Lumen Project Documentation

1. Instrument classifier as a product

From ancient times human kind has been making music, and by extension musical instruments. It started with a pair of vocal cords and a couple of sticks, and ended up with an enormous plethora of instruments we have today. Music has been developing with humanity at a steady pace, so much, in fact, that in 2022 the music industry generated a global net revenue of 26.2 billion USD, and we decided we wanted to get in on that. As we've mentioned, throughout history music has been created in mostly the same way, a person learned to play the instrument and used those skills to entertain some general public. A noticeable problem we wanted to solve is how to *remember* music that sounded good so it could be replayed or reused. Thus musical notation was invented, and a couple of hundred years later music could be recorded analogy and finally digitally.

Today, a great deal of music is made using short samples of music which are combined and mixed to produce a desired sound. There even exists a market selling these samples back and forth, and in 2021 one of the websites hosting these exchanges made 141 million USD in revenue.

So there are truly many ways one can make money in the music industry, however using the aforementioned insights, and some strategic planning we narrowed our options down to two.
Either to become a soundcloud DJ or design some useful machine learning tool that the music industry needs. As none of us have any musical talent whatsoever, we decided we could try our luck with machine learning, after all there are more tutorials online on how to make a neural network than how to play a guitar. Lucky for us, LUMEN hosted a data science competition with that exact theme.

This year's LUMEN data science competition revolved around music. The goal is to create a machine learning system that can identify the presence of a given number of instruments. As not to complicate things too much there were 11 instruments in total we had to identify: cello, violin, flute, clarinet, piano, saxophone, electric guitar, acoustic guitar, trumpet, organ and human voice. More formally, this problem boils down to a multilabel classification task. This means that for each of the possible instruments, we need to determine whether it is present or not. Instrument labels of the final set on which our solution will be tested are not available to us, so it's important for our solution to be robust. The official metric for tracking and comparing the performance of our solution is the fraction of labels which are correctly predicted, usually denoted as the hamming accuracy or multilabel accuracy. The final score is then calculated as the mean of the accuracy over all examples.

Since a substantial amount of music today is made using samples, it seemed obvious to check the needs that part of the industry has. A simple problem popped up: when people are buying samples they are usually looking for specific instruments or their combinations, which means each audio sample needs to be labeled with the instruments it contains. Furthermore, labeling audio samples is much more tedious than labeling images. The latter can be done faster, and in parallel for multiple images, as with audio one needs to listen to every sample, one by one and label the instruments present which also may prove difficult to the untrained listener. This seemed like a perfect job for an AI system.

Several other product ideas came to us during our exhaustive brainstorming sessions, one implementation of audio instrument classification would help DJs in playing music. DJs often have to listen to the songs they want to play in the future simultaneously while playing the one you hear at the party. The reason is they're listening for the next sound to play, often a DJ might want to add a specific sample from the entire mix, for instance a start of a groovy saxophone to get the crowd into the right mood, the start of those samples are called cue points. With these sorts of machine learning algorithms DJs would no longer need to listen over large parts of the song to find specific one cue point, rather our software could find the instrument in question so the DJ would need to search a much shorter part of the entire mix. The actual implementation of that search procedure and its integration to existing DJ equipment and software lies beyond the scope of this project.

2. Data

IRMAS dataset

The initial dataset we were given was the IRMAS dataset. It is composed of audio tracks which have one or more instruments playing for a short period of time. The possible instruments are: clarinet (cla), piano (pia), flute (flu), trumpet (tru), electric guitar (gel), acoustic guitar (gac), cello (cel), organ (org), violin (vio), saxophone (sax), and human voice (voi). Although there are plenty of more instruments available in the world we limit our model to these instruments. The full IRMAS is split into two large groups: *IRMAS train* that contains only single instrument tracks, while *IRMAS validation* contains both single instruments and multi instrument tracks. Ironically both of them were used for training in some capacity, so to avoid confusion we from now on refer to them IRMAS single instrument and IRMAS multi instrument.

