Multiple predominant instrument classification

Lumen 2023. — Data Science

Contents

1. Problem description — What?

A problem well stated is a problem half solved.

– John Dewey

What are we looking to accomplish in this work? In technical terms, we wish to construct a model that can recognize multiple instruments being played in a single polyphonic audio file. Why should one care about our ability to do so? First of all, information on appearing instruments is desired in general: there is an increasing demand for music search owing to the ever growing number of music files available in digital format. Of course, compared to text search, it is difficult to search for music because input queries are usually textual. In addition, the obtained instrument information can be used indirectly. For instance, more instrument-specific and tailored audio equalization can be applied to music of particular author. Similarly, a music recommendation system can be adjusted so that it takes into consideration the user preference of particular instruments. Furthermore, it can also be used to enhance the performance of similar audio tasks. For example, knowing the number and type of the instrument would significantly improve the performance of source separation and automatic music transcription, as well as identifying the genre of the music [1], again with broad uses as mentioned above.

Having (hopefully) motivated the exploration of this challenge properly, why is it even such a challenge? For example, monophonic audio classification is a problem that has generally been solved with high accuracy [2], so what changes when we consider polyphonic data? In polyphonic music, there is interference of simultaneously occurring sounds, and this makes the problem more involved due to several reasons: there is significant variance in timbre and performance style within the same class of instruments, along with perceptual similarity of some instruments that makes them hard to distinguish even for humans, and temporal and frequency superpositions convert simple signals into less trivial mixed ones [3] — this process is difficult to reverse, i.e. to separate the sources is perhaps an even more challenging problem than this one. Finally, there is a lack of large amounts of (freely available) training data for deep learning algorithms in this area, which are notoriously data-hungry [4].

There have been several attempts [3, 5, 6, 7, 8] of solving this problem and considerable progress has been made. In this work, we will consider some of these methods and expand them, as well as consider some novel ones which could be used in the future.

1.1. Dataset exploration

> *You can have data without information, but you cannot have information without data.*
>
> – Daniel Keys Moran

Dataset on which we work is IRMAS [9]. This dataset is separated into 6705 monophonic audio files 3 seconds long, used for training the model and 2874 polyphonic audio files, with length ranging from 5 to 20 seconds, for testing, all in 16 bit audio format sampled at 44.1 kHz and stemming from different western music genres. Regarding the monophonic training data, the number of files per instrument is given in middle column of Table 1.1, with the instruments being (in order) cello, clarinet, flute, acoustic guitar, electric guitar, organ, piano, saxophone, trumpet, violin and voice. In polyphonic, i.e. testing data, the distribution is even more skewed, as can be seen from the rightmost column in the same table. These class imbalances will be accounted for in several ways in what follows, some examples being: training mono classifiers by balancing the dataset to ratios of training and testing data be the same for each instrument, augmenting training data, properly weighting the testing data metrics, etc.

Table 1.1.: Distribution of instruments in training and testing datasets.

Instrument	N° in mono	N° in poly
cel	388	111
cla	505	62
flu	451	163
gac	637	535
gel	760	942
org	682	361
pia	721	995
sax	626	326
tru	577	167
vio	580	211
voi	778	1044

There are also additional labels on our training dataset, concerning the existence of drums and the music genre (albeit the former being incomplete). Regarding the existence of drums, 892 files are noted to contain them, while 990 are noted not to contain them. Out of 6705 training files in total, we note that this is not a significant number of annotations and furthermore, we will later find that drum removal has little to no effect on our models. Thus, we mostly disregard this information. On the other hand, each training audio file is categorized in particular genre (country-folk, classical, pop-rock, latin-soul, jazz-blues) and their distribution can be seen in table 1.2. As opposed to drum labels, we will exploit the genre labels later during data augmentation.

Table 1.2.: Distribution of genres in training dataset.

Genre	N° in mono
cou_fol	482
cla	1675
pop_roc	2476
lat_sou	42
jaz_blu	2030

More specifically, we will use it to mix music in a more educated manner (although more elaborate mixing will prove to be needed). Further regarding the drums, we note that they have not been included in the aforementioned 11 instruments. Thus, we arrive at another two peculiarities of this dataset. First, the drums may or may not be included, but should not be counted as instruments to be recognized: this further complicates the signals of polyphonic audio. Second, our "monophonic" audio is not truly monophonic in another way: there are often other instruments present in it, however only the instrument with which the audio is labeled is present during the whole duration of audio segment (even then, there can be multiple instances of that instruments). All of these complications call for more elaborate models than those applied to simpler datasets, such as OpenMIC [10].

However, in the end we note that there are certain favorable properties of this dataset as well. First of all, all of the files are labeled, albeit weakly. Second, as can be seen from Table 1.2, the dataset covers a variety of genres with all of the files having such label. Third, by manual inspection and consulting the literature [9], it seems that the dataset is labeled well, i.e. there is no need for manual label corrections. Four and final, although the environment and the recording procedure are different for different files, all of the data is of high quality, thus allowing use to use the data without extensive cleaning or remastering.

2. Solutions and Discussions — How and Why?

To steal ideas from one person is plagiarism; to steal from many is research.
– Steven Wright

How do we approach solving this problem and why in that particular way? As usual in problems that require at least some degree of feature/properties analysis, two directions emerge naturally: the classical path of statistical analysis and its modern counterpart — deep neural networks. In this work, we explore both in great detail, however find that deep neural networks are significantly more efficient and effective. Before we dive in, let us emphasize that no excruciating mathematical and physical considerations will be given if they are easily found in standard textbooks — we focus on importance of features and models we explore and provide intuitive explanations, while delegating the rest to accessible references.

2.1. Classical methods

We know that sound is characterized by its frequency and amplitude, so why does then a same tone sound different when played on different instruments? This is due to the fact that each instruments is additionally characterized by its timbre, which is in turn affected by excitations of higher harmonics of the base tone with different amplitudes — based on the different ratios of these amplitudes, instruments gain their unique sounds. For example, we can see in Figure 2.1 how different are the frequency profiles of cello and organ. There are many more natural ways of characterizing the sound of different instruments, and an exhausting overview can be found in [11]. Here, we overview three main categories of features and give examples of each, along with their intuitive significance. After this, all other features[1] can simply be explored in accompanying code, under classical methods.

Features can be broadly separated into those of frequency domain, time domain and cepstral domain. These are connected by (inverse) Fourier transforms, a standard tool in signal processing (which can be understood easily on intuitive level: Fourier transform of a signal will produce representation in which one finds out which particular frequencies contribute with highest amplitudes).

As examples of frequency domain features, we consider spectral centroid and spectral roloff.

Spectral centroid refers to the frequency that is weighted and averaged by the energy of the sound [12]. More intuitively, it describes the brightness of the instrument's timbre: e.g. if the high frequency part of the spectral centroid is more significant, the timbre of the audio tends to be cheerful [13]. For example, Figure 2.2 shows the images of the spectral centroids of the flute and trumpet. The trumpet has a lower timbre compared to the flute, and its spectral centroid is significantly lower than that of the flute.

[1] Chroma STFT, RMS, spectral bandwidth, harmony, etc.

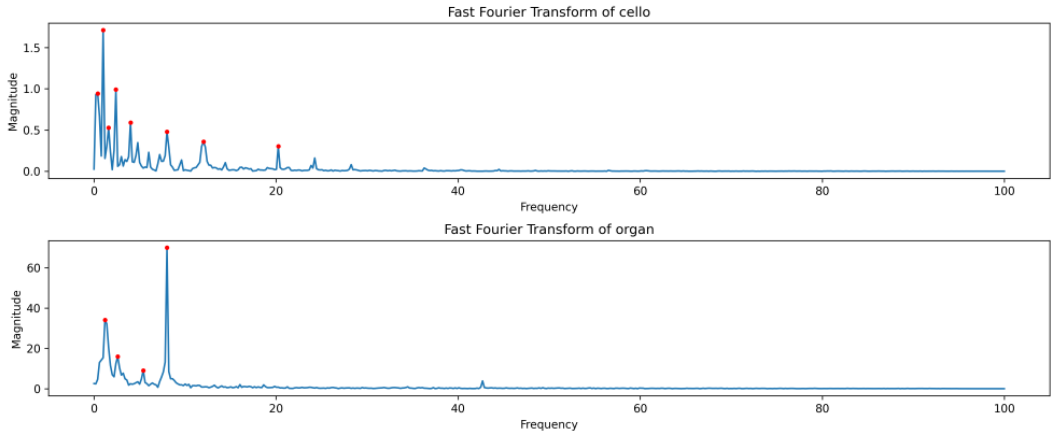

Figure 2.1.: Spectral peaks of cello and organ.

