Neuroment - Instrument Detection using Convolutional Neural Networks

Audio Engineering Project

Lorenz Häusler, Lukas Ignaz Maier
Supervisor: Univ.Prof. DI Dr. Alois Sontacchi

Graz, June 15, 2022

Institute of Electronic Music and Acoustics

Abstract

Musical source separation is the task of retrieving the essential elements of a given audio signal. State-of-the-art implementations often provide unsatisfying results containing respectable amounts of audible glitches and artifacts. We propose an algorithm which uses artificial intelligence to determine activation functions of single instruments in a mix. The results of our approach can be helpful in various musical source separation tasks.

The basis of our algorithm is a trained Convolutional Neural Network (CNN). It analyzes multi-resolution spectrograms of temporally framed audio recordings. We compare various frequency transformation methods to generate the spectrograms.

Our network predicts the activation function of each instrument in the input spectrogram over time. The input spectrograms and the output activation functions have the same number of frames and are temporally synced. The features and labels needed to train our network are generated from a dataset of solo instrument audio tracks. These single instrument tracks are mixed in various combinations to ensure flexible training with fixed envelopes.

The trained network reaches a test loss of 0.022. We further evaluate the performance of our network by comparing the predicted activation functions with the reference activations using two reference audio samples. With the noise matrix and the leakage matrix we also define and use two new methods to evaluate the performance. Our results indicate that our approach works reasonably well for single instrument samples, but fails for samples containing a mix of two or more instruments.

All code written in the course of this work[1] is open source and published under the GNU GPL3 license.

[1] See: `https://git.iem.at/s1531597/neuroment2`

Contents

1 Introduction

In recent decades neural networks have seen a considerable increase in popularity for their flexibility and use in a wide range of applications. Some of the main tasks these networks are being designed for are pattern recognition and data interpretation. One sub-task falling under these categories is source separation in audio signal processing. Its goal is to extract a number different elements, often instruments or voices, from an audio signal containing mixes of these elements.

A big obstacle in source separation is to obtain reasonable activations. These indicate how much energy an element, here more specifically an instrument, possesses at a certain time instance within an audio stream. Instruments often sound similar when played in different ranges. For example, a bass being played in an unnaturally high range may sound very similar to a guitar, even for the experienced listener. Naturally, algorithms used for source separation have to prevent these confusions in order to produce a clean separation.

Our goal was to provide an approach which helps to improve the source separation process. For that we suggest an approach which determines single instrument activation functions in a mix of multiple instruments. The output of our approach can be used to improve existing algorithms for transcription or for the determination of self-similarity within a piece. They can also provide the activation matrix for Non-Negative Matrix Factorization (NNMF). Additionally, source separation in general could benefit from the results of our approach.

Recent scientific approaches to musical instrument detection mostly limit themselves to the task of recognition, i.e. whether an instrument is present in a whole input sample or not. Garcia et al. [1] integrate hierarchical neural network structures and few-shot learning into their musical instrument recognition approach. Watcharasupat et al. [2] or Gururani et al. [3] make use of specialized attention mechanisms in order to improve their recognition performance. We want to go one step further by suggesting an approach, which not only outputs a single value describing the probability of an instrument being present in a whole sample, but which outputs a function over time instead.

We feed frequency-domain features computed from an audio signal with multiple instruments into a Convolutional Neural Network (CNN). This network predicts the activation function of single instruments over time. The network has been implemented with high flexibility regarding the amount of training data and methods of feature extraction. The end user setup of the framework is streamlined by an automatic dependency installation and the possibility of using either a CPU or (if available) a GPU for training. The prediction accuracy is measured during training and logged to disk.

In chapter 2 the theoretical basics are explained. First neural networks are introduced, followed by the implemented feature extraction methods.

Chapter 3 depicts the building blocks of the data processing and evaluation pipeline. It lists the most important libraries used in the code base and shows how the code is structured. Furthermore it explains how a user running our framework can configure it in order to work with desired input parameters.

The implementation of the framework is depicted in chapter 4. Data processing, data selection and labeling, file management as well as the algorithms for feature extraction are described here in detail. Also the structure of our CNN is visualized and explained.

The results are shown and discussed in chapter 5. First the training process is evaluated. Then the output of the network is put to the test using plots comparing network input and output and using more comprehensive methods. Finally, various examples of the separation performance are presented.

The underlying work finishes with a conclusion in chapter 6.

2 Theoretical basics

2.1 Artificial neural networks

In recent history artificial neural networks have proven to be useful for classification tasks in various applications. Many of these applications, for example object detection or character recognition, use two-dimensional images as input data for the network.

Audio data is usually represented in the temporal domain. In this domain the audio data only has one dimension, which is time. However, an audio stream can be converted into a (magnitude) spectrogram via a frequency transformation. The resulting spectrogram is two-dimensional, with the dimensions being time and frequency. By using this spectrogram as input feature tensor a network can detect patterns in the spectrogram like it would in an image.

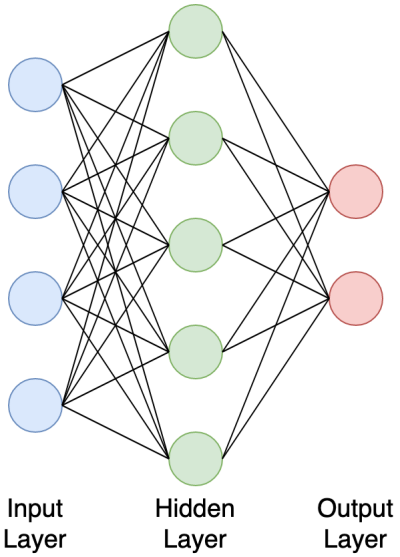


Figure 1: Structure of a sample artificial neural network with three fully connected layers (the input layer, one hidden layer and the output layer).

In figure 1 an example artificial neural network is depicted. It consists of three fully connected layers, which are the input layer, one hidden layer and the output layer. A special form of artificial neural networks are Convolutional Neural Networks (CNNs). Instead of flat layers of neurons, which are fully connected, they use convolutional layers as shown in figure 2.

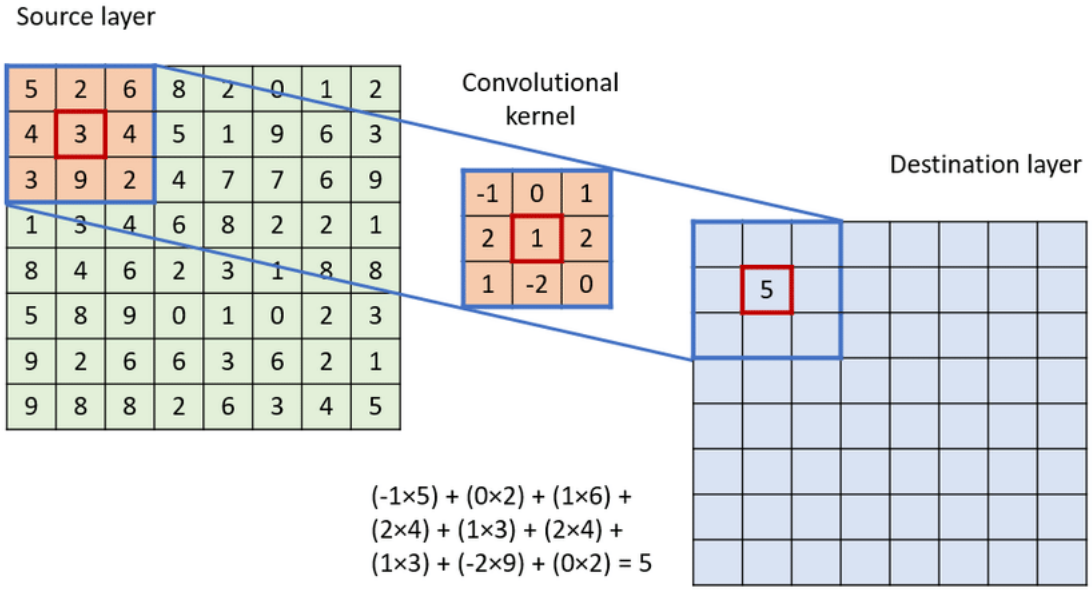


Figure 2: A convolutional kernel [4].

Convolutional layers make use of 2-dimensional convolution kernels. These kernels slide over each possible segment in their input image and convolve each segment with the weights in the kernel. The convolution results then compose the output image. Depending on the stride (i.e. the step width of the kernel) and the padding parameters that are being used the output image generated by the convolutional layer may be either the same size as the input image or smaller.

Using convolutional layers in neural networks allows for better recognition of patterns in the two-dimensional image data than with fully connected layers. They also require considerably less parameters than fully connected layers, thus saving training resources and allowing for faster convergence. Earlier convolutional layers in the network usually detect simple shapes like horizontal and vertical lines or borders, while later convolutional layers allow for detection of more complex patterns.

In figure 3 an example CNN is shown. It consists of an image as input data, three convolutional layers and an arbitrary number of fully connected layers, with the last fully connected layer generating the output predictions.

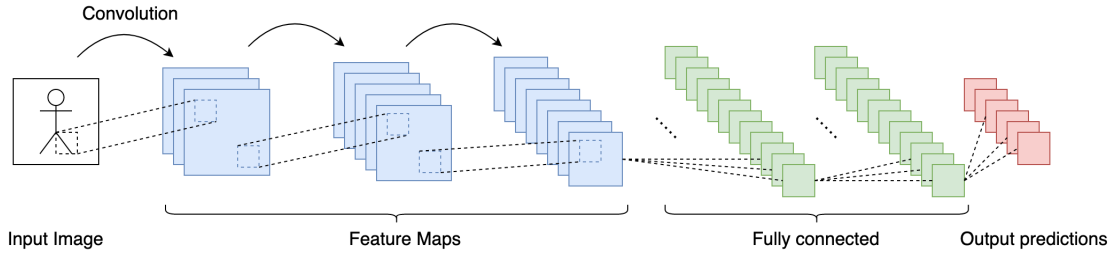


Figure 3: Structure of an example CNN.