A comment about the metrics

Taking into account the aforementioned training set properties, it is fairly easy to achieve high accuracy on this dataset. Since the labels for each of the single-instrument set examples correspond to exactly one instrument, this can be denoted by a one hot vector. Just by predicting a vector of all zeros, we can easily achieve 90% accuracy. Of course, this doesn't correlate to the multi-instrument set very well, since the amount of instruments ranges from 1 to 5, but it still does achieve rather high accuracy. A more suitable metric for this particular dataset might be the F1 score, which is calculated using precision and recall. Precision takes into account the amount of true positives and false positives, while the latter focuses on true positives and false negatives. It's much harder to fool these metrics the same way as accuracy so we will keep track of them during our experiments. Also we keep track of the classification metric per instrument as well as the mean value across the instruments. **Note:** Since the F1 metric is dependent on the decision threshold, we need to specify which value of threshold was chosen. For most experiments it was set to 0.5, and F1 graphs are plotted for that value unless stated otherwise.

Splitting the data

For training we devote the entire IRMAS single instrument plus 50% of the IRMAS multi instrument dataset, and for validation we use the remaining 50% of the IRMAS multi-instrument. However this sort of splitting needs to be handled with care, since this opens a door for potential data leakage. The IRMAS multi-instrument dataset consists of a total of 208 songs, songs are split into multiple files according to the instrument that is played in the segment in such a way that all of the labeled instruments are present in every instant of the audio track. A simple random split here would doubtless place pieces of the same song into the train set as well as the validation set if this happens it might give a skewed overestimation of the model performance. We grouped all the files according to their song name and assigned each group with the instruments present in the entire song. We then split this song dataframe in half in such a way that the label balance was preserved, this was done using multi label stratification via scikit-multilearn's iterative_train_test_split. This assured that no song was leaked between two datasets. Once we add the multi-instrument part of IRMAS to the single-instrument part we end up with a new dataset we named Boosted IRMAS.

