

The dataset I chose as a starting point is the IDMT-SMT-GUITAR dataset (https://zenodo.org/records/7544110), which consists of raw DI electric guitar signals recorded at 24-bit / 44.1kHz, and I used the portion representing 3 octaves of picked notes on multiple fretboard positions. According to the dataset's authors, two guitars have been used ("Ibanez Power Strat" and "Fender Strat"), and 4 pickup selector positions in total.

The original dataset contains annotations for each audio file in XML form, stating the file name, a pitch label from 40 to 76, silence timestamps, fret and string numbers, and picking/expression techniques employed.

I have processed the XML documents for each pickup selector position into a single JSON file, containing a list of JSON objects, each containing the file names, pitch labels re-scaled to integers from 0 to 71 representing the A0 - G#6 range and silence timestamps.

After some experiments that I will mention later on, I have also recorded a dataset using my own electric guitar, that, coincidentally, is also of the Ibanez Power Strat family. I have recorded single notes on all 24 frets including open notes and all 6 strings picked at a medium velocity. Two iterations of all 150 positions have been recorded, one with the bridge humbucker pickup, and the other with the neck humbucker pickup. The duration of the samples ranges from 2 to 10 seconds, which will ultimately yield more training data than the other dataset. At the time of recording, the guitar was strung for C Standard tuning.

I designed the signal chain to be as noiseless as possible with the gear at my disposal. The guitar is plugged into a Fractal Audio AX8 unit that sends an unprocessed copy of the main instrument input through a balanced signal. This effectively acts as a DI box, a device strongly recommended in any guitar recording setup. The balanced DI signal is fed into an XLR analogue input of a Focusrite Scarlett 6i6 2nd gen audio interface, that acts solely as an A/D converter, since the signal is already pre-amplified by the Fractal unit, with the level manually adjusted to keep the signal peaks below but near 0dB. The Focusrite unit sends the 24-bit / 48kHz mono digital signal via USB, which is finally managed and recorded by the Presonus Studio One 6 DAW software.

The metadata for my own dataset samples has been similarly compiled in a JSON file as the ones from the other dataset, but I leveraged a custom file naming convention and computed the silence timestamps using the librosa.trim() function.

Feature Extraction

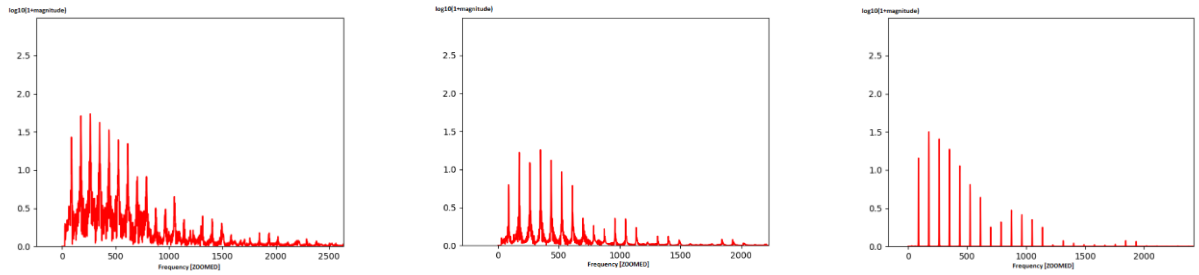
This being a pitch identification task, in theory, all that would be needed are the magnitudes of the fundamental frequency of the played note and its overtones. Therefore, some variant of the STFT would be necessary for the features on which we'll train the classifier.

Since we're aiming to classify notes up until G#6 = ~1660 Hz, we'd want at least 3 of its overtones to be represented reliably in the spectrum. Therefore, a Nyquist frequency of around 6800 Hz would be appropriate, so I settled for an STFT of n = 16000, with a Blackman window of the same number of samples, so it divides the 48kHz sampling rate evenly. I also picked to perform a log10(1+<FFT>) operation to clamp the values to a smaller interval and to suppress negative values.

After some experiments with overlapping windows, it became apparent that it is not feasible for real-time operation. I attempted to hold a buffer 16000 samples long and read 4000 samples into it at a time in an effort to increase the "refresh rate" of the analyzer loop, but this caused large amounts of spectral leaking when visualizing the computed spectrums.

This didn't happen whenever I performed the STFT with overlap of pre-recorded samples of my own, so I could safely rule out signal noise as a possible cause.

As it turns out, the Python sounddevice stream doesn't work like a queue; meaning that the newly read samples didn't come immediately after the previous ones, which will inevitably cause phase discontinuity. Therefore, I settled for non-overlapping, 16000 samples-long windows for training and full buffer reading for real-time operation.



(1) – Example of severe spectral leaking when reading ¼ of the analysis buffer at a time; effect was also present on the note sustain frames

(2) – Example of acceptable spectral leaking when reading an entire analysis buffer at a time; leaking occurs solely on the note attack frame

(3) – Example of no spectral leaking on note sustain frames

For extracting the features used as training data, I wrote a feature extractor configurable via a JSON file, that outputs the spectrum-label pairs into h5py database files.