2.2 Constant-Q Transform (CQT)

The Constant-Q Transform (CQT) transforms data from time to frequency domain. While this is something other frequency transformations also achieve, its difference lies in the equal spacing of frequency bands.

A quite handy way of explaining the CQT is to start with its parent, the DFT (Discrete Fourier Transform) [5]. The DFT transforms a discrete time signal into a discrete frequency signal. One variant of it, the STFT (Short-Time Fourier Transform) algorithm

$$X[k] = \frac{1}{N} \sum_{n=0}^{N-1} x[n] e^{-j\frac{2\pi kn}{N}} \quad k = 0, ..., N-1 \quad , \tag{1}$$

is one of the most commonly used algorithms in signal processing. N samples $x[n]$ are transformed to N frequency bins $X[k]$. Due to its symmetry properties, the spectrum produced by the STFT is mirrored around $\frac{N}{2} + 1$. To avoid redundancy the mirrored part usually is neglected during computations.

An important property of the STFT is an equal spacing of frequency bins. The spacing between bins is given by $\frac{f_S}{N}$, with f_S being the sampling frequency. To avoid the effect of aliasing, the audio signal may not contain frequencies above the Nyquist frequency of $\frac{f_S}{2}$ [6]. While an equal spacing is computationally easy to handle, it has drawbacks for the analysis of audio signals, specifically ones containing music.

Fundamental frequencies in music are distributed *logarithmically* over the frequency domain. To explain this one may assume the interval of an octave, which is nothing else than two frequencies being in a ratio of $2 : 1$ to each other. In a musical sense, an octave always comprises of 12 semitones (a twelfth of on octave), no matter how high or low the the fundamental frequency. Following, higher semitone intervals have a higher bandwidth $f_\Delta = f_{high} - f_{low}$, where f_{high} is the fundamental frequency of the higher tone and f_{low} the fundamental frequency of the lower tone. Depending on the use case it may therefore be rather unhandy to describe music signals with the equal frequency spacing of the STFT.

The CQT offers help. In its domain there is a minimum frequency f_{min} in the transformation. A usual value for f_{min} is $27.5\,\mathrm{Hz}$. Furthermore, the CQT defines a fixed number of CQT bins k per octave. The number of bins per octave is defined by the value B. The formula for the frequency f_k of a bin with index k is

$$f_k = f_{min} 2^{\frac{k}{B}} \quad . \tag{2}$$

By using the bandwidth f_Δ between two tones we can define the Q-factor. We can determine Q with

$$Q = \frac{f_k}{f_\Delta} \quad . \tag{3}$$

Now, in the traditional STFT domain the bandwidth f_Δ between two tones is fixed, which means that the Q-factor increases with higher frequencies. As already mentioned this is not intuitive for musical intervals, where a semitone interval always sounds like a semitone interval, regardless of the actual Q-factor.

The CQT offers a solution to this problem. In the simplest sense it keeps the Q-factor constant. It does so by essentially using a bank of K band-pass filters that share a common Q-factor, resulting in a logarithmically spaced frequency scale [7]. The CQT formula

$$X[k] = \frac{1}{N[k]} \sum_{n=0}^{N[k]-1} W[k,n]x[n]e^{-j\frac{2\pi kQn}{N[k]}} \tag{4}$$

shows two main differences to the STFT formula. First, the Q-factor is in the exponential term, which makes sense considering the aforementioned logarithmic spacing. Second, the window length $N[k]$ now depends on the frequency bin k. With f_s being the sampling frequency the window length per frequency bin f_k can be computed with

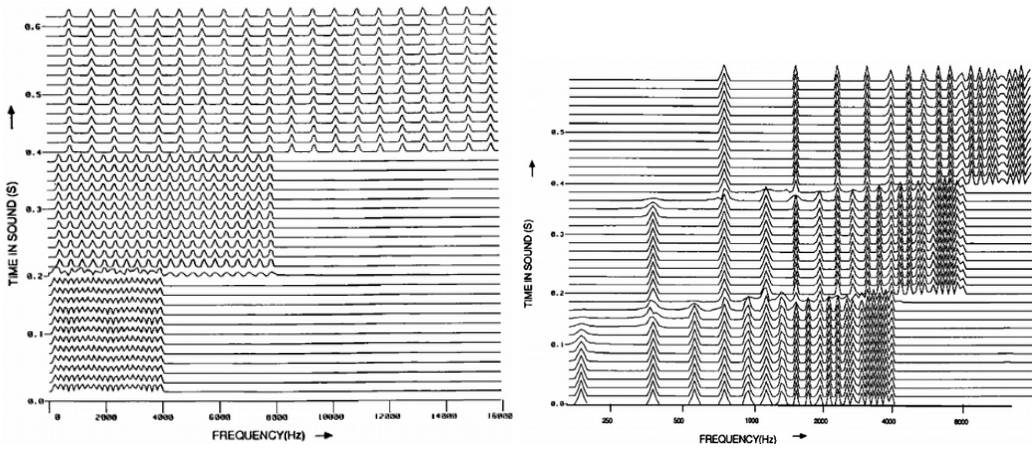
$$N[k] = Q\frac{f_s}{f_k} \quad . \tag{5}$$

The term $W[k,n]$ represents a window function, often a Hann window. In contrast to the window used in the STFT this window function now also depends on two dimensions (time and frequency) instead of only one dimension.

The frequency f_k of each bin depends on the musical interval that is chosen to be between two frequency bins. A usual interval would be a semitone. Considering that there are 12 semitones in each octave we could, as an example, set $B = 12$. This leads to a frequency f_k of each bin k being expressed as

$$f_k = f_{min}2^{\frac{k}{12}} \quad . \tag{6}$$

Figure 4 further points out the differences between the STFT and the CQT domain.



(a) STFT. (b) CQT.

Figure 4: Comparison of STFT and CQT using 3 complex sounds with fundamentals G1 (196 Hz), G4 (392 Hz) and G5 (784 Hz). Each sound has 20 harmonics with equal amplitude. [8]

In figure 4 we see 20 harmonics on a frequency scale. Each of the harmonics is equally spaced. A linear frequency scale in this plot results in equal bin spacing for the STFT and in a decreasing bin spacing for the CQT. Thus a logarithmic scale leads to a decreasing spacing for STFT bins and a continuous spacing for CQT bins. Considering that the human ear works logarithmically this feels natural in a musical sense.

The advantage of the CQT for the underlying task of instrument detection lies in the continuous spacing on a logarithmic scale. The spectral patterns, here musical tones, are invariant from the actual pitch of the signal. This means that it does not matter whether a signal is played in the low, mid or high frequency region of the spectrogram. The resulting pattern will be the same across the frequency dimension and thus is shift-invariant, which makes it better suitable for a CNN.

2.3 Logarithmic Mel-Spectrogram

Another frequency transformation approach similar to the CQT is the so called mel-spectrogram. Here a filter bank is used to achieve a logarithmic spacing in the frequency axis. The basis for the spectral transformation again is a STFT which is then multiplied with the mel filter bank in order to produce the mel-spectrogram. The mel filter bank with which the STFT spectrum is multiplied consists of triangular windows with varying window lengths that depend on the frequency. It is also possible to normalize these triangular windows in order to keep the energy density in the resulting bins constant over the frequency range.

In figure 5 an exemplary filterbank is shown.

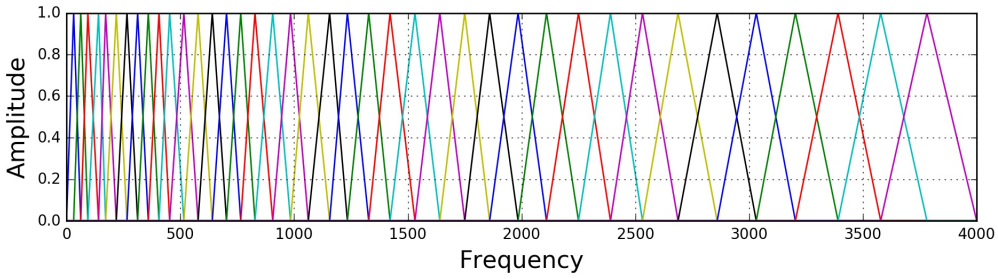


Figure 5: Example of a mel filterbank.

The most important characteristic of a mel-spectrogram is its size. The frequency resolution of a mel-spectrogram covers the observed spectrum with significantly less coefficients than the STFT spectrum without losing much of the information.

3 Code base

3.1 Programming language and libraries

The code base in this work has been implemented in Python 3.9. Python is a programming language which offers a lot of well-developed libraries for data science, music information retrieval and visualization as well as a relatively good readability. It is also one of the most commonly used languages for data science which benefits code accessibility for many users [9].

3.1.1 Libraries and dependencies

Here is a list of the most important Python libraries and dependencies that were used:

NumPy: A library for numeric calculations and linear algebra. Computationally expensive numerical algorithms in this library are implemented via a C backend to allow for much faster computations than with pure Python [10].

LibROSA: A library offering a collection of music information retrieval methods and utilities [11].

PyTorch: A deep-learning API written in Python. It enables easy implementation of neural networks using multiple fast methods of defining and setting up the structure. With it neural networks can be run on a GPU as well, which makes training and prediction with a network much faster than with a CPU [12].

Hydra: A framework which allows for easy and elegant configuration of complex applications [13].

3.2 Configuration

Our application can be configured via a configuration file written in YAML format. This file format is well known for its simplicity and readability and therefore enhances easy access to the project [14].

There are four main scripts in our application (see next section). Their parameters can be set via the aforementioned configuration file or via command line overrides. There are numerous parameters grouped by category which can be modified to the need of the user.

3.3 Structure

The four main scripts performing the main tasks of our framework are as follows:

parse_dataset.py Parses and balances the raw dataset from which features and labels are computed.

data_generation.py Loads audio signals containing single instruments, mixes the raw audio signals in order to generate samples with multiple instruments as well as generates features and labels using these mixes.

train.py Initializes, loads and trains the CNN model. It also writes checkpoints of the model during training and logs the training process to disk.

inference.py Executes all tasks related to predicting single instrument activation functions using the network, which includes reading the raw audio data, converting it to a feature tensor and storing the predicted network output in a given directory.