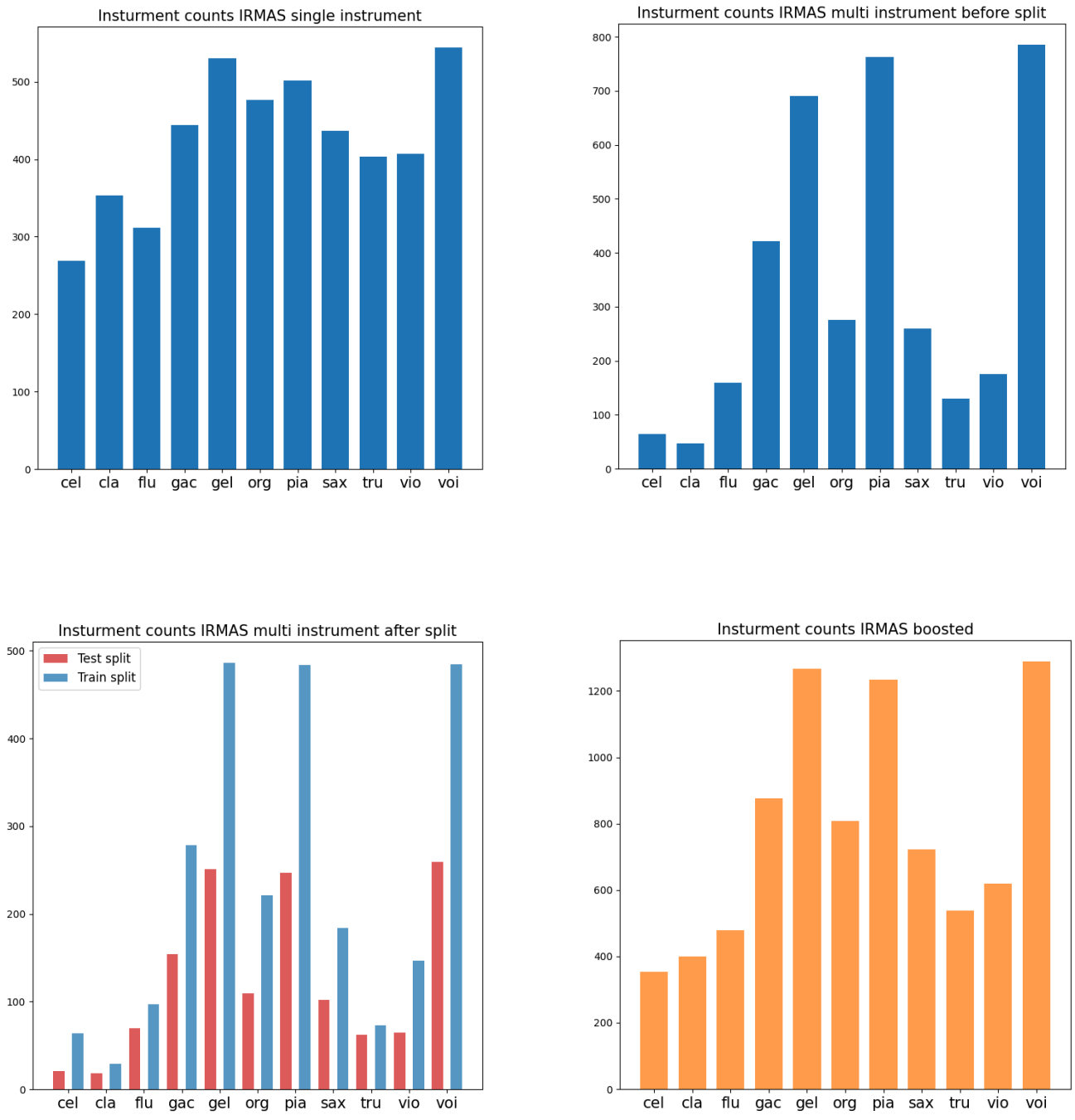


Figure: Instrument distributions. Top left IRMAS single-instrument, top right IRMAS multi-instrument before splitting, bottom left IRMAS multi-instrument after splitting, bottom right IRMAS boosted.

Label imbalance solutions

As we've already mentioned both IRMAS single-instrument and IRMAS multi-instrument sets suffer from class imbalances, as well as sparse labels meaning the number of instruments present in a song is often low. If these issues are not properly handled they can have significant detrimental effects on the model performance. We decided to approach these issues from several angles of attack. We implemented a so-called positive weight to the loss function, we also tried waveform adding and lastly we experimented with different sampling schemes. The first two solutions will be described in sections of their own: the positive weight is explained in the context of the loss function while waveform adding is placed among augmentations. Although all of these approaches have their merit there is no better

way to fix issues with data than with more data, to this end we also added two more musical datasets to support IRMAS in training. The datasets in question are OpenMIC and AudioSet, both will be briefly commented on in the following subsections.

Weighted random sampler

During training however, we sample data in a randomized fashion, aside from a simple random sampler we also used a weighted random sampler. A weighted random sampler accepts an array of weights assigned to each sample, the weight is proportional to the probability of the sample being drawn. The weights were calculated for each instrument as a fraction of the total number of tracks divided by the number of tracks containing the instrument in question. If we have a dataset with 20 tracks with only pianos, 10 with pianos and violins and 50 saxophones the weight for pianos is 80/30. Single instrument samples were assigned a weight of the instrument present, while multi instrument tracks were assigned a median of instrument weights present in the track. This makes the distribution of labels more uniform.