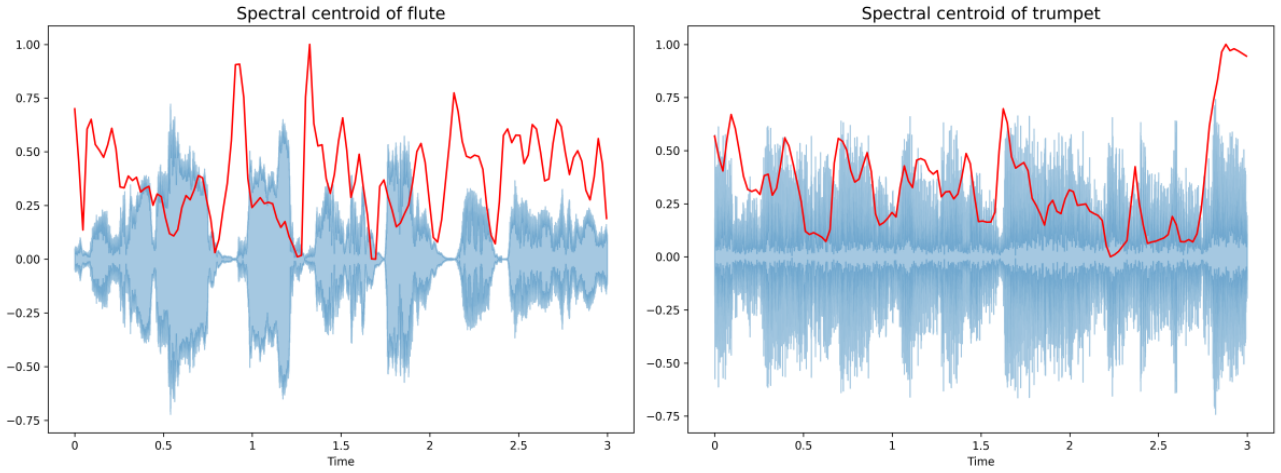

Figure 2.2.: Spectral centroid of flute and trumpet.

Spectral rolloff is related to frequency below which a certain percentage of the total spectral energy is contained. In other words, it represents the frequency below which the majority of the signal's energy lies.

For example, an electric guitar typically has a bright, metallic sound that is characterized by a strong presence of high-frequency harmonics. As a result, as can be seen in Figure 2.3, the spectral rolloff for an electric guitar is likely to be higher than for a saxophone, which has a more mellow, warm tone that is dominated by lower frequencies. In other words, a higher percentage of the total spectral energy for an electric guitar is likely to be contained in the higher frequency range, compared to a saxophone.

Moving on, as an example of time domain features, we consider zero-crossing rate.

The zero-crossing rate is simply a number of times a waveform of our signal crosses the horizontal time axis. This feature can characterize the pitch of the audio as well as the percussive sound [14]. Intuitively, the ZCR can be thought of as a measure of the "busyness" or "activity level" of the signal. For example, a piano typically has a fast decay time and produces sharp sounds. As a result, the ZCR for a piano is likely to be higher than for a

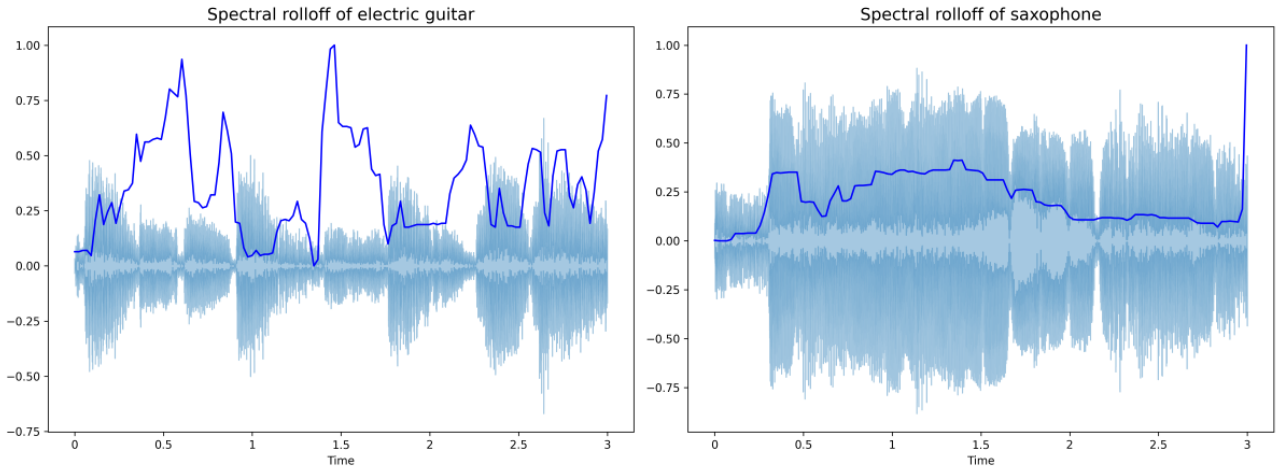

Figure 2.3.: Spectral rollof of electric guitar and saxophone.

violin, which has a more sustained sound with less variation in amplitude. In other words, a piano note will have more frequent zero crossings than a violin note, as can be seen in Figure 2.4.

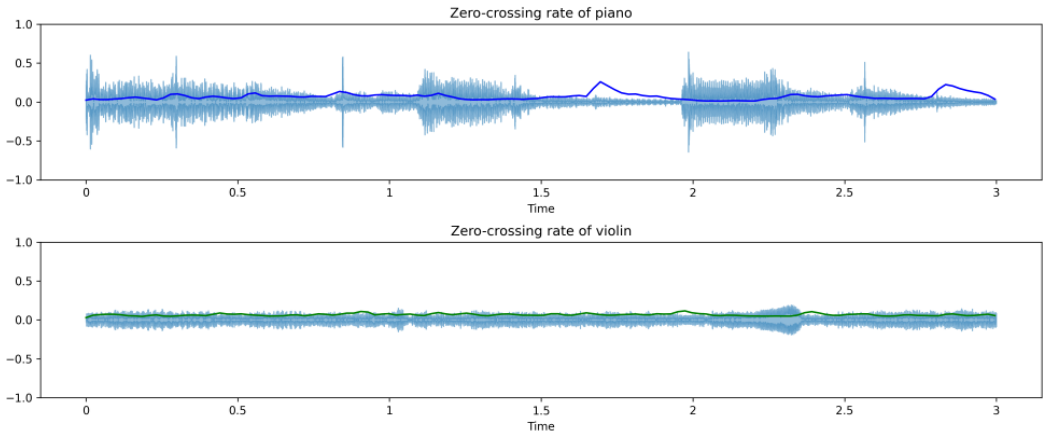

Figure 2.4.: Zero-crossing rate of piano and violin.

Finally, as an example of cepstral domain features, we consider Mel-Frequency Cepstral Coefficients (MFCCs).

MFCCs are calculated from mel spectrograms, which will be discussed later when we venture into neural network methods, however let us point out that they are obtained by applying filters that mimic the human auditory system to the power spectrum of the signal. Intuitively, humans are more sensitive to changes in pitch at lower frequencies, which are important for distinguishing between musical notes and for understanding speech. At higher frequencies, humans are more sensitive to differences in timbre, which are important for distinguishing between different types of musical instruments and for recognizing speech sounds that are produced by different speakers. By using a mel-frequency scale to analyze audio signals, we can capture the spectral characteristics of a signal in a way that is more closely aligned with the way that humans perceive sound. Regarding the MFCCs in particular, the output of each

filter is compressed and transformed using the discrete cosine transform to produce a set of coefficients that represent the spectral envelope (i.e. surrounding shape) of the signal, which is also known to be different for distinct instruments. The first few coefficients typically capture the overall spectral shape, while higher coefficients capture more detailed spectral features (we will later calculate 20 of them).

2.1.1. Monophonic classification

Our first approach is training simpler models based on different features of musical instruments — it seems useful to use this either in conjunction with source separation or simply applying it to smaller segments in which one can perhaps expect there to be only one or two instruments.