Usually the scripts are executed in the order in which they are listed here.

4 Implementation

4.1 Data generation

Our untrained network needs a dataset of labeled audio mixtures for training. This means we require either a ready-to-use labeled dataset, or we need to create and label the mixes for the dataset ourselves. Since labeled datasets of mixtures are seemingly unavailable, we chose to pursue the latter.

4.1.1 Data selection

First and foremost, labeled audio files of single instrument recordings are needed. There are a lot of datasets to choose from but we settled on Medley-Solos-DB [15]. It provides mono samples at $44.1\,\mathrm{kHz}$ from a number of solo instruments, while being presented in a file structure that can be parsed easily by our framework.

4.1.2 Dataset balancing

Due to some shortcomings present in the dataset we needed to balance the Medley-Solos-DB dataset. Neural networks learn from training samples. If said samples are predominantly from one class, the network will get biased towards this class in the prediction stage. As figure 6 shows the underlying dataset was greatly unbalanced.

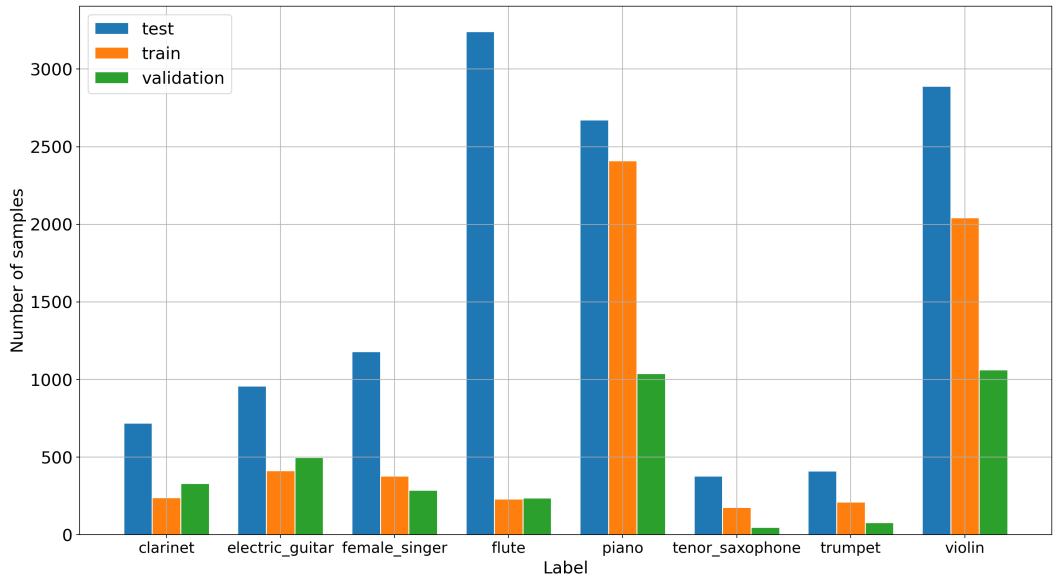


Figure 6: Original, unbalanced class distribution in the Medley-Solos DB dataset.

We see that the distribution of instruments is not nearly evened out. If we train our network using this distribution of instruments it will bias towards the instruments with

more samples. We can minimize the prediction bias if we oversample instrument segments from classes containing little samples while undersampling segments from classes like the piano, which has plenty of recordings. This implies that not every piano sample will be used, but samples from the clarinet class will most likely be part of a mix many times over. Overfitting should not occur since the solo samples in a mix are very unlikely to ever be grouped together again.

We chose a maximum oversampling factor of 2. This led to 420 samples per instrument in the training set, and 3360 samples in the training set in total. The balanced dataset is depicted in figure 7.

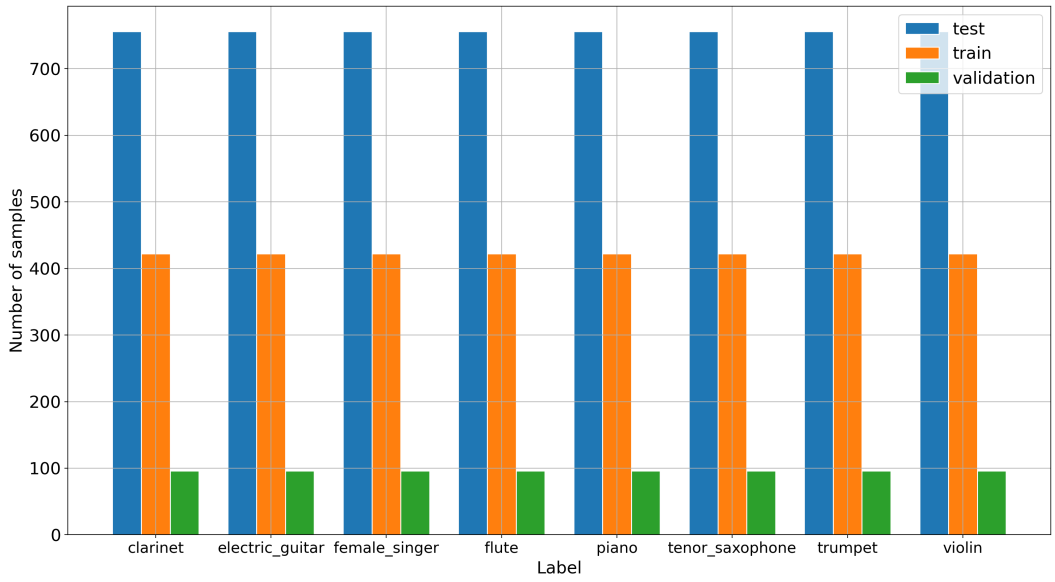


Figure 7: Balanced class distribution after oversampling under-represented classes and undersampling over-represented classes.

4.1.3 Mixing

Real world audio recordings often consist of multiple instruments playing together. The decision to train the model not only with single instrument signals but also with mixed instrument signals therefore was pretty natural. However, directly using samples with multiple instruments for training bears the problem of not having a proper target vector or label (i.e. the "real" activation of each instrument). In order to solve this problem our framework generates mixed instrument samples itself by mixing recordings of solo instruments.

The mixing process is characterized by three main parameters:

- The number of mixes to create (in total)
- The number of instruments to take per mix

- The level of each instrument in a mix

The implementation of the mixing process is straightforward. Creating one mix requires choosing a set of recordings (either from one instrument or from multiple, different instruments), multiplying them with randomly drawn levels and summing them up.

Number of mixes The total number of mixes to create. Together with the batch size that is being used this parameter specifies the number of training steps in each training epoch.

Number of instruments to take per mix To generate more diverse mixes, the number of instruments per mix was modeled via a uniform distribution. This means, that between a minimum and a maximum number of instruments, each number of instruments is equally probable.

$$p[n_{instr}] = \begin{cases} \frac{1}{n_{instr,max} - n_{instr,min} + 1} & n_{instr,min} \leq n_{instr} \leq n_{instr,max} \\ 0 & else \end{cases} \tag{7}$$

We used $n_{instr,min} = 1$ and $n_{instr,max} = 4$. The lower limit represents a single instrument recording. For the upper limit we refer to Stoeter et al. [16] which state that humans can only correctly distinguish up to three voices in a polyphonic piece of music. In our mixing process it is not unlikely that at least one of four single instrument samples contains an audio recording that is either shortly before an onset or fading out. If that happens the corresponding instrument has little to no contribution to the mix. This is why we decided to set the maximum number of instruments to four (instead of three).

Levels of instruments in a mix To simulate different levels of instruments in a mix the levels l were also modeled by the help of a probability distribution. We used a normal distribution $\mathcal{N}(\mu_l, \sigma_l^2), 0)$ for that. Via

$$l = \max(\mathcal{N}(\mu_l, \sigma_l^2), 0) + \epsilon \tag{8}$$

it was also ensured that the level l is bigger than 0 (i.e. silence). We chose to use $\mu_l = 0.5$ (i.e. $-6\,\mathrm{dB}$), $\sigma_l^2 = 0.01$ and $\epsilon = 1e - 8$.

The total level of the instruments is $l_{total} = \sum_{i=1}^{N_{instr}} l[i]$. A total level $l_{total} < 1$ is perfectly fine. However, a total level $l_{total} > 1$ is problematic as it may introduce clipping. To prevent that we chose to normalize each respective instrument level $l[i]$ by the total level l_{total} in case it becomes bigger than 1, like in

$$l[i] = \begin{cases} \frac{\mathcal{N}(\mu_l, \sigma_l^2)}{l_{total}} & l_{total} > 1 \\ \mathcal{N}(\mu_l, \sigma_l^2) & else \end{cases} . \tag{9}$$

Mixing random segments multiplied with randomly drawn levels does not take into account temporal offsets or musical context of the signals that are to be mixed. We regard improving this aspect as an extension to the underlying work.

4.2 Feature extraction

4.2.1 Feature extraction algorithm

The mixing stage generates the audio data as a temporal waveform. To train the network, this audio data now needs to be converted to features from which the network can draw predictions.

When handling audio data a conversion to the frequency domain comes to mind. We decided to compare multiple different frequency representation to see with which transformation our network works best. Therefore we agreed to compare the following representations: a STFT spectrum, a CQT spectrum and a mel-frequency spectrum. The parameters for the computation of the spectra are listed in 4.2.3.

4.2.2 Output matrix for training (Y-matrix)

Often neural networks are fed an input vector at a single time instance and from that predict a one-dimensional output consisting of classes and their likelihood. In our first approaches we used the frequency bins of a single audio frame as input vector for the network to let the network predict one activation value per instrument in this frame. However, we realized that this leads to problems with audio data.