IRMAS dataset sample relabeling

Bearing in mind the inconsistency between the IRMAS single-instrument and IRMAS multi-instrument datasets, we decided to relabel a sample from the train set to determine the severity of the differences. The sample consisted of 297 examples, where 27 examples were randomly taken from each label in the IRMAS single-instrument dataset. We noticed that 47.97% of instruments are completely correct, meaning the instrument from the original label is equal to the new label, and that only it is present in the audio sequence. We also wanted to calculate if the original instrument is even there. We determined that 13.85% of the relabeled examples do not include the original label. Out of the relabeled examples, 159 of them were labeled by a single instrument, 127 by two instruments and 10 by three instruments. This meant that 46.23% of the single instrument dataset actually contains more than one instrument.

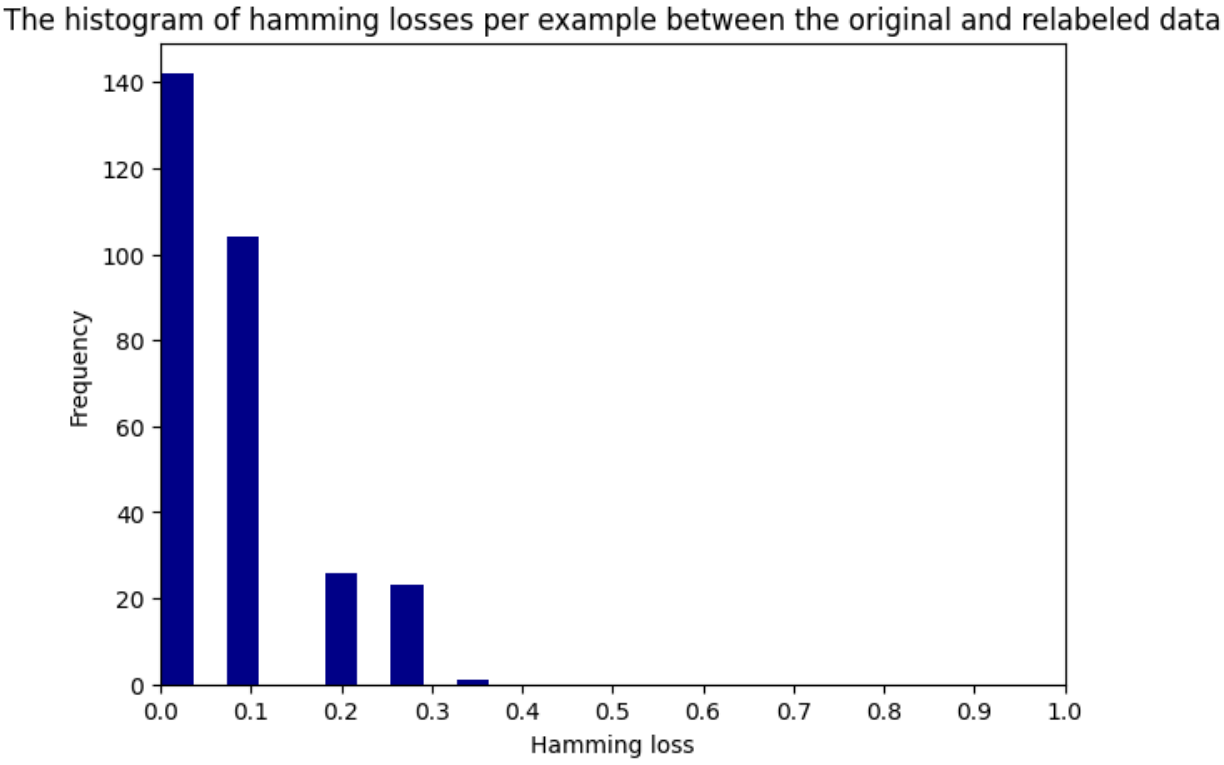


Figure: The hamming loss (distance) distribution between the original labels and new labels.

The figure above represents the hamming loss (distance) distribution between the original labels and new labels, indicating that most often no instruments are incorrect or only a single instrument is incorrect.