Currently in the field of musical instrument classification, the strategy of feature selection for feature engineering is mostly carried out in the same domain i.e. time, frequency or cepstral). This approach has some merits, but single-scale feature learning approach can cause certain limitations [15], e.g. regarding the resonance structure of harmonic and percussive instruments (intuitively, percussive instruments do not have well-defined pitch because they do not produce sustained or continuous sounds with clearly defined fundamental frequencies). Thus, we try and combine features from all three domains, and for those in time and frequency domains, we calculate (using `librosa` package) both their mean and variance. In this way, we end up with 60 features in total and try to train several monophonic classifiers: Naive Bayes, Random forest, Logistic regression, Stochastic gradient descent, Support vector machine and XGBoost. By far, the best results are obtained using SVM and XGBoost. This is expected since, e.g. Naive Bayes classifiers are based on probabilistic models and make strong assumptions about the independence of the features in the data, which is known not to be the case here, as we will soon find out — features are correlated. Similarly, regular logistic regression is not expressive enough to capture more nontrivial boundaries between different instrument classes. The performance of our model is calculated by splitting our training set into training and test, in 80-20 ratio. Furthermore, due to class imbalance of different instruments, we ensure that the splitting is done with this ratio for each instrument, however we find that this does not affect the performance or precision significantly. In this way, we find XGBoost to have precision of 70% and SVM of 74%, which is around 5% better than current best result on classical monophonic classification on IRMAS training data. To arrive at this precision, we have done several adjustments to the classical method. First, we have used Boruta feature selection algorithm. We categorize it as a wrapper method since it uses additional model to fit data. It is based on Random Forest classifier, namely, on tree ensemble model. It's basic idea is similar to a well known statistical tea-cup (lady tea tasting) problem where we examine whether one has sense of taste. It is done by shuffling placebo, in this case, water for cup of tea, and examining whether one recognizes that the cup is filled with tea or not. In the same way we extend our original data set with shuffled data set and compute Random Forest feature importance metric Z value. Then, observing the HIT score we try to determine feature importance. If we succeed, we remove the feature from query set and iterate again, until all features are classified. In this way, we removed features that are statistically insignificant (based on data in Figure 2.5), and also those that are similar

(correlated) to some others, i.e. we reduced the total number of features while retaining the predictive power of the model. After this, the parameters have been optimized using `optuna` package and simple grid search.

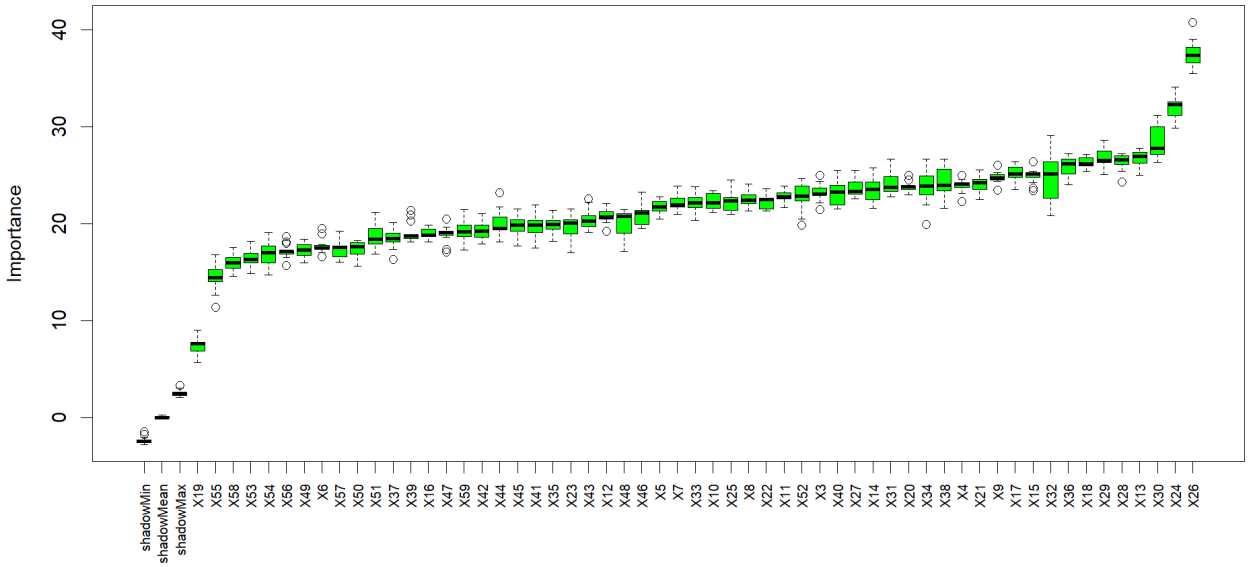

Figure 2.5.: Boruta feature importance.

At this moment, we stop and emphasize that, based on training with much less features (e.g. 7), we find results differing in at most 1% precision. This is due to the fact that this approach does not consist of merely increasing the number of different features: the key here is that these features stem from different domains and thus fusing them allows the information that each carries to complement each other. There are two ways to check this: first, using smaller number of features from same domain, which yields inferior results. Second, adding more MFCCs (e.g. 30 in total) to see that the performance does not change significantly, even if the pure number of features is greater[2]. This intuition of fusing different representations will carry nicely to neural network approach later on.

There have also been several preprocessing variants applied to data, which showed little to no improvement. First, we considered removing the drums using `librosa`'s `hpss` function and found that this actually slightly hurts performance — however, we suspect that since data is of high quality, a paid tool for drum removal could do this much more precisely and without detrimental effects to the rest of the audio, and could thus provide a better sound for classification. In other words, drum removal using this tool is fine for calculating some of the features, but not all. Similarly, we used `nussl` to only select the foreground instrument so that our data is cleaner and truly monophonic. Although this tool yielded satisfying foreground separation when we tested it on random selection of different instruments (e.g. removing piano and drums to only leave saxophone), when applied to the whole dataset it

[2]We have also done manual comparison of features and some of them that are intuitively expected to be similar have indeed been as such, however with this large number of features stemming from three families, the most appropriate way is to do this using fast program packages.

did not improve performance. We further tried to apply it only to instruments for which we have noticed more background sounds, but once again there was no significant improvement in performance.

Continuing, we consult the confusion matrix of our optimal SVM classifier in Figure 2.6. As one can observe and somewhat expectedly, instruments of similar timbre and sound are the hardest for our classifier to distinguish, e.g. cello and flute or cello and violin. Thus, an additional idea is to train several more classifiers to only distinguish pairs of instruments and then in case that any one of them is selected, our data is sent to be classified additionally. Our conjecture is that there is probably a limit on how far this principle can be pushed and that the best approach would be to gather more data specifically for similar instruments. As a side note, let us admit that based on Table 1.1 one may observe that these instruments that we can separate are indeed the predominant ones and that this approach may seemingly be successful, however, we aim for our model to be robust and making strong assumptions about the instrument distribution in the dataset does not bode with that.

Finally, this monophonic classifier has been tried out on polyphonic data based on the following. Since in most of the polyphonic audio, at one particular moment there is usually no more than two or three instruments and often only one, one could hope that after segmenting these audio files, monophonic classification would be applicable. Using different windowing techniques, we find that this is surprisingly not the case. Checking out the calculated values of the features, we note that even small cutoffs and differences can make many features vary significantly and thus for this approach to work, more elaborate feature analysis ought to be made.

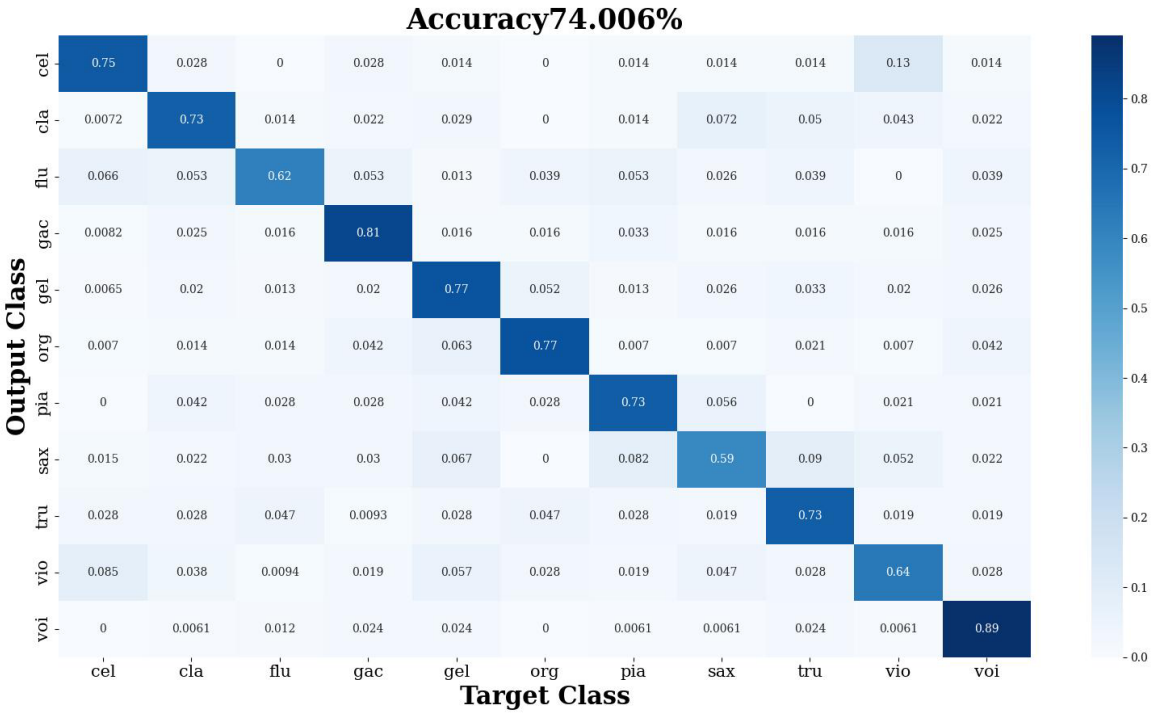

Figure 2.6.: Confusion matrix of monophonic SVM classifier.