Instrument activations tend to have an envelope in form of a so called ADSR curve (Attack, Decay, Sustain and Release) and have sharp increases of value especially in the first moments of excitation. This can not be predicted very well via a one dimensional output frame. Our mixing process does not take ADSR curves into account, as it simply mixes randomly drawn single instrument samples without considering the temporal offset between the samples in the mix. However, we empirically determined that the model performance still increases if the input features and output predictions of the model comprise enough frames to capture most of an ADSR curve.

We therefore decided to work with a two-dimensional output matrix for each time instance like shown in figure 8. It consists of the instruments and their activation functions over time. The activations extend over a fixed number of frames, which in return is defined by the chosen length of the observation window. We decided to use an observation window of $348\,\mathrm{ms}$. We targeted to use approximately $350\,\mathrm{ms}$, the exact length of $348\,\mathrm{ms}$ is given by the frame size of 2048 samples, the hop size of 1024 samples and the centering of frames applied during the forward frequency transformation.

Supervised training requires labels. These labels in our case are the reference output matrices containing envelopes which are computed from the single instrument recordings in a given mix. The model is trained with the provided envelope matrices instead of

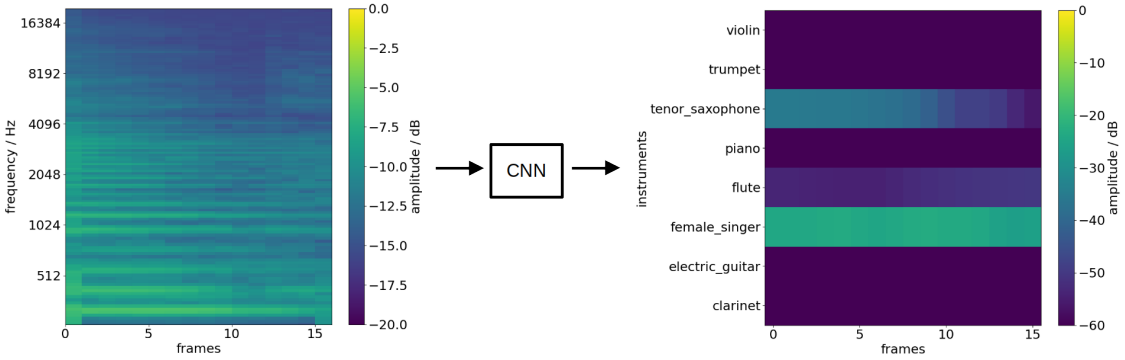


Figure 8: Neural Network structure (left-to-right).

one-hot class vectors and, as already mentioned, also outputs a prediction in matrix form.

The activations are computed for each observation window in an audio sample. During training time only one observation window at a time is of interest. However, during inference we want to have the output matrix of the whole audio file returned by the network. This means that we need to concatenate the output matrices of each observation window in a way that the output frames are temporally synced with the input frames.

We decided to stitch together all predicted output matrices of an audio file via overlap-add, using a Hann window and $50\,\%$ overlap. Applying a Hann window to each observation window also minimizes the contribution of the first frames and the last frames in each observation window to the concatenated matrix. This may be beneficial, as the convolutional layers in the network are set to zero-pad outside of these frames in order to keep the number of output frames equal to the number of input frames. This zero-padding could potentially lead to edge effects, which in return are nullified by the Hann window.

4.2.3 Dataset

We created our dataset by mixing samples of single instrument recordings. The mixing process ensures that each available sample contributes to one mix during one mixing epoch. We decided to use five mixing epochs. This means, that in a training epoch each single instrument file appears five times, each time in a different mix.

The balanced training set has 3360 single instrument samples in total. The average number of instruments in a mix can be computed with

$$n_{instr,avg} = n_{instr,min} + \frac{n_{instr,max} - n_{instr,min}}{2} = 1 + \frac{4-1}{2} = 2.5 \quad . \tag{10}$$

Therefore the total number of mixes in the training set is $\frac{3360}{2.5} \cdot 5 = 6720$. Furthermore, we use an observation window length of $0.348\,\mathrm{s}$. All single instrument files in the dataset are $3\,\mathrm{s}$ long. The observation window fits 8 times into that, which means that each single instrument file may contribute 8 different samples to the mix generation.

This leads to a final number of $6720 \cdot 8 = 53760$ samples in the training set, or $53760 \cdot 0.348s = 18708.48s = 5.20\,\mathrm{h}$ of mix signals in the training set. Using the same computation method we derive that the validation set has 12160 samples, which corresponds to $1.18\,\mathrm{h}$ of audio data. The test set has 96630 samples, which corresponds to $9.34\,\mathrm{h}$ of audio data. The split into training, validation and test data was given by the dataset (see 4.1.2).

For the computation of the instrument activations we decided to use the root-mean square (RMS) value. It describes the energy of an instrument at a given time instance and it can be computed very efficiently in the time domain.

The number of frames in a feature matrix (STFT, mel-spectrogram or CQT) is 16. The number of bins in a feature matrix is defined by the number of bins generated by the frequency transformation algorithm that is being used. Here is a list of the parameters used for each feature computation algorithm:

- Sample rate: $44\,100\,\mathrm{Hz}$
- STFT parameters
  - Frame size: 2048 samples
  - Hop size: 1024 samples
  - Window function: Hann
  - Centering frames such that they are centered around multiples of the hop size
- Mel-spectrogram parameters
  - Length: 128 bins
  - Min. frequency: $27.5\,\mathrm{Hz}$
  - Max. frequency: $20\,\mathrm{kHz}$
- CQT parameters
  - Number of bins per octave: 24
  - Number of octaves: 8
  - Min. frequency: $27.5\,\mathrm{Hz}$

Note that the mel-spectrogram uses the same frame size, hop size, window function and centering parameters as the STFT.

Using the parameters above we receive the following feature matrix dimensions:

- STFT: 1025×16
- Mel-spectrogram: 128×16
- CQT: 192×16

4.3 Model Structure

The structure of our model is visualized in figure 9. The main building blocks of the network are its convolutional layers. The first pair of these uses a kernel size of 3×3, the second pair a kernel size of 5×5 and the third pair a kernel size of 7×7. All convolutional layers use a stride of 1×1 and symmetric zero-padding in order to keep the tensor dimensions constant. We apply the Rectified Linear Unit (ReLU) activation after each convolutional layer.

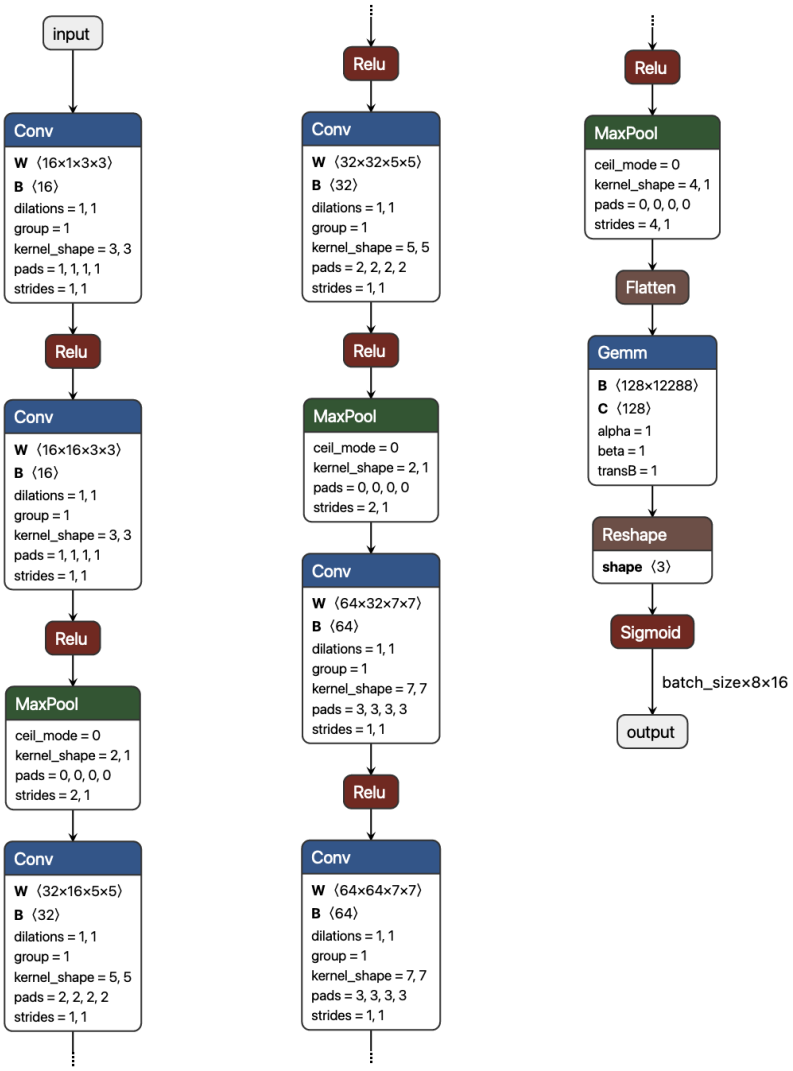


Figure 9: Neural Network structure (left-to-right).

After each pair of convolutional layers there is a maximum pooling (MaxPooling) layer to reduce the tensor size and thus the network complexity. All MaxPooling layers only pool in the frequency dimension, thus keeping the temporal dimension (i.e. the number of frames) constant. The first two MaxPooling layers use pooling kernel of size 2×1, while

the third layer uses a pooling kernel of size 4×1.

After the first and second MaxPooling layer we also used a spatial dropout layer during training, which drops entire channels in the feature map with a probability of 0.1. It is inserted in order to avoid overfitting.

The total number of model weights depends on the feature computation algorithm that is being used:

- STFT: $8.73\,\mathrm{M}$ weights
- Mel-spectrogram: $1.39\,\mathrm{M}$ weights
- CQT: $1.92\,\mathrm{M}$ weights

4.4 Training

During training we used a batch size of 32. One batch consists of one observation window of features (and labels). We used the Adam optimizer with a base learning rate of 0.001 and a weight decay of 0.001 in order to avoid network weights becoming too large. The dataset containing the mixes was shuffled in between epochs.