Exploratory data analysis of IRMAS dataset

As it was already mentioned, the IRMAS dataset is divided into two subsets named the IRMAS single-instrument (train) dataset and the IRMAS multi-instrument (test) dataset. Each of the subsets is analyzed separately and here we note the results of the analysis.

The IRMAS single-instrument dataset

Training set contains 6705 files with a duration of 3 seconds which adds up to the duration of 20115 seconds overall. Also, the duration of each instrument in the set was calculated and depicted on the figure below.

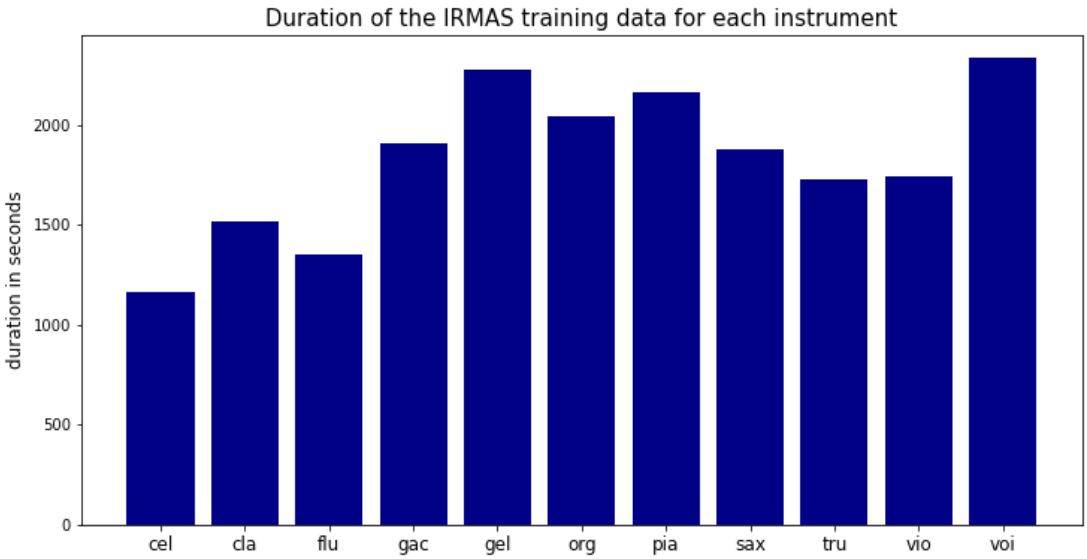


Figure: Total duration of instrument tracks per instrument in IRMAS single-instrument dataset.

On the given plot, we can obviously see the imbalance between the instrument classes. For example, far more examples contain voice than, for example, cello. This fact can affect the metrics in the end because the model will see more voice than other classes and it may perform better in recognizing the voice patterns. So, it would be a good idea to try and balance the classes to be roughly equally represented.

The IRMAS multi-instrument dataset

Same analysis was made for the IRMAS multi-instrument dataset. It was shown that this dataset contains 2874 examples of duration between 5 and 20 seconds. The audio sequences come from 208 songs and the whole dataset is 48531 seconds long. On the blue histogram below, it is shown how much each instrument is represented in the multi-instrument dataset. As in the IRMAS single-instrument dataset, human voice is far more present in the dataset than clarinet, trumpet or cello. For that reason, we can expect better classification results for the human voice than for those less represented instruments.