2.1.2. Instrument number detection

There are several reasons to think that even without classifying each instrument in polyphonic audio, we can determine their total number. There have been methods of unsupervised learning proposed for such task [8], intuitively based on the idea that some features can be clustered depending on the number of instruments present in polyphonic audio. We try this idea out using k-means clustering and SVM in 2D space in which the two coordinates are spectral centroid and zero-crossing rate, calculated for segments of 20 ms and then clustered. Segmenting was done using several windowing functions, and finally Hamming one was used since it yields better performance than rectangular or Hanning windows. The idea behind this window is that it smoothly cuts off the edges of the window and thus preserves several spectral properties of audio, at least compared to rough rectangular cutoff.

k-means

When calculating the optimal number of clusters, there are several notions of cost functions that are intuitive. For example, inertia is a measure of how well the clusters are separated from each other. It is calculated as the sum of the squared distances between each data point and its assigned cluster centre. The lower the inertia, the better the clustering performance. Similarly, the silhouette score additionally measures how similar the data points within each cluster are to each other. It is calculated as the difference between the mean distance between a data point and all other data points in its own cluster, and the mean distance between a data point and all other data points in the next closest cluster.

There is also the elbow method, a graphical technique used to determine the optimal number of clusters. It involves plotting the inertia or some other measure of cluster quality against the number of clusters used. The point where the plot bends or starts to level off is known as the "elbow", and this is often used as an indicator of the optimal number of clusters. Intuitively, the elbow method suggests that as we increase the number of clusters, the clustering performance initially improves but then starts to level off, and beyond a certain point, adding more clusters does not significantly improve the clustering performance. We do not rely on this method for objective calculations and optimization, but rather as a visual sanity check of our method.

Finally, there is also a technical aspect of deciding whether or not there should be more than on clusters at all, e.g. when our testing file is monophonic. Since silhouette score makes no sense for one cluster, we simply implement this with threshold of inertia ratios for one and two clusters, and if it is below that point, we simply say that there is only one instrument in audio.

Based on all of this and after implementing and optimizing the method, we find that although it yields somewhat acceptable results, i.e. precision of 54%, this is not enough to be used in tandem with our successful monophonic classifier.

To understand this low precision score, we check out some examples for which the classification is wrong, such as the one in Figure 2.7. These are hard for humans to classify as well, so there is little chance that k-means will correctly predict the number of clusters with high confidence. On the other hand, there are several nicely separated clusters, such as the

one in Figure 2.8, and thus we conclude that perhaps additional features as new dimensions could help to better distinguish different clusters since additional dimensions would allow to separate along them. Similarly, based on the desired tradeoff between speed, precision and robustness, it seems natural to experiment with k-medoid, DBSCAN, and OPTICS clustering algorithms. However, based on the fact that much higher precision is needed for our model to be comparable to neural network performance, we choose to pursue this in some other project and not in this one anymore.

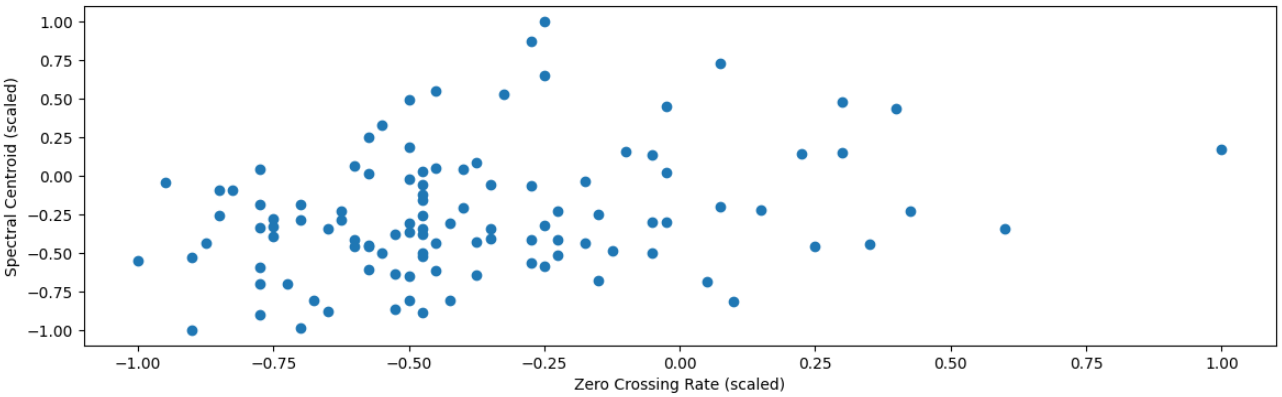

Figure 2.7.: Example of audio file for which there is no obvious optimal way of clustering.

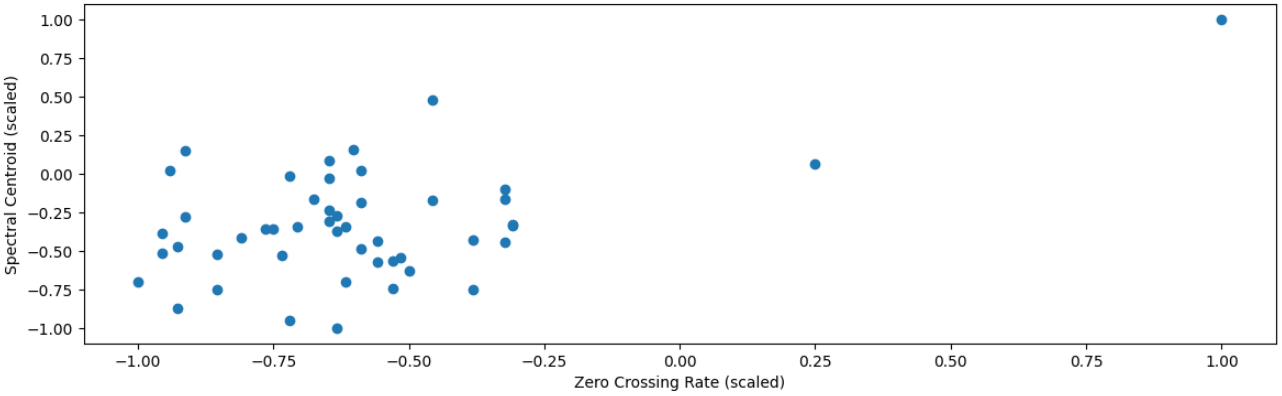

Figure 2.8.: Example of audio file for which there is a comparatively obvious optimal way of clustering.

SVM

Results are similar, but slightly lower than those for k-means instrument number detection and we suppose that this is due to the fact that the dataset is not large so switching from unsupervised to supervised learning is not enough to raise precision. The same possible future considerations as for k-means are applicable in this case as well, but again due to low initial accuracy, we do not pursue this model further at the moment.

2.1.3. Separation

As announced in 2.1.1, one could naturally hope that after having developed and optimized a monophonic instrument classifier, we would be able to separate the initial polyphonic audio and then simply apply our classifier to each separated signal. Several issues arise here. First of all, using intuitively simple classical methods such as NMF, NMF mosaic and PCA, we find very poor performance, as can be calculated by metrics in package `mir_eval`. Along this, we find that there are two additional difficulties when trying to do separation even using deep neural networks.

First, most of the freely available separators are available only for drums, vocals and bass, which is far from sufficient for our purpose. There is hope of applying these separators to at least remove these instruments if they can do it in a precise way and then at least reduce the total number we ought to classify. Once again, we recognize that this could lead to noticeable improvement when complemented with our classifier, this method would both not be quite robust, and also quite cumbersome to implement, none of which is acceptable for our goal of producing an easily useable model.

Second, using commercial instrument separators that include more instruments is evidently not an option for this project, however there could be a possibility of training distinct neural networks to separate each of the instruments, but for this purpose we would need large amounts of data per instrument, which is not available, even when considering all free public music datasets.

These observations represent a natural segue into the methods that work well end to end: these have shown to be different neural network architectures. Let us provide an overview of classical methods before we venture into deep learning world.

2.1.4. Overview of classical methods

Why even include these classical models since they give results that although being acceptable, are far from those of neural networks? There are several reasons for this.