For the dropout layers we used a dropout rate of 0.1. To avoid overfitting after a certain number of epochs we implemented a learning rate reduction scheduler, which divided the learning rate by 5 each time the validation loss did not decrease for at least 8 epochs. The minimum learning rate was set to 0.00001. Because of the learning rate reduction we decided that an early stopping mechanism would not be necessary, so we trained our network for a fixed number of 150 epochs.

For the loss function we used the binary crossentropy (BC) loss. This is pretty unusual for a regression task, as BC loss is usually applied to classification tasks. However, with the more prominently used mean-square error (MSE) loss, the predictions of our network collapsed during training. After some epochs suddenly all values in the output matrix of the network started being constantly zero for the remaining training process.

Still looking forward to giving an explanation to that, we did research and realized that this often happens if label tensors are sparse. Our label matrix always consists of eight instruments with a maximum of four instruments active in a mix. This means that each label matrix is at least 50% sparse. We therefore assume that our label matrices are sparse enough to make the prediction collapse.

This collapsing issue did not occur with the BC loss, so we agreed to use it. We also tried to add additional additive loss terms, like the Kullbeck-Leibler divergence, or a Frobenius norm based loss. However, the best results were produced when solely the BC loss was used.

5 Results

5.1 Comparison of feature types

In the first step we wanted to compare the feature types: STFT, CQT and mel-spectrogram. We generated three separate datasets using each of the feature types. For each of these datasets we trained the model from scratch in order to compare the feature types.

The STFT features did not work at all. The predictions were mostly zero when using these features, not grasping any instrument. This is probably due to the relatively long STFT feature vector (1025 bins) in comparison to the CQT vector (192 bins) and the mel-spectrogram vector (128 bins). Our network is simply not complex enough for 1025 bin feature vectors. Additionally, the spectrum generated by the STFT is not shift-invariant regarding the pitch in the audio signal and thus not well suited for convolutional layers with fixed kernel sizes.

Both CQT and mel-spectrogram features worked well during training of the network. They showed some slight advantages in certain instruments where the other feature type was not that good. Overall their performance was equally well. As the mel-spectrogram produces smaller feature tensors than the CQT we decided to use that in the end. All the results shown in the subsequent sections are therefore generated using the model trained with mel-spectrogram features.

5.2 Training process

Now we look at the results of the training process. It is best described by the loss curve over the training steps, shown in figure 10.

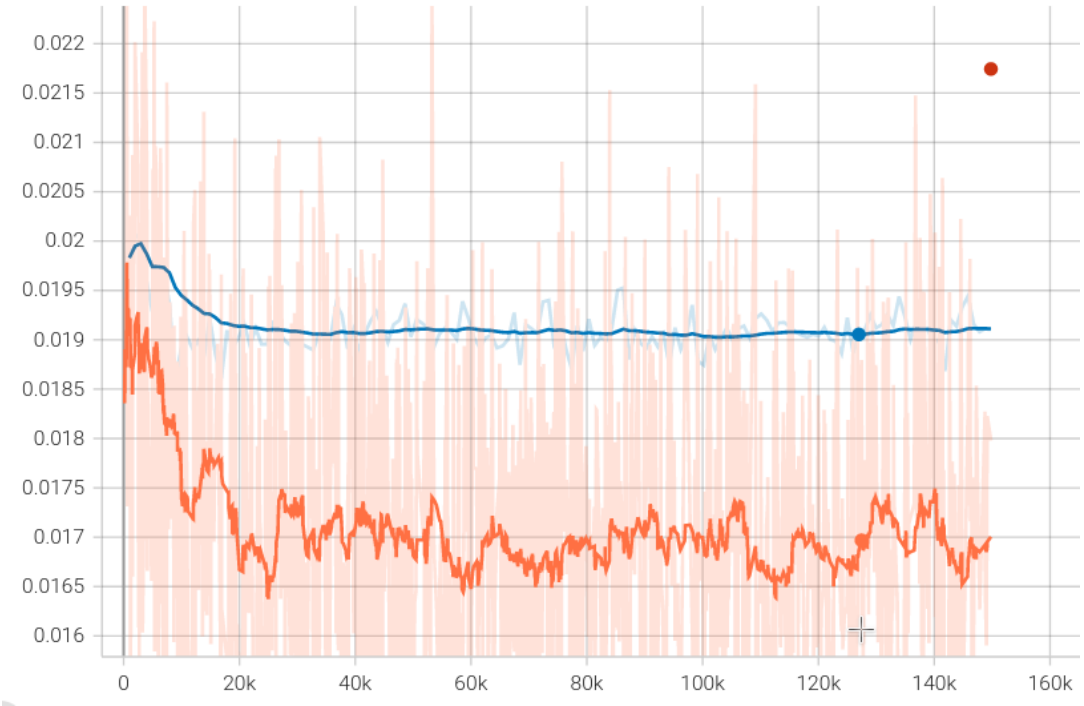


Figure 10: Binary cross entropy loss over training steps, using 150 epochs. The training loss is shown in orange, the validation loss in blue. The test loss is the red dot in the top-right corner. The blurry lines in the background show the real values, while the thick lines show a moving average over the number of epochs.

At the beginning of training the loss decreases faster. With a growing number of epochs however, the slope of the decrease becomes lower. The lower the slope becomes the more likely overfitting may happen, which is why we implemented the learning rate reduction. The averaged curves show that after approximately 40 epochs (40k steps) there is no real increase in performance anymore.

This also makes sense if we look at the learning rate in figure 11. The learning rate is at its minimum possible value of 0.00001 from the 36th epoch on.

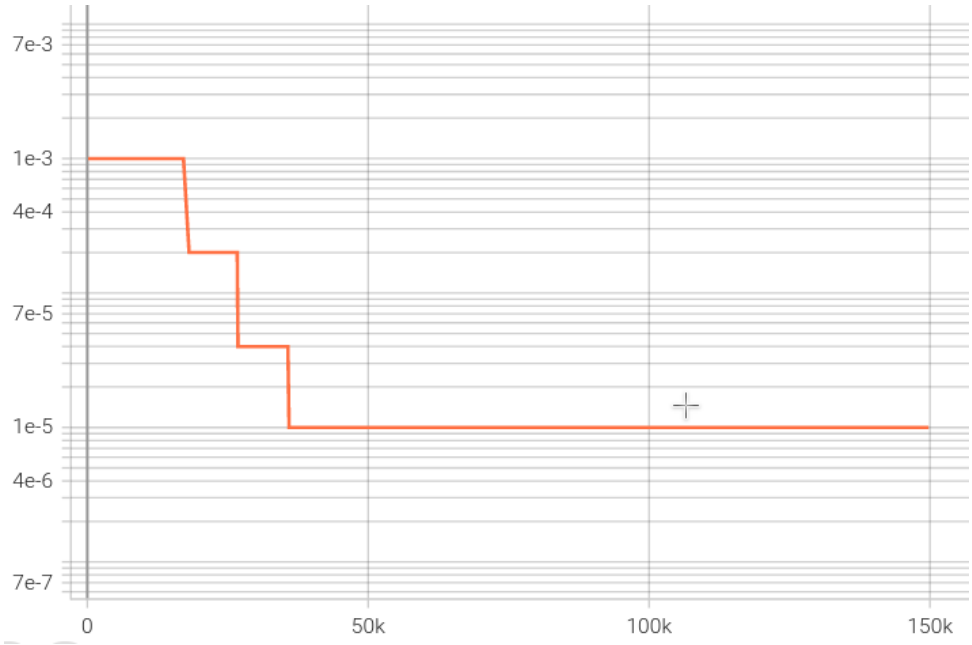


Figure 11: Adaptive learning rate over training steps. We see a learning rate reduction at approximately 18 epochs, 26 epochs and 36 epochs.

We observe that the network converges pretty fast overall. This may be explained by our training dataset containing each original single instrument recording five times (in five different mixes). However, we also see that the validation loss is considerably higher than the training loss, indicating that the network does not generalize well. The same goes for the final test loss.

5.3 Predictions

To evaluate the predictions three audio samples were created. The first two samples are named "Sequence1" and "Sequence2". They contain a sequence of all 8 instruments the network was trained with. Each instrument is played sequentially for 5 seconds. By playing them sequentially it can be determined how well the network differentiates between instruments when they're not playing concurrently and how much "leakage" or confusion exists between said instruments. Sequence1 uses audio samples from the test set of the MedleySolos-DB dataset, while Sequence2 uses publicly available audio samples from freesound.org.

The third sample was a self-generated example from a piece from the video game Zelda - Ocarina Of Time including the piano, clarinet and flute. This example is especially fitting since only two instruments are playing simultaneously at any given instance. This sample is a simulation of a real world example where multiple instruments are playing at a time, while providing more information, due to the accessibility of the solo tracks.

We limited the dynamic range of our network output (and the labels) to 60dB. Figure 12 shows the results for Sequence1, figure 13 shows the results for Sequence2 and figure 14

shows the results for the Zelda sample.