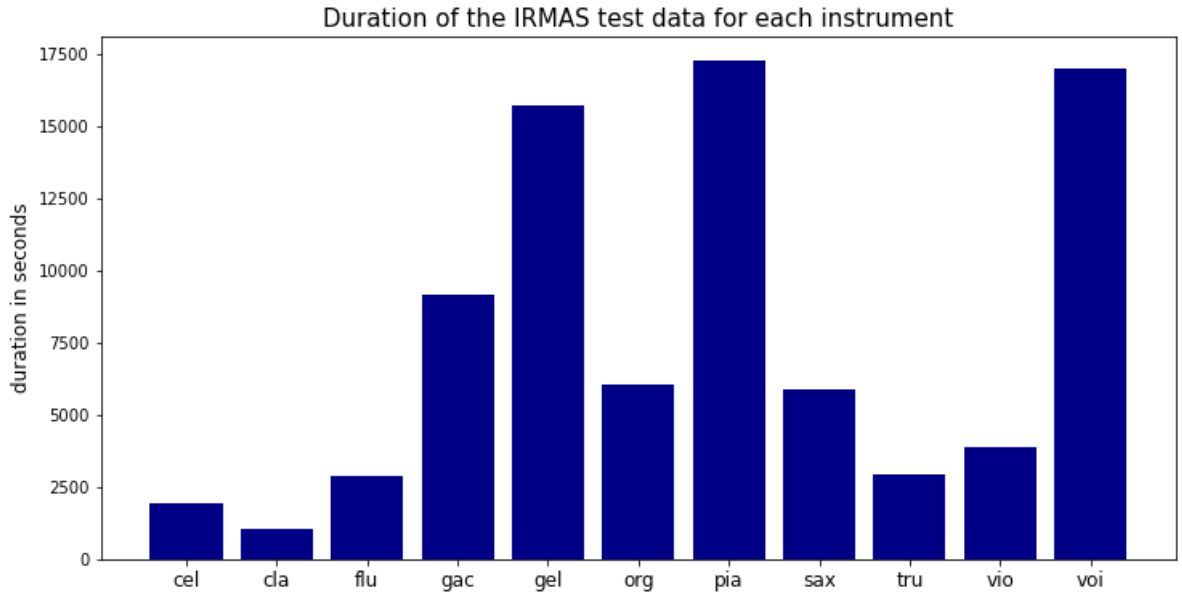


Figure: Total duration of instrument tracks per instrument in IRMAS multi-instrument dataset.

On the other hand, on the red histogram below, we can see the distribution of duration of examples. We note that the most examples are about 20 seconds long. If we chunked these long examples in shorter intervals, e.g. 3 seconds, there would be much more files in the test dataset than in the training dataset. This suggests that it is a good idea to transfer part of the test data in the training set.