First of all, some observations stemming from representations and methods used in classical algorithms translate to neural networks, e.g. we do not use methods such as drum or background removal that have been found to not be useful in both cases. Similarly, learning about different features and domains helps us consider different representations as neural network inputs: e.g. we will apply fusion of different representations of audio files there as well. Furthermore, based on results of monophonic classification, we find which musical instruments cause the most confusion to our model, thus we can decide which additional data to collect so that we can make our neural network perform better as well. Of course, intuitions such as this one ought to be checked before acting based on them since the ways in which classical methods and neural networks learn are different.

Second, if at some moment we construct an appropriate method for task such as number of instruments calculation, this method can e.g. be used on outputs of neural networks to further increase their precision. One such example is after neural network outputs probabilities of each instrument appearing in polyphonic audio, we can use number of instruments information to optimize our predictions, not only use probability thresholds.

Third, if along the line some of the methods complementary to our classical ones are developed, e.g. source separation, we can use it with the developed ones and obtain a significantly different end to end method for our polyphonic problem.

2.2. Deep learning

> *The longer I work in AI, the more I think humans are just simple pattern matching machines with a small scratch pad for memory.*
>
> – Peter Welinder

After exploration of classical methods, we turn our attention to neural networks. In this part, we give an overview of all the methods in which we tried enhancing our results using different metrics — these methods consist of using different architectures, augmenting our data, tuning the network hyperparameters, fusing the results, representing data in different ways, etc. These ideas and approaches will be outlined in following sections and some assessments of their scores will be given, however we relegate details of the results and their analysis to the following chapter 3.

Before diving into particulars, one should note that the following subsections are intentionally incomplete in the sense that we do not provide every single combination of all parameters since we use several representations, architectures, hyperparameter values, etc. and thus the amount of all possible combinations is huge: we will mention some that worked and some that did not. In this part, we wish to gain an understanding of which approaches worked in general and why they did so.

For our results to be reproducible, all trainings were done (at least) five times and in all cases it was found that their deviation was not significant and thus only the average value was taken.

2.2.1. Data representations

First of all, we ought to choose in which way to represent our data. The most obvious and rudimentary way to do this is in a form of waveform plotted as an amplitude in time, as can be seen in Figures 2.3 and 2.4 One of the main advantages of this representation is that it contains all of the information of the original audio signal, including subtle nuances in timing, amplitude, and phase. This can be important for tasks such as music transcription or speech recognition, where precise timing information is critical, however as we have outlined in introduction to section 2.1, there are features to be learned from other domains (frequency and cepstral) that have more palpable interpretations regarding the unique instrument properties. Furthermore, waveform representation does not capture higher-level acoustic features, such as pitch and timbre, in the same way as some other representations do. This can make it difficult to apply machine learning algorithms to the raw waveform data, especially if we are not able to, due to technical limitations, build larger models for training. Thus, even there indeed are adequate models that perform well in this task and are based on waveform representations [16], we choose to pursue another ones since they provide more intuitive insight into our problem, are less technically demanding and also generally yield better results.

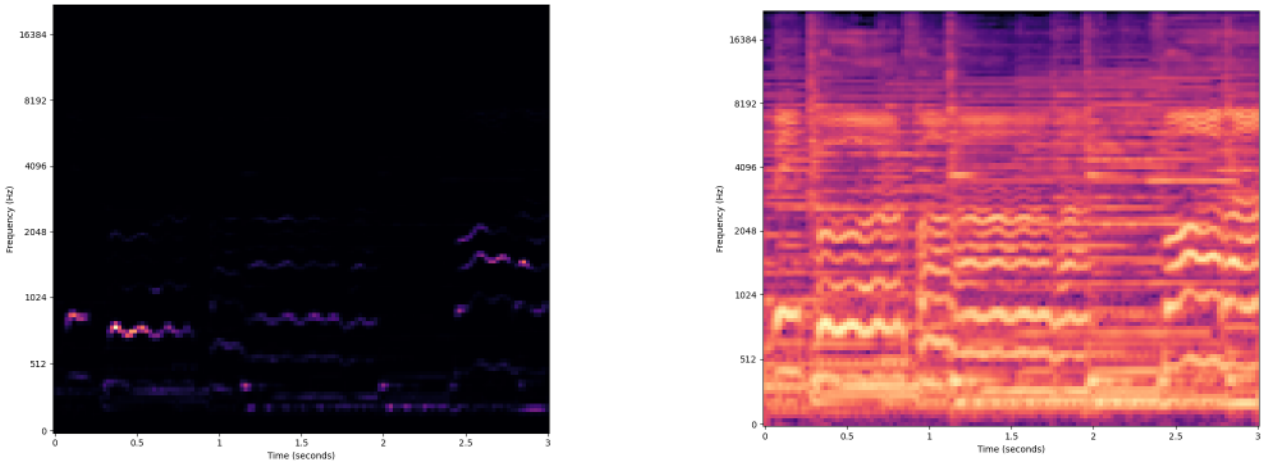

Figure 2.9.: Two examples of frequency representations of human voice sound, with (right) and without (left) adjusting the scale logarithmically.

One such representation are mel spectrograms (MELs), an example of which is given in the right image in Figure 2.9 with time on x axis, frequency on y and colors representing the amplitudes. We have already explained that mel frequency scale is logarithmic so as to give more emphasis to the frequencies that are relevant for human hearing and sound recognition, and thus for instrument recognition as well, since their sounds are generally suitable for human ears. To note how much more information is presented when working at this scale, we can compare this to regular spectrogram given on the left image in same Figure. On it, we can see most of the spectrum being occupied by significantly lower amplitudes (or none at all), converting to mel scale we "zoom" in to the more relevant part so that all its details and intricacies can be analysed at higher resolution.

A similar type of representation are Constant-Q Transforms (CQTs), which differ from mel spectrograms due to their time axis being scaled linearly (compared to logarithmically), but also with frequency resolution not being constant across the entire frequency range: constant ratio between the frequency resolution and calculated center frequency (by averaging). In this way, it is even better suited for modelling how humans perceive sound since it takes into the account the difference between sounds of different average frequencies and centers the frequency range better. One example of CQT is given in Figure 2.10.

Continuing, we turn to a somewhat novel and distinct representation: modulation group delay spectrograms (MODGDGs). To understand them, we first note that the group delay refers to the time delay experienced by different frequency components of a signal as they pass through a system. In the case of audio signals, the group delay can provide information about the phase relationships between different frequency components, which is important for capturing the timbre of a sound, which we have emphasized to be important for distinguishing different musical instruments. This phase relationship information represents additional information which is not contained in MELs or CQTs, however we note that in general this may make MODGDGs more complex, but paradoxically not as useful as MELs and CQTs —

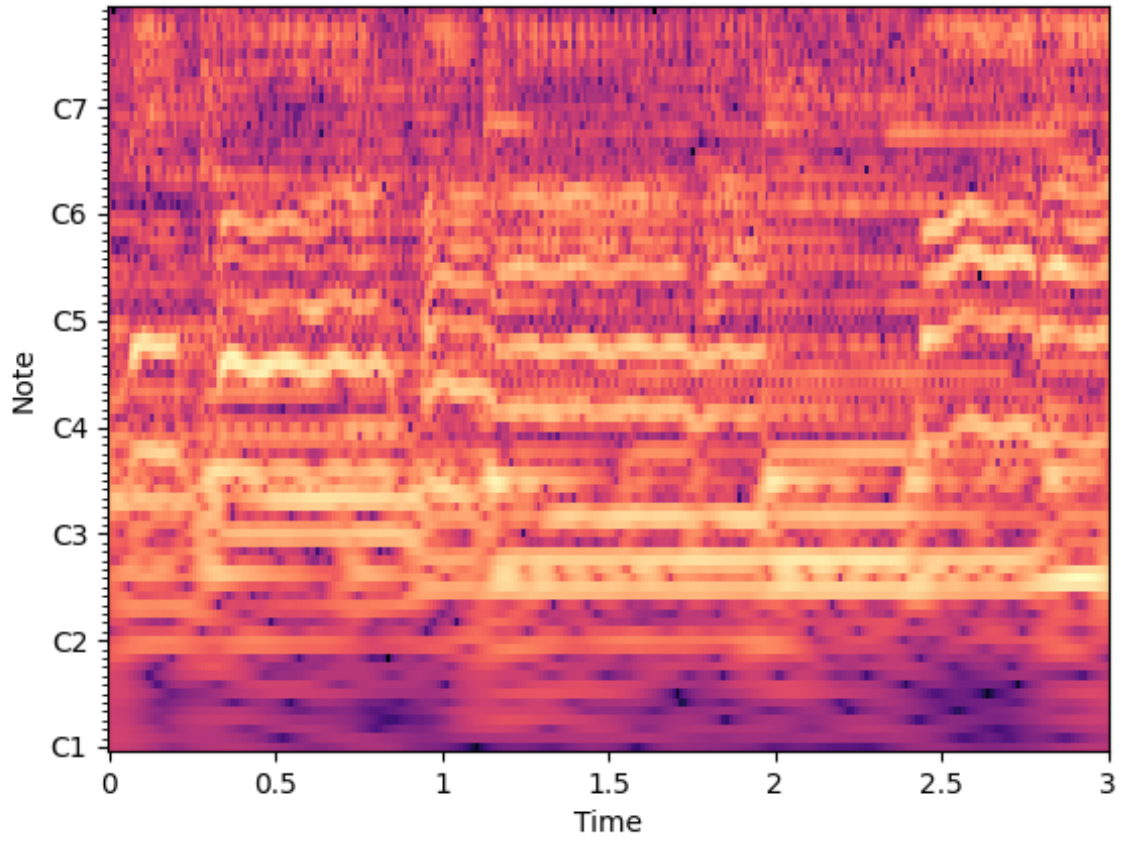

Figure 2.10.: Example of CQT representation of human voice sound.