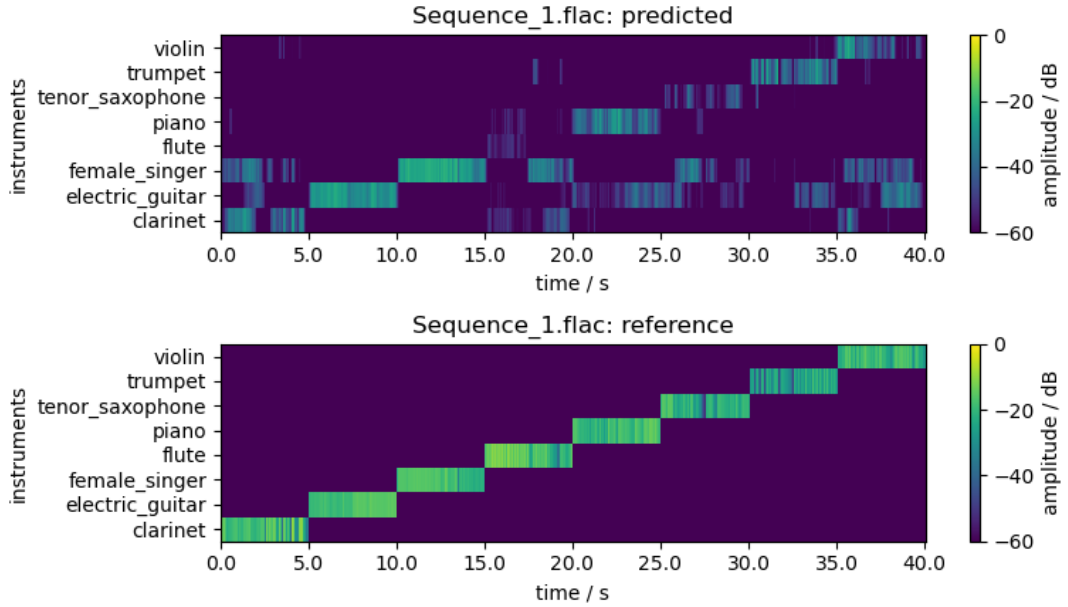


Figure 12: Predictions for Sequence1 (audio samples from test set).

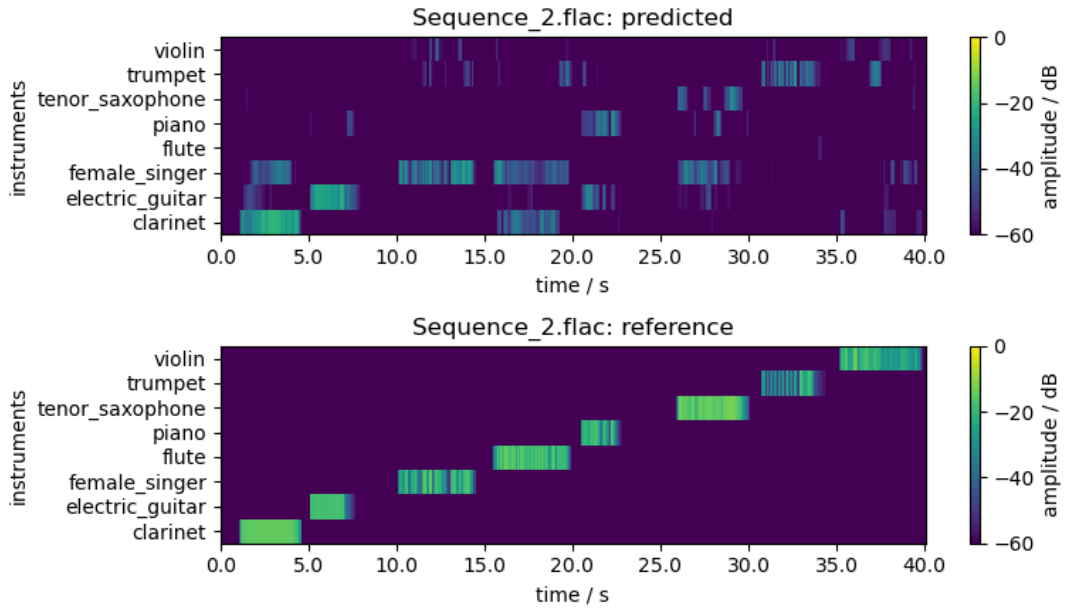


Figure 13: Predictions for Sequence2 (audio samples from freesound.org).

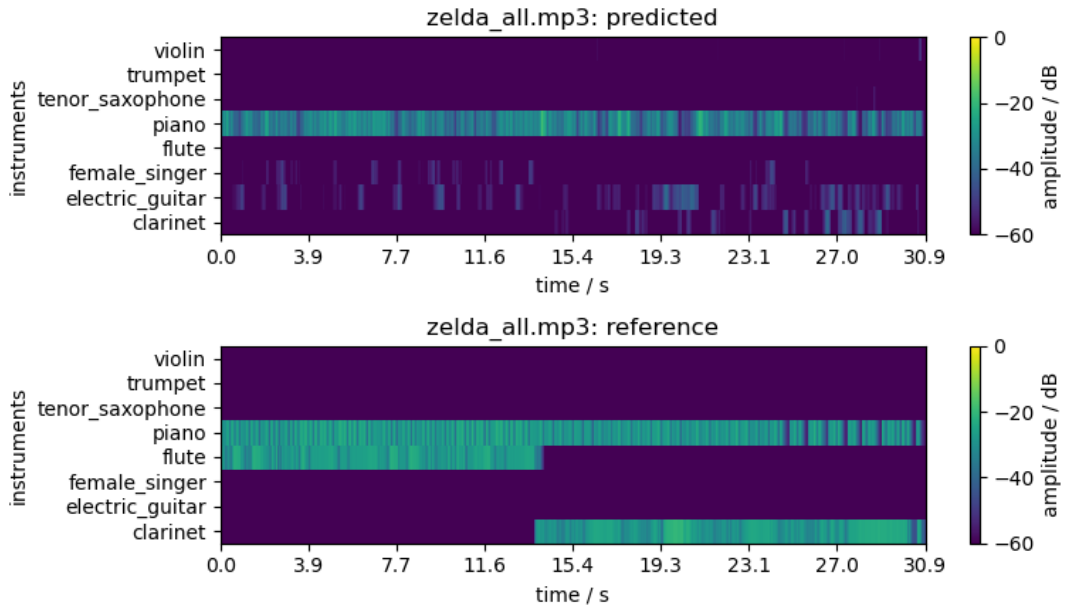


Figure 14: Predictions for the real world sample.

Some of the instruments like the piano, the electric guitar and the female singer are being detected quite well and distinctly. Others like the flute and the tenor saxophone are nearly not being detected at all. In addition to the visualization above we provide a more detailed analysis in the following sections.

5.4 Prediction Error

In order to evaluate the performance of a common neural network the prediction results can be analyzed in many different ways. In most cases the error from the expected results is evaluated and the confusion with other classes is being investigated. Evaluating the confusion of classes is tricky for our given task.

Instead of computing a confusion matrix we decided to evaluate the more fitting measures *noise* and *leakage*. We evaluate these measures in a similar way a confusion matrix is normally determined.

5.4.1 Noise Matrix

Here the resilience to falsely detected activation functions is being investigated. In other words, how much is an instrument being detected when it should not be. This is evaluated by averaging the absolute deviations of the predicted activation functions from the reference activation functions of each instrument over the 5 second window, even if they are not active (the envelope is zero in this case).

This matrix does not describe the confusion, like the visualization suggests, but only gives insight about how much the network learned to follow the intended envelope (even if it is zero). Furthermore, the values were cut below $-60\,\mathrm{dB}$ for visualization purposes since this would be considered not perceptually relevant if another sound source is present.

The noise matrix for Sequence1 is shown in figure 15, the matrix for Sequence2 in figure 16.

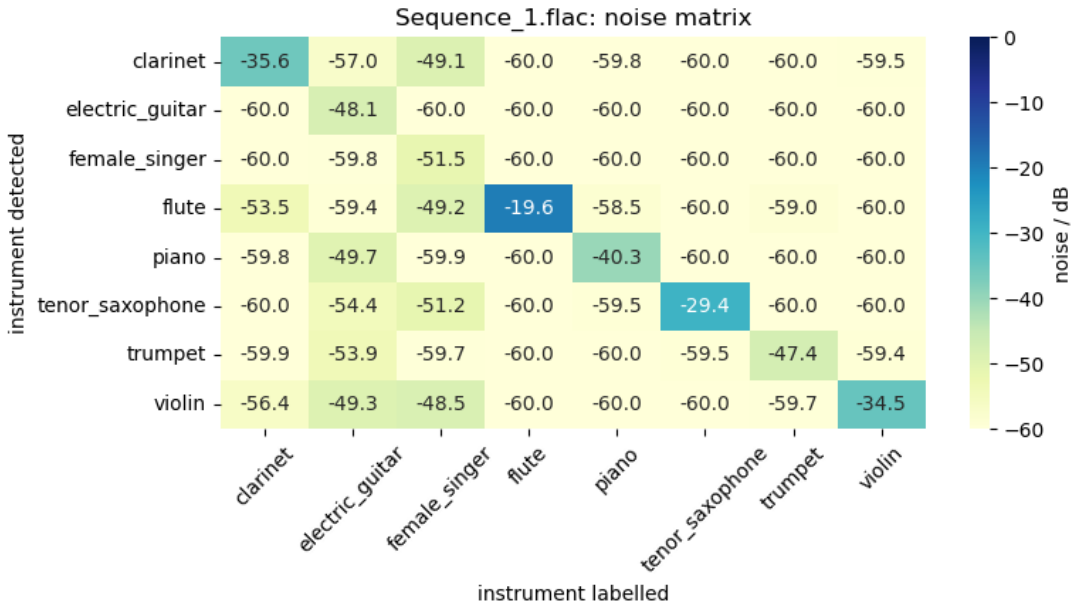


Figure 15: Noise matrix for Sequence1 (samples from test set).