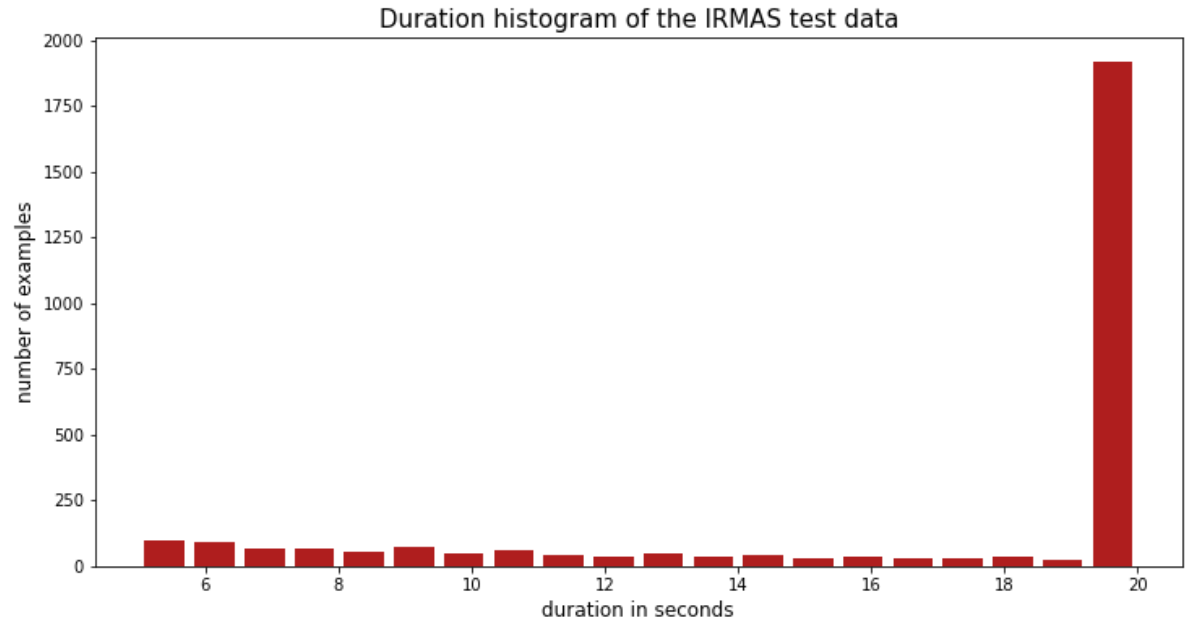


Figure: The counts of the duration of audio sequences in the IRMAS multi-instrument dataset

Analysis of classical features:

We can extract a lot of information from the given .wav files. For example, we can take a Fourier transform in order to find out something about the frequencies in the example or we could construct a spectrogram. There are many ways to obtain useful information from the data. In this subsection, we will focus on the classical features that can be derived from the given signals. Although 28 features were used in this analysis, we will investigate more closely only a few which have the highest variance among the instruments. Those features are:

- Roll-off frequency
  - The frequency under which some percentage of the total energy of the spectrum is contained (distinguishes harmonic and noisy sounds).
- Spectral centroid
  - The spectral centroid indicates where the center of mass of the spectrum is located and it measures the brightness of a sound.
- Spectral bandwidth
  - The spectral bandwidth is the wavelength interval over which the magnitude of all spectral components is equal to or greater than a specified fraction of the magnitude of the component having the maximum value.
- Mel frequency cepstral coefficients
  - A small set of features (10 in our examples) which concisely describe the overall shape of a spectral envelope and the sound timbre.

These features vary the most between different instruments so they should be the most insightful regarding the classification. We also used and examined other classical features such as the zero crossing rate, root mean square energy, spectral contrast, spectral flatness and various types of the entropies.

In the figure below, we can see the correlation matrix for the labels which means it shows how correlated is the appearance of one instrument with the appearance of another in the IRMAS multi-instrument dataset. It makes no sense to make this analysis for the IRMAS single-instrument dataset as there are no examples that contain more than one instrument there. We can notice that instrument incidences are correlated for some pairs of instruments more than for others. For example, we see that piano is highly inversely correlated with the acoustic and electric guitar which makes sense as guitars appear in more modern (rock and roll and similar) genres while the piano is an older and more classical instrument so we do not expect them to show up together quite often.