we observe something to this effect in our trainings, i. e. our computational possibilties are not enough to train networks on modgdgrams on very large networks or they do not encode information as well as other representations, even though they still yield positive results. An example of MODGDG is given in Figure 2.11 for the same example of human voice sound as 2.10.

Finally, we consider tempograms (TEMPs), a type of representation that captures the rhythmic structure of music. They are calculated by analyzing the audio signal to detect the onsets or beats of the music and then constructing a histogram that shows the number of onsets that occur within a given tempo range, as given on y axis in Figure 2.12. The resulting image provides a visualization of the rhythmic content of the music, with peaks in the histogram indicating strong rhythmic patterns or tempos. Compared to MELs and CQTs, which represent the spectral content of the audio signal, and MODGDGs, which represent both spectral content and group delay modulations, TEMPs provide information specifically about the rhythmic structure of the music, but may not capture all aspects of the music's structure, as they only show the occurrence of onsets within a given tempo range. For this reason, they are often used in combination with other audio features to provide a more complete representation of the audio signal for various music analysis tasks, and we find that

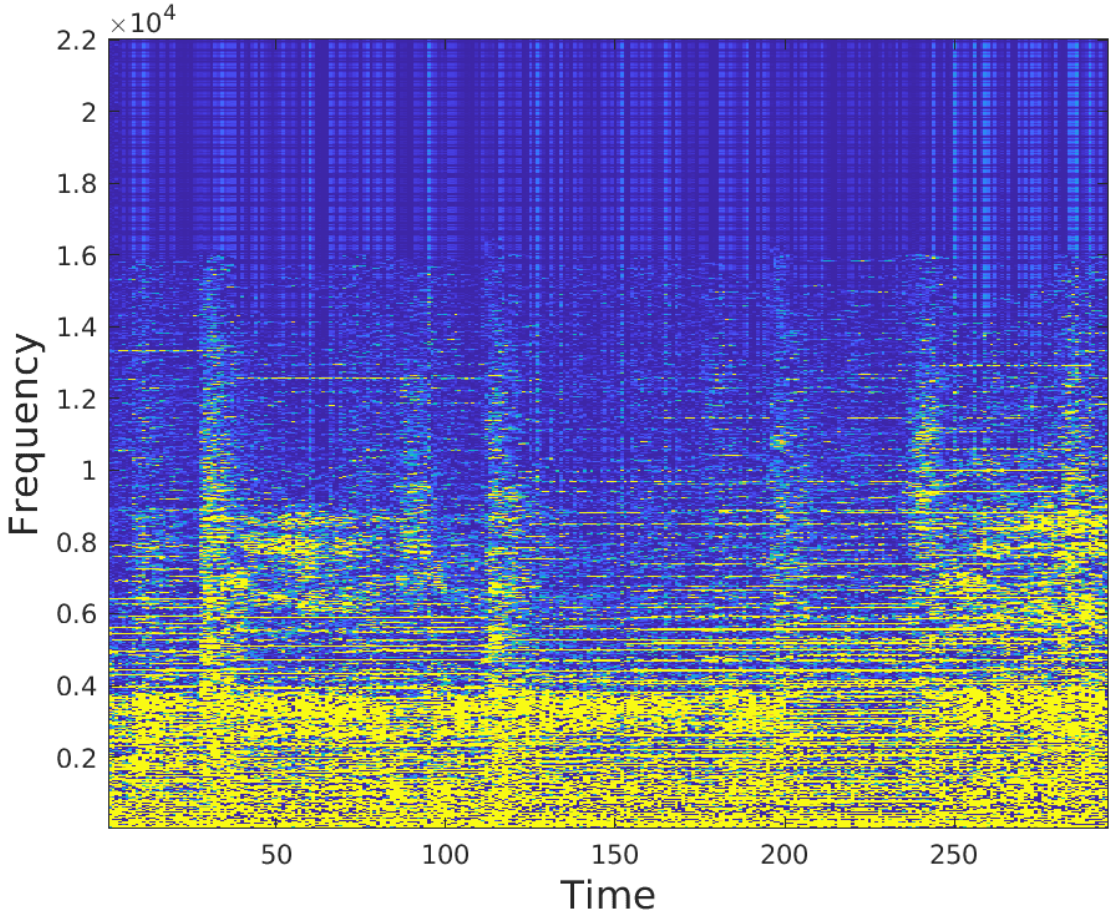

Figure 2.11.: Example of MODGDG representation of human voice sound.

this is necessary in our case as well.

In the end, we announce that MELs and CQTs will show better performance than MODGDGs, which will in turn show better performance than TEMPs, as expected, albeit with less difference between different representations [1].

Particular details on all of these representations and their detailed tradeoffs can be found in [11, 1], however in here we refrain from delving into unneccessary details, but rather point out the most important intuitive differences and provide visual representations so that they are internalized more easily for what follows.

All representations are calculated using frame size of 50 ms and hop size of 10 ms. Furthermore, while MELs, CQTs and TEMPs were calculated using `librosa`'s built in functions, MODGDGs were created by a custom `Python` script that was checked to produce results equivalent to an available one in `MATLAB` [17].

2.2.2. Augmentation methods

After deciding on several representations encompassing different information on audio files, we proceed to discuss augmentation methods that have proven to be useful due to the relatively small size of our dataset.

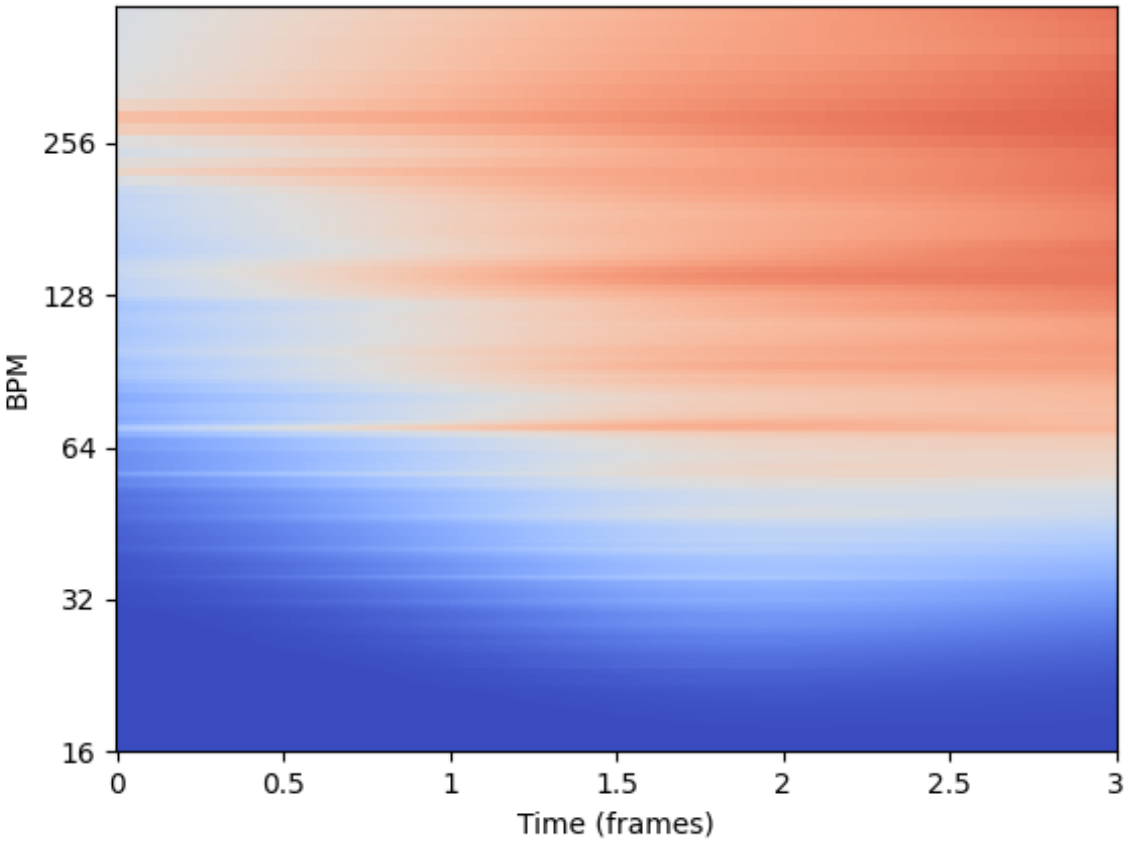

Figure 2.12.: Example of TEMP representation of human voice sound.