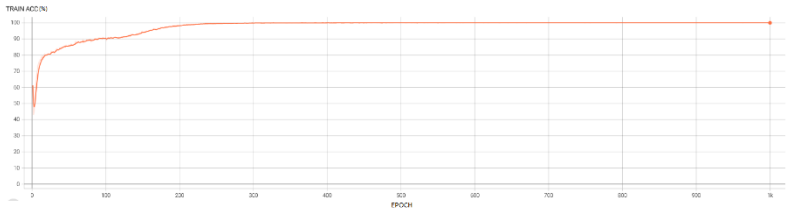
Model Selection

Since we're dealing with 1D data of a fixed length of 8000, I kept things simple in regards to the model choice, namely a Multi-Layer Perceptron. It features an input layer that outputs 4000 features, two hidden layers that output 2304 and 576 features, respectively, and an output layer that outputs the probabilities for the 72 possible labels using a Softmax function. ReLU activations are employed after the input and hidden layers.

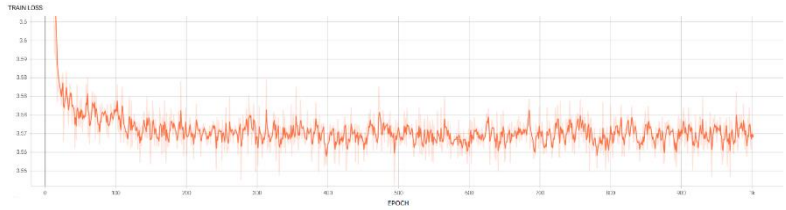
I designed the training script to have its file paths and hyperparameters configurable via a JSON file. I found a learning rate of 0.00002, a batch size of 24 and the default Adam optimizer parameters to be optimal. Categorical Cross Entropy Loss is being used, and the argmax of the outputted vector of probabilities represents the detected note.

At first, I trained the model using only the data from the IDMT-SMT-GUITAR dataset and I had to use it in its entirety, since each note has 1 to 3 samples at most 2 seconds long, which left only the possibility of testing it on a real-time stream of sound coming from my own instrument.

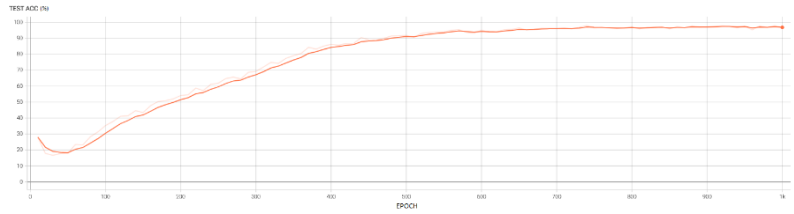
Then, after recording my own dataset, I used it in its entirety for training, totaling over 11000 spectrum-label pairs after processing each recording and used the "Ibanez Power Strat" single-note iterations from IDMT-SMT-GUITAR as the test data, totaling around 1350 spectrum-label pairs after feature extraction.



Plot of accuracy on training data. Converges to 100% at around 300 epochs.



Plot of training loss. While it is seen to fluctuate when zoomed into its narrow range, its median value does decrease slightly from epochs 200 to 1000.



Plot of accuracy on test data during training. It is seen steadily increasing from 30% to 98% from epoch 50 up until epoch 800, where it converges.

Results

The model trained on IDMT-SMT-GUITAR dataset was only tested on the real-time pipeline, since my own dataset wasn't recorded at the time. After settling the spectral leaking issue, the accuracy was decent.

After training the model on my own dataset, I ran tests once again on the single-note iterations from IDMT-SMT-GUITAR, this time on each iteration separately, including the "Fender Strat". The computed accuracy wasn't the same from run-to-run using the same extracted data, so the accuracies shown are computed by averaging results from 5 runs.

	"Ibanez Power Strat Bridge HU"	"Ibanez Power Strat Neck HU"	"Ibanez Power Strat Bridge+Neck SC"	"Fender Strat Neck SC"
Accuracy	96.30 %	97.21 %	96.36 %	97.35%

Then, I ran the model on the real-time pipeline receiving signals from my own instrument, which was used for recording the training dataset.

The model's note tracking was near-perfect, and the detection latency was great. This is expected given that the signal comes from the same instrument, however, the model also performed great on two other pickup positions that were not part of the training data, namely the middle humbucking position and the neck single coil position.

Conclusion

While such an approach is overkill for this relatively simple problem, there are a few takeaways from the obtained results.

Unlike signals obtained by recording acoustic guitars with a microphone, electric guitar DI signals have a lot less variance, due to the microphone type and the recording chamber not being part of the equation. An argument can certainly be made in our case for the guitar pickups differing in sound, but that difference isn't nearly enough to throw off such a classifier, as the results show.

The one problem with the model I obtained is its size, namely 162 MB. This may be reduced by picking a smaller window for computing the STFT, but such a measure will most certainly degrade the performance for lower notes. Employing the Constant Q Transform as a feature may prove quite useful, since its entire point is to allocate more magnitude bins to lower frequencies and less bins to higher frequencies, though its extra computational cost needs to be taken into equation for real-time performance. Another possible improvement is to use 1D convolutions to reduce the number of connections, therefore decreasing the model size.

Finally, there's the matter of extending the model to polyphonic tracking, namely up to 6, 7 or 8 notes at once. Issues and limitations may arise though, for instance, multiple notes played at the same time may interfere with each other due to various resonance phenomena occurring on a physical level. Lower notes and intervals of simple ratios (e.g. octave – 2:1; perfect fifth – 3:2) would be especially susceptible to these.