Andrew Harless
Pace Compatibility Project

The purpose of the project was to use machine learning to find the paces (tempos) consistent with a given song (input as an MP3 file). What made this different from ordinary tempo estimation is that it allowed for multiple compatible paces: for example, a song nominally clocked at 200 beats per minute, but with those beats consistently in groups of 2, would be compatible with 100 bpm pace as well. (The project was motivated by my own experience using music to keep the pace for jogging/walking/running/marching, and the realization that using mutually exclusive lists of songs classified by tempo was an imperfect solution.)

The input data consisted of my own MP3 collection, along with multiple hand labels that I entered for each song, based on tapping out the beat at different tempos and deciding subjectively which ones felt right. (Typical songs had 2 or 3 labels, but a few had only one.) The input data presented several problems:

- There wasn't much of it. If each song was to be treated as a separate training case, I didn't have enough to fit a meaningful machine learning model.

- It wasn't obvious what the appropriate loss function would be: you want to maximize the number of correct labels but penalize duplication (i.e., labels too close together) and incorrect labels. The idea of defining and assigning weights to these 3 objectives seemed too subjective and arbitrary.

- Some songs had inconsistent tempos, especially at the beginning and the end.

- How should I represent the data from a song to generate features for machine learning?

I solved the first two issues by redefining the problem: instead of training a model to generate a set of tempos for each song, I would train it to classify a candidate tempo, associated with a clip from a song, as either correct or incorrect. I used a music analysis library (LibROSA) to generate a set of candidate tempos for each song, then I selected random clips from each song to use as training data. Since the clips began at random points in the song, each clip from a given song had a different phase relative to where the beats would be at the candidate tempo, so the model would perceive each clip as being different, even if the song itself were repetitive. Thus I was able to generate a large set of training data from a small set of songs.

To address the third issue, I just cut off the beginning and end of each song and threw out any songs that had significant tempo shifts in the middle.

I chose to represent the data using mel spectrograms generated by LibROSA, which generates a 128-dimensional time series of spectra. I realized, however, that the periodic nature of beats makes the relevant time coordinates themselves 2-dimensional. Much as one might, in searching for diurnal patterns, stack each day's time series on top of the previous day's, I made a stack of beat quanta. Thus one time dimension represented the immediate passage of time associated with a given (hypothetical) beat, and the other represented the passage of time between successive (hypothetical) beats.

I also resampled each clip to make them all the same size. (The clips were selected to represent a constant number of beats at the candidate tempo, so the raw lengths were different depending on that candidate tempo.) The input data for a each case now consisted of a 3-dimensional tensor, with one dimension representing the spectrum, one representing time within a beat quantum, and one representing time between beats.

I modeled the data using a convolutional neural network, with the spectral frequencies treated as separate channels of 2-dimensional matrix. The initial convolutions were 1x1, to reduce the dimensionality of the spectrum. The subsequent convolutions were narrow. That is, though the input data happened (arbitrarily) to be square (16x16), I found that 1x4 and 4x6 convolutions worked better than square ones. Per common practice, I used a dense layer on top. I found that using a lot of dropout (0.8) at the top layer improved performance. The relatively simple structure of the model allowed me to implement it using the Keras Sequential API.

I made an interesting discovery about CNN modeling. (Perhaps this is known, but I didn't know it.) It seems (at least for this problem) that alternating between (spatial) dropout and batch normalization between successive convolutional layers works better than using either one consistently.

This project is still a work in progress, but the latest version gets about 91% validation accuracy. (Validation uses a separate set of songs from training, so as to minimize leakage.) I have yet to try the model on a clean test set, and I have only taken a quick, casual look at per-song performance (as opposed to per-validation-case performance). I also haven't looked at better measures of performance yet, but a casual look at the results suggests that the model is making good discriminations rather than just favoring the modal class. Given that many of the cases are ambiguous, 91% accuracy, if it pans out, will be quite good performance. The repo is public (https://github.com/andyharless/paces), in case anyone wants to look more closely.