As usual, we employ some standard image augmentations such as horizontal flips and Gaussian noise. Similarly, we shift the images in time so that our model is invariant to small variations in the timing of the audio signal since the exact timing of particular musical events (e.g., beats, notes) is less important than the overall audio structure and content.

Since our test set is polyphonic, a natural way of augmenting our data seems to be mixing files of different instruments to create synthetic polyphonic data. This is first done in a simple way by simply overlaying randomly selected files (two or three of them), but then we move onto two more elaborate methods: pitch syncing and tempo syncing. Regarding the pitch syncing, we choose e.g. two segments to be overlayed and then segment them into 300 windows to find frequency shifts between each segment pair using `CREPE` package. After smoothing the pitch shift vector by median filtering in windows of 90 ms length, we finally shift one audio file by the computed semitone shifts using `pyrubberband`. In a similar way, we detect audio tempo using `librosa` and then apply time stretching so as to render both files the same tempo, again using `pyrubberband`.

Finally, as a more complex method of augmentation, we try to use WaveGAN neural network [18]. WaveGAN is a type of generative adversarial network (GAN) that is used to synthesize realistic audio signals. It works by learning to generate new audio samples that are

similar to a set of training examples. The network is composed of two parts: the generator and the discriminator. The generator takes a random noise vector as input and produces a synthetic audio signal. The discriminator takes both real and synthetic audio samples as input and tries to distinguish between them. The generator is trained to produce signals that can fool the discriminator, while the discriminator is trained to correctly identify real and synthetic samples. In other words, WaveGAN works well because it learns to capture the statistical properties of the training data and generate new samples that are similar to the real ones. Regarding its performance, we find slight improvement in general when using WaveGAN augmentation, however we note that a large number (> 4000) of training epochs should be employed so as to generate complete audio files and our local computational setup is not sufficient for this — however, by showing that even with shorter training we observe improvement, it seems promising to use WaveGAN in full and then try and train the neural networks again.

2.2.3. Architectures

After deciding how to represent and augment our inputs, we ought to construct architectures appropriate for the task, i.e. multilabel classification based on images.

We try out an array of different neural network architectures: some of them show excellent results, while others do not (at least on some representations), or are simply too complicated to train with our limited resources, but show promise.

We start with convolutional neural networks (CNNs) that contain at least some sense of successive expansion. The initial layers are able to capture low-level features such as edges, corners, and textures, while the deeper layers can capture more complex and abstract features such as parts of larger units and locations of these units. By combining both low-level and high-level features, our networks are able to learn representations that are highly discriminative and robust to variations in the input, as will be seen in the results section since they are trained on monophonic data and tested on polyphonic data. It is important to view this approach to building CNNs in terms of tradeoffs between said learning ability and its complexity: based on technical demands, we cannot train arbitrarily large neural networks and thus we experiment with number and sizes of convolutional layers until we find the optimal values that we are still able to train.

After this, we add bidirectional GRU layer and remove two convolutional layers, and find that, although it has significantly less parameters, it is only slightly reduced in each evaluation metric. This confirms the suspicion that introducing recurrence into our CNN via GRU layer allows it to learn more complex features of input representations with less parameters and thus we do not need as many convolutional layers as before.

Finally, we consider transformer neural networks, specifically swin and vision transformers. An amazing property of these networks is their ability to model long-range dependencies between image elements: unlike (R)CNNs, which rely on local connections, transformers can capture global patterns and interactions across the entire image. Another advantage of transformers is their ability to handle variable-sized inputs. While our (R)CNNs require fixed-sized inputs, transformers can take in images of different sizes and process them efficiently, making them more robust when training on different resolutions, since no direct image resizing

is needed.

Unfortunately, transformers have been proven to be too complex for us to train properly, although we reach acceptable results with certain representations. To train them fully, several more changes are needed, such as pretraining them on some larger computer vision datasets, as well as training them for a larger number of epochs on our data.

2.2.4. Neural network parameters

All of the networks we train have been proven to be particularly sensitive on many hyperparameters so great care was taken to optimize them, as well as consider alternative solutions — this is one of the many aspects that made the networks challenging to train. We emphasize that three of the models were also optimized using random search and that similar results were obtained when we optimized them parameter by parameter — in other words, the assumption of hyperparameters being orthogonal seems correct and thus we can optimize them one by one.

As mentioned, we tried out different arhitectures and also changed their layers in number, size and type and found these to sometimes make significant difference, but usually not. Some examples of this were: changing the number and size of convolutional layers, introducing recurrent layers, adding and removing fully connected layers, playing with order of pooling and normalization, etc.

Learning rate was always either 0.001 or 0.0001 since other values have proven to be inappropriate, on both ends. Weight decay was also implemented in all models, but has shown not to make much of a difference. For more complex networks, such as transformer ones, more elaborate learning schemes (cosine and similar) were tried out, but to no positive impact on the results.

As a standard loss function appropriate in multiclass labeling, we used categorical crossentropy, mainly due to the fact that most of the literature relies on it, without introducing another formulas. An additional loss function we tried out was binary crossentropy, again to no change in performance.

As an usual optimization algorithm, Adam produced good results, but in some cases its extension, AdamW was evaluated as well — they yielded similar results.

Furthermore, in all our neural networks, batch normalization and dropout were necessarily implemented, both to induce regularization effects and prevent overfitting, but former also for the purpose of training stability, due to effectively forcing the data to have the same scale of values as it passes through the network. For example, stabilizing the network helped when training the transformers, otherwise their loss would quickly start oscillating wildly and then increase.

Due to our dataset without augmentation not being large, we employed an additional method of reducing overfitting, which proved to also have significant impact on our model's performance: early stopping. Particularly, we stopped the training when the validation loss did not reduce for more than a particular number of consecutive epochs: this value ranged from 2 to 8 in our networks.

As another approach to try and remedy transformer performance, we tried pretraining them on large datasets and managed to do so on low resolution ones (MNIST, CIFAR), but

this did not yield improvements — we suspect that longer training on large high-resolution dataset is needed before we attempt to fine tune those models on our dataset.

The rest of the neural network hyperparameters were mostly not tuned.

3. Results

However beautiful the strategy, you should occasionally look at the results.

– Winston Churchill

As a general note in this chapter, we emphasize that since models give very little variation after several trainings, we omit deviations of different metrics and keep only the average values after five trainings.

We start by introducing and discussing the different metrics and elaborate on why we consider a particular subset of them and then proceed to use them to evaluate the different models. The classical models are not included here since they do not produce competitive performance on this task and their precisions for particular subtasks were given in section 2.1.

3.1. Metrics

There is a plethora of metrics that make sense for our task of multiple predominant musical instruments recognition. For example, we can consider precision and recall, which denote the proportion of true positive predictions out of all positive predictions made by the model and out of all actual positive instances in the dataset, respectively. These two metrics are simple and intuitive enough, however we will not use them in this work directly for three reasons. First of all, they can be combined and averaged, as we will soon see, to obtain metrics such as (weighted) F1 micro and macro scores that are usually used in problems similar to the one we are solving and second, a more elaborate notion of precision is found in LRAP score, i.e. the one that measures the average precision of retrieving relevant labels for each instance, but taking into account the ranking of the labels as well. Third, while accuracy is useful for evaluating single-label classification tasks, they can be misleading for multi-label tasks because they do not take into account the fact that a sample can have multiple correct labels.

We recall from data imbalance and discussion in section 1.1 that the relevant metrics ought to be weighted, taking class imbalance into account in order to represent faithful measures of our model's performance.