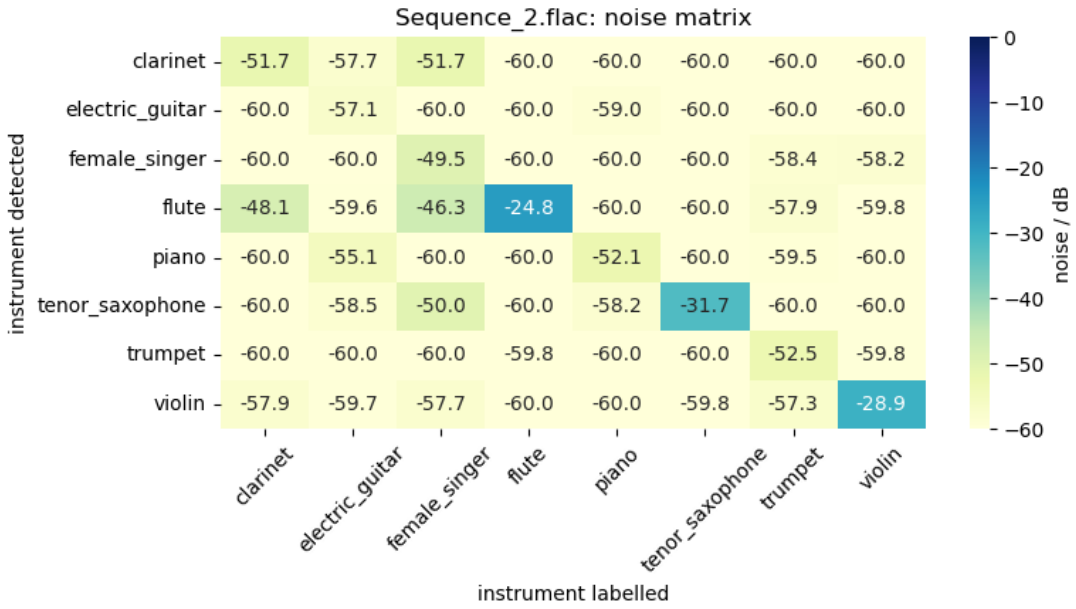


Figure 16: Noise matrix for Sequence2 (samples from freesound.org).

The results of the noise matrices are best interpreted by inspecting the female singer and the guitar: the female singer and the electric guitar were falsely detected within nearly every other instrument, since the values of their respective columns are comparably high. On the other hand when looking at their rows, no other instrument was falsely detected while they were active.

This means that the presence of these instruments in the network output is generally very dominant, and that the network is probably biased towards these instruments. This is why they also tend to generate noise (i.e. predictions where should be silence).

5.4.2 Leakage Matrix

Now we try to construct our variant of a confusion matrix from the labels and the predictions. In the leakage matrix, we want to analyze how much of the activation values of the currently playing instrument is being detected in another instrument. This means that we compare the predicted activation value of each instrument with the one currently active. Then, the difference derived from this comparison is subtracted from our defined dynamic range of $60\,\mathrm{dB}$.

Note that for the leakage it does not make sense to compare the labeled and predicted values of the same instrument, which is why there are no main diagonal values.

The leakage matrix for Sequence1 is shown in figure 17, the leakage matrix for Sequence2 in figure 18.

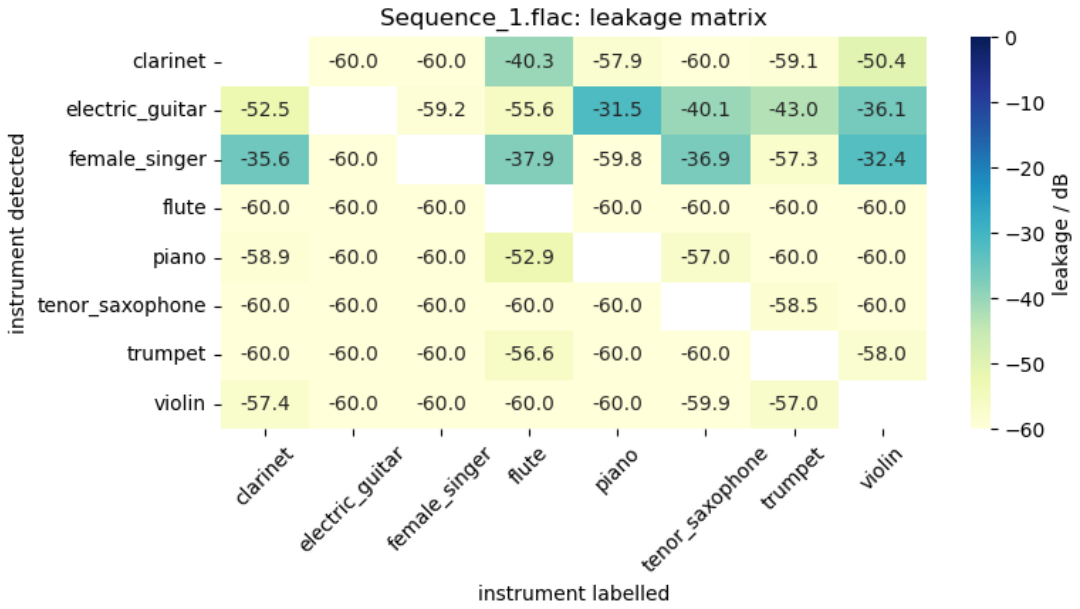


Figure 17: Leakage matrix for Sequence1.

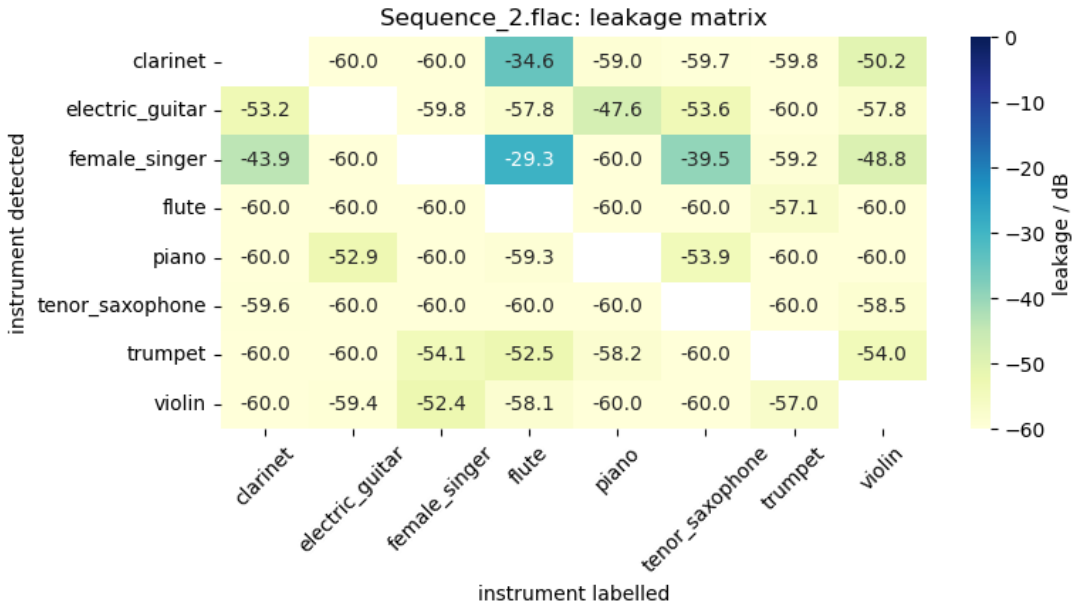


Figure 18: Leakage matrix for Sequence2.

Again we inspect the electric guitar and we find that it nearly does not leak into other instruments, as in the second column of the matrix (where electric guitar is playing) nearly all values are at $-60\,\mathrm{dB}$. The flute on the other hand is greatly leaking into clarinet, the female signer and the piano, which we can see in its column within the matrix.

5.5 Envelope

For the evaluation of the envelope the "real world" example was put under investigation. Figure 19 depicts the predictions for the mix sample (containing piano, clarinet and flute). Figures 20, 21 and 22 show the predictions for the single instrument tracks from which the mix was generated.

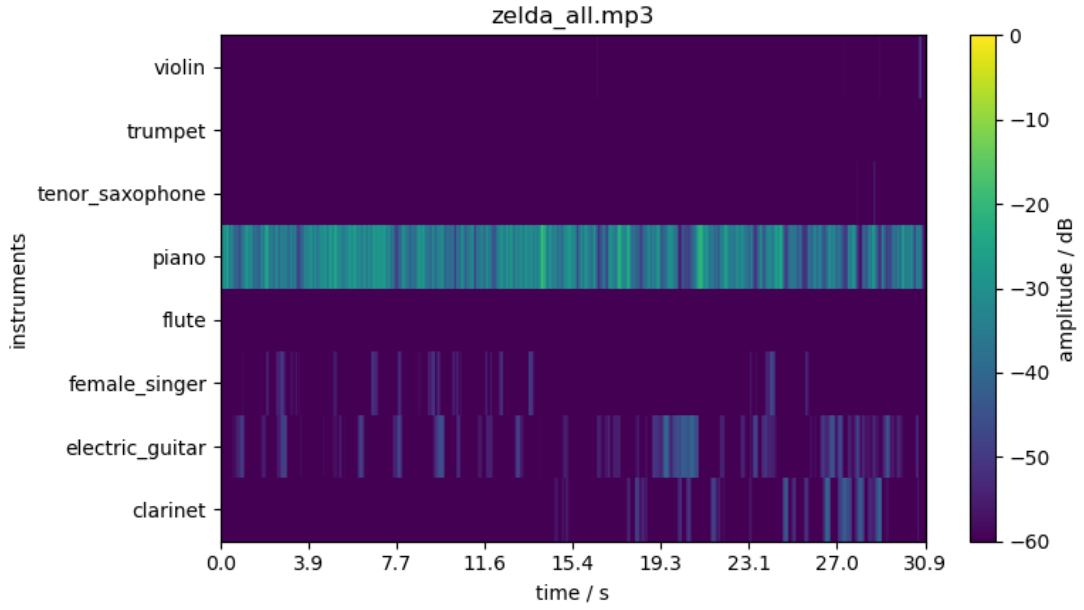


Figure 19: Prediction result of the mix signal.