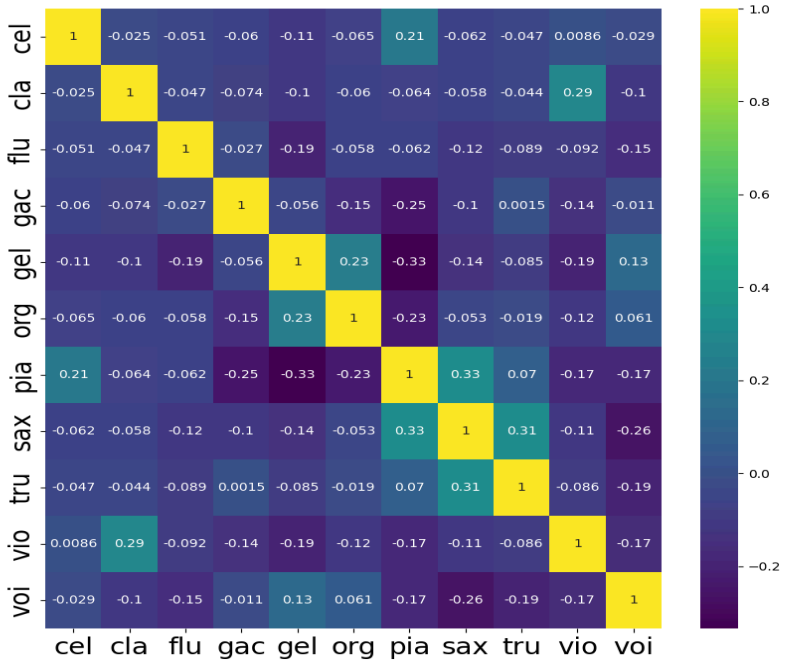


Figure: Correlations between instrument occurrences in the IRMAS multi instrument set. The highest levels of correlation are present for saxophone and piano, trumpet and sax, and lastly violin and cello.

It is also interesting to note that the voice is inversely correlated with the piano which is not surprising as we would far more often hear voice with an electric guitar then a piano, organ or cello. Additionally, there is correlation between saxophone, trumpet and piano which is typical for jazz and similar so it also makes sense that correlation between them is high. Violin and cello are positively correlated which makes sense as they are both classical instruments so we expect them to show up simultaneously in an audio file.

Now we wish to get some intuition about the IRMAS single-instrument and multi-instruments datasets and the aforementioned features. We could simply plot the features but the problem is that they are vectors in a 28-dimensional feature space. It is obvious that if we want to visualize our data, we should reduce the number of dimensions and that is done with the principal component analysis (PCA). We use PCA techniques to reduce the number of dimensions from 28 to 2. This obviously results in information loss but it is not a problem because we only want a rough visualization of data. When we transform the data into 2-dimensional space, we can plot it and the acquired plots are shown below. Plots are made exclusively for the IRMAS single-instrument dataset. For the IRMAS single-instrument dataset, only one plot is enough because of the fact that it contains only single-instrument examples. Those plots show which points (audio files) in this new 2-dimensional PCA feature space contain the given instrument. In the IRMAS single-instrument plot, it is obvious from the legend which color corresponds to which instrument. For the IRMAS single-instrument dataset, cumulative explained variance ratio is 0.59 so we can be confident in these results. Also, it is worth noting that, as we noticed earlier, there is a big disbalance in the count number of each instrument.

Figure: PCA of IRMAS single-instrument using 2 components

OpenMic

Description

The OpenMIC dataset consists of 20000 examples that are 10 seconds long with 20 possible instruments as labels. It contains multilabel as well as single labels examples. If we wish to use it we first need to format it in the same way IRMAS is formatted, that is we removed all of the tracks that didn't contain at least one IRMAS instrument. This

meant that some songs had IRMAS instruments as well as some non IRMAS instrument playing, we believe this makes the model more resilient to noise.

The main problem when properly formatting the OpenMIC dataset is handling labels, a major part of the dataset was labeled using crowdfunded labelers, while a non negligible part was labeled using pre-trained models for automatic labeling. This means that the labels instead of being binary values signaling the presence or absence of an instrument, are confidence scores that assign a level of certainty that an instrument is present, they range between 0 and 1. We assigned the instrument label as 1 if its confidence score is above 0.8, otherwise we assumed the instrument was absent. We used this dataset for two reasons: adding more multi labeled data, and balancing the instruments with lower counts.

Handling guitars

One issue with OpenMIC is that the labels do not distinguish between an acoustic and electric guitar, instead they are only labeled as a guitar. We decided we were too lazy to reannote all of the guitars manually, so we had to use some clever labeling scheme. Lucky for us we had a large Audio Spectrogram Transformer at our disposal and used it to create embedded vector representation of electric guitars and acoustic guitars from IRMAS, next we embedded all the OpenMIC guitars as well. Since the AST was pre-trained on a large dataset and with a large number of classes we trusted it could distinguish electric from acoustic guitars. In this embedded space we trained a logistic regression on IRMAS guitars, and used it to classify OpenMIC guitars as electric or acoustic.