3.2. Performance of different models

Four main (R)CNN models appearing or similar to those appearing in literature [3, 6, 5, 1, 16] (and also with elements explained in subsection 2.2.3) that we train are shown on the following

page, with

$$
\begin{aligned}
d_i &= [32, 64, 128, 256] \text{ ,} \\
e_i &= 2 \times [64, 128, 256, 640] \text{ ,} \\
f_i &= [8, 16, 24, 32, 64, 128, 256, 512] \text{ ,} \\
g_i &= 2 \times [32, 64, 128, 256] \text{ ,} \\
p_i &= [(2, 2), (2, 2), (3, 3), (3, 3)] \text{ .}
\end{aligned}
$$

.

model1	
x4	2× Conv2D(3×3, d_i)
	MaxPooling2D (3×3)
	Dropout(0.25)
	GlobalMaxPooling2D()
	Dense(1024)
	Dropout(0.5)
	Dense(11, sigmoid)

model2	
x4	2× Conv2D(3×3, e_i)
	BatchNormalization()
	LeakyRelU(α = 0.3)
	MaxPooling2D (pi)
	Dropout(0.2)
	Dense(1024)
	LeakyRelU(α = 0.3)
	BatchNormalization()
	Dropout(0.5)
	Dense(11, sigmoid)

model3	
x4	2× Conv2D(3×3, f_i)
	LeakyRelU(α = 0.33)
	BatchNormalization(β = 0.99)
	MaxPooling2D (3×3)
	Dropout(0.25)
	Dense(1024)
	Dropout(0.5)
	Dense(11, softmax)

model4	
x3	Conv2D(3×3, g_i)
	RelU()
	BatchNormalization (β=0.99)
	MaxPooling2D (3×3)
	Dropout(0.25)
	Dense(512)
	Dropout(0.5)
	Dense(11, softmax)

As announced, several architectures, and augmentation and fusion methods have been used on inputs to our models, with results announced and discussed before: they are precisely stated in Table 3.1. For our chosen solution, we note that it was based on our intuition of fusing different representations from classical methods: trying out several different combinations, we found that fusion of three neural networks with MEL and CQT representations yields the best results, furthermore it yields results similar to state-of-the-art, but with less models being combined compared to five that were fused in literature [6]. In this way, we find a clever way of complementing different representations for high scores on all metrics and thus we use it for our proposed final model: fusion of MEL CNN, along with genre and BMP mixing on another CNN twice, using CQT representation. For this model, we find LRAP of 80%, F1 micro score of 0.628 and F1 macro score of 0.552.

3.3. Overview of deep learning methods

In this chapter, we have provided several architectures that were trained to solve the proposed problem. Even though neural networks are sometimes regarded as "black boxes", let us emphasize that we have went over a lot of justification and intuitive reasoning behind the different choices that were made. Among the technical considerations regarding the difficulties in training the networks and how we overcame them, we used several representations, nontrivial augmentation methods, architecture elements, etc. that resulted in high precision on our testing data. Furthermore, we demonstrated (as we did using classical methods) that fusion of qualitatively distinct representations yields better performance, and expectedly so, since each model learns a certain representation well, resulting in them complementing each other.

Regarding the even deeper interpretations of neural network on structural level, there are several methods of going about this. Before we outline some of them, let us note that these are not immediately relevant to our present work, at least not until some other ideas have been tried out and perfected since we have provided numerous explanations of why certain aspects of the models are as they are. First, we can use a tool such as `TensorBoard` to visualize network activations, and final outputs can help understand what the network is learning and how it makes its predictions. Similarly, we can use attention mechanisms to selectively focus on different parts of the input, and understand which among them are the most relevant for network's predictions. When experimenting with models which have features as inputs, we can use methods similar to ones used in classical case to rank the importance of each feature in the input — this can help understand which features affect network's predictions the most. Finally, we can use model explanation tools such as `LIME` or `SHAP` to generate human-readable explanations of the model's predictions.

Table 3.1.: Some results obtained for different combinations of input representations, architectures, and augmentation methods.

Architecture	Representation	augmentation	LRAP	F1 macro	F1 micro
model2	CQT	/	77.8	0.52	0.61
model2	CQT	pitch	65	0.49	0.51
model2	CQT	random	73	0.53	0.56
model2	CQT	gauss	72	0.49	0.56
model2	CQT	genre	77.49	0.537	0.613
model2	CQT	BPM	78.15	0.534	0.613
model3	MEL	/	75	0.53	0.62
model3	TEMP	/	50	0.41	0.43
model3 (RNN)	MEL	/	71.3	0.501	0.55
model2(BPM, GENRE) + model3 (RNN)	MEL + CQT	BPM, GENRE	79.4	0.56	0.63
model4	MEL	/	72.26	0.497	0.546
SWINT	MEL	/	57.35	0.327	0.41
SWINT	MODGDG	/	58.12	0.34	0.39
model2 (BPM) + model3	MEL + CQT	BPM + /	79.6	0.548	0.62
model2 (BPM, genre) + model3	MEL + CQT	(BPM, genre) + /	80	0.552	0.628

4. Future Prospects — Where?

"Begin at the beginning," the King said, very gravely," and go on till you come to the end: then stop."

– Lewis Carroll, *Alice in Wonderland*

Where do we go from here? We had a lot of additional ideas which we wanted to implement but did not due to technical and temporal limitations. Many of these were mentioned in the body of this work, but among the classical ones, we reiterate combining other solutions for source separation with our classical monophonic classificator or implementing a more sophisticated instrument number detector that can be used in tandem with neural networks. Furthermore, more elaborate augmentation architectures such as WaveGAN and SpecGAN are hoped to be applicable in future with more capable hardware. Similar considerations hold for transformer networks as well. Other datasets could be added to our training data so as to make it more diverse — especially after performing all of the implemented preprocessing/augmentation methods. Finally, we would also like to consider different ways of fusing the models, be it early or in the middle of the model compared to current late type of fusion, as was done in [5].

Bibliography

[1] L. C. Reghunath and R. Rajan, "Transformer-based ensemble method for multiple predominant instruments recognition in polyphonic music," *J AUDIO SPEECH MUSIC PROC.*, vol. 11, 2022.

[2] Y. Liu, Y. Yin, Q. Zhu, and W. Cui, "Musical instrument recognition by xgboost combining feature fusion," *arXiv:2206.00901*, 2022.

[3] Y. Han, J. Kim, and K. Lee, "Deep convolutional neural networks for predominant instrument recognition in polyphonic music," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 25, 2016.

[4] Y. Singh and A. Biswas, "Robustness of musical features on deep learning models for music genre classification," *Expert Systems with Applications*, vol. 199, 2022.

[5] C. R. Lekshmi and R. Rajan, "Multiple predominant instruments recognition in polyphonic music using spectro/modgd-gram fusion," *Circuits Systems and Signal Processing*, 2023.

[6] A. Kratimenos, K. Avramidis, C. Garoufis, A. Zlatintsi, and P. Maragos, "Augmentation methods on monophonic audio for instrument classification in polyphonic music," *2020 28th European Signal Processing Conference (EUSIPCO)*, 2021.

[7] D. Yu, H. Duan, J. Fang, and B. Zeng, "Predominant instrument recognition based on deep neural network with auxiliary classification," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 28, 2020.

[8] N. T. T. S. Pandey, A., "Combination of k-means clustering and support vector machine for instrument detection," *SN Computer Science*, 2022.

[9] J. J. Bosch, J. Janer, F. Fuhrmann, and P. Herrera, "A comparison of sound segregation techniques for predominant instrument recognition in musical audio signals," *ISMIR*, 2012.

[10] E. J. Humphrey, S. Durand, and B. McFee, "Openmic-2018 (v1.0.0) [data set]," *Zenodo*, 2018. [Online]. Available: https://doi.org/10.5281/zenodo.1432913

[11] M. Müller, *Fundamentals of Music Processing: Audio, Analysis, Algorithms, Applications*. Springer-Verlag New York, 2015.

[12] Y. Verma, "A tutorial on spectral feature extraction for audio analytics," 2021. [Online]. Available: https://analyticsindiamag.com/a-tutorial-on-spectral-feature-extraction-for-audio-analytics/

[13] E. Schubert and J. Wolfe, "Does timbral brightness scale with frequency and spectral centroid?" *Acta acustica united with acustica*, vol. 92, 2006.

[14] N. S. Chauhan, "Audio data analysis using deep learning with python (part 1)," 2022. [Online]. Available: https://www.kdnuggets.com/2020/02/audio-data-analysis-deep-learning-python-part-1.html

[15] G. Chen and X. Zhang, "A voice/music classification method based on grey relational analysis," *Acoustic Technology*, 2007.

[16] K. Avramidis, A. Kratimenos, C. Garoufis, A. Zlatintsi, and P. Maragos, "Deep convolutional and recurrent networks for polyphonic instrument classification from monophonic raw audio waveforms," *arXiv:2102.06930*, 2021.

[17] J. Sebastian and H. A. Murthy, "Group delay based music source separation using deep recurrent neural networks," *International Conference on Signal Processing and Communications (SPCOM)*, 2016.

[18] C. Donahue, J. McAuley, and M. Puckette, "Adversarial audio synthesis," in *ICLR*, 2019.