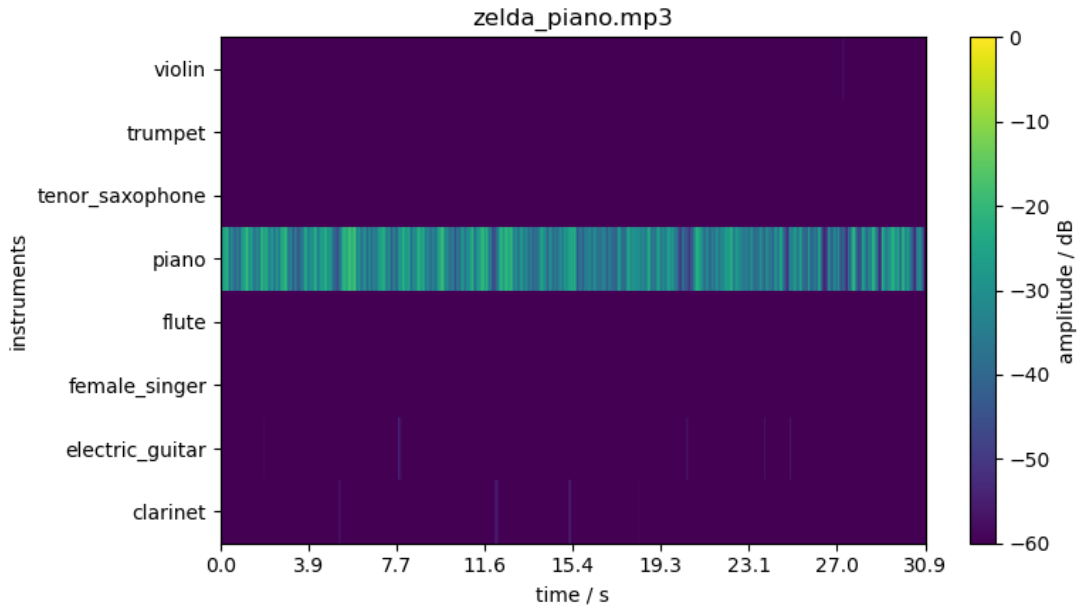


Figure 20: Prediction result of the piano track.

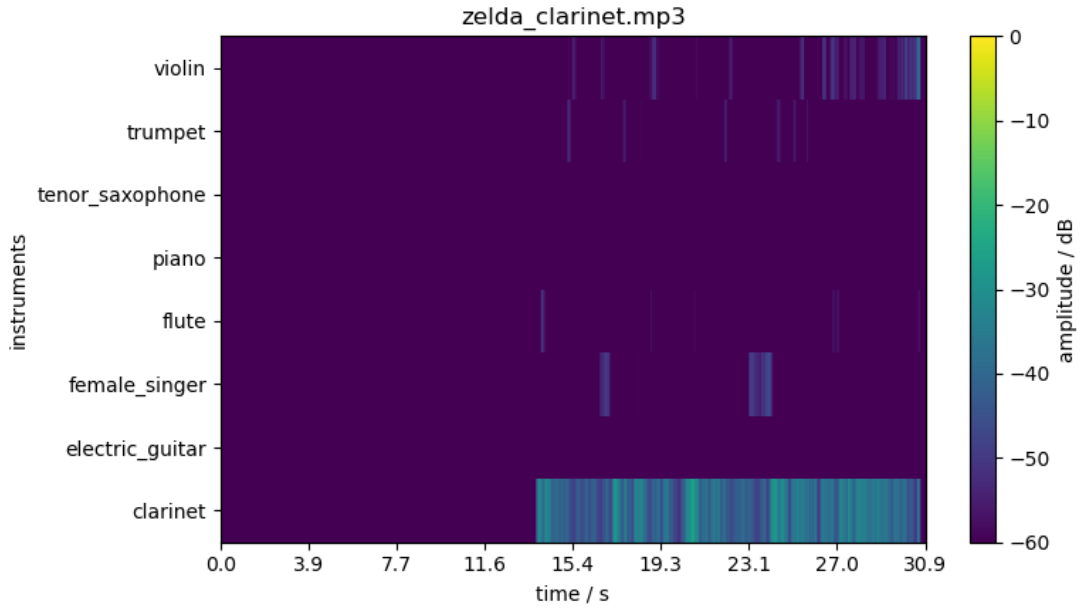


Figure 21: Prediction result of the clarinet track.

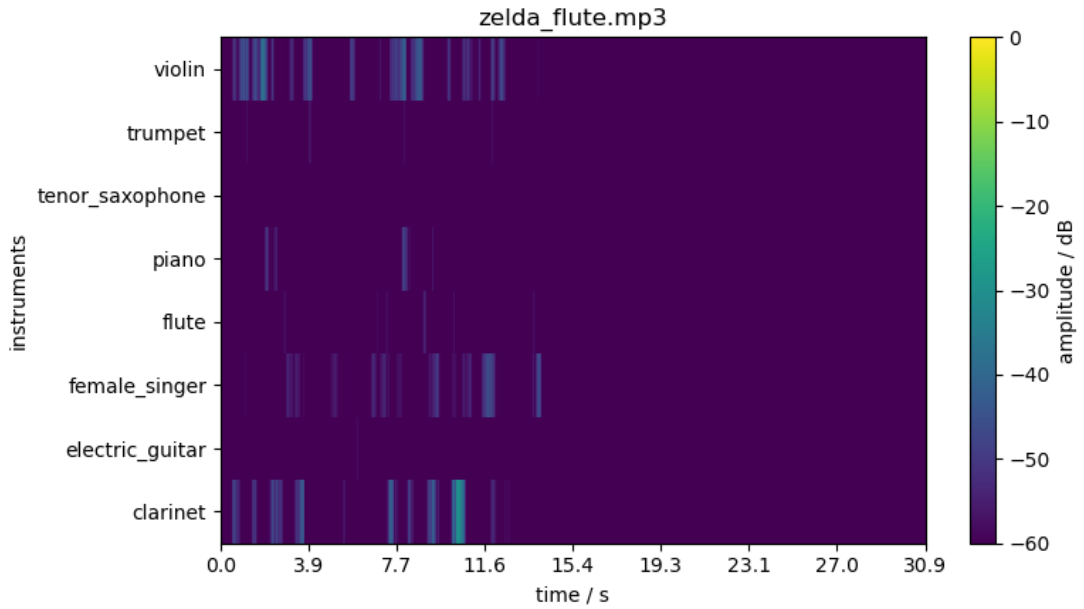


Figure 22: Prediction result of the flute track.

The first thing to be stated is the difference of detection quality between the mixed and the solo signals. The network has obvious problems telling the instruments apart, even when being trained on mixed signals. The detection of solo instruments on the other hand performs well on certain instruments (as can be seen for the piano and clarinet) but performs bad for others like the flute.

One thing the network seems to be well capable of (in respect to instruments it can detect sufficiently) is enforcing silence if an instrument is not active. On the other hand, the envelopes show similarities to the input, while still varying in amplitude and trend. They are also better detected for solo signals when compared to the mixes.

6 Conclusion

In conclusion it can be stated that our CNN is capable of learning and identifying various instruments. However, it only functions properly if it is fed single instrument samples. It can not properly distinguish all the sources of a complex mixture of multiple instruments.

The network is able to learn and reproduce complex envelopes. To achieve this we developed a data generation approach which included mixing and leveling the raw audio signals and converting them to features which could be reasonably interpreted by the network. The data set imbalance of the original data set used for training the CNN could be improved significantly as well.

There is room for improvement like applying audio effects to the raw audio recordings in order to create more diverse training data. Another way of improving the performance would be to take advantage of the two channels present in stereo recordings in order extract more information from the mixtures. Furthermore, an onset detection could be used to show the networks only relevant slices of the signal. This was implemented in a prototype of this project, but was not investigated further due to the difficulties it brings in relation to synchronicity and varying tempos. Lastly, a more sophisticated network structure, like an Encoder-Decoder approach, could be applied.

Usage for the underlying musical instrument detection approach could be guidance for source separation using algorithms such as Non Negative Matrix Factorization (NNMF) or automated transcription of music.

References

[1] H. F. Garcia, A. Aguilar, E. Manilow, and B. Pardo, "Leveraging hierarchical structures for few-shot musical instrument recognition," *CoRR*, vol. abs/2107.07029, 2021.

[2] K. Watcharasupat, S. Gururani, and A. Lerch, "Visual attention for musical instrument recognition," 2020.

[3] S. Gururani, M. Sharma, and A. Lerch, "An attention mechanism for musical instrument recognition," 2019.

[4] D. Podareanu, V. Codreanu, S. Aigner, and C. van Leeuwen, "Best practice guide-deep learning," 2019.

[5] A. Oppenheim and R. Schafer, *Discrete-time Signal Processing: International Version*. 01 2010.

[6] H. Austerlitz, "Chapter 4 - analog/digital conversions," in *Data Acquisition Techniques Using PCs (Second Edition)* (H. Austerlitz, ed.), pp. 51 – 77, San Diego: Academic Press, second edition ed., 2003.

[7] C. Schoerkhuber, "Applications of a constant-q transform for time- and pitch-scale modifications," 12 2011.

[8] J. Brown, "Calculation of a constant q spectral transform," *Journal of the Acoustical Society of America*, vol. 89, pp. 425–, 01 1991.

[9] G. LLC, "2018 kaggle machine learning and data science survey," 2018.

[10] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, "Array programming with NumPy," *Nature*, vol. 585, pp. 357–362, Sept. 2020.

[11] Brian McFee, Colin Raffel, Dawen Liang, Daniel P.W. Ellis, Matt McVicar, Eric Battenberg, and Oriol Nieto, "librosa: Audio and Music Signal Analysis in Python," in *Proceedings of the 14th Python in Science Conference* (Kathryn Huff and James Bergstra, eds.), pp. 18 – 24, 2015.

[12] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32* (H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, eds.), pp. 8024–8035, Curran Associates, Inc., 2019.

[13] O. Yadan, "Hydra - a framework for elegantly configuring complex applications." Github, 2019.

[14] "YAML: yaml ain't markup language." https://yaml.org/. Accessed: 2020-04-26.

[15] V. Lostanlen, C.-E. Cella, R. Bittner, and S. Essid, "Medley-solos-DB: a cross-collection dataset for musical instrument recognition," Sept. 2018.

[16] F.-R. Stoeter, M. Schoeffler, B. Edler, and J. Herre, "Human ability of counting the number of instruments in polyphonic music," *Proceedings of Meetings on Acoustics*, vol. 19, no. 1, p. 035034, 2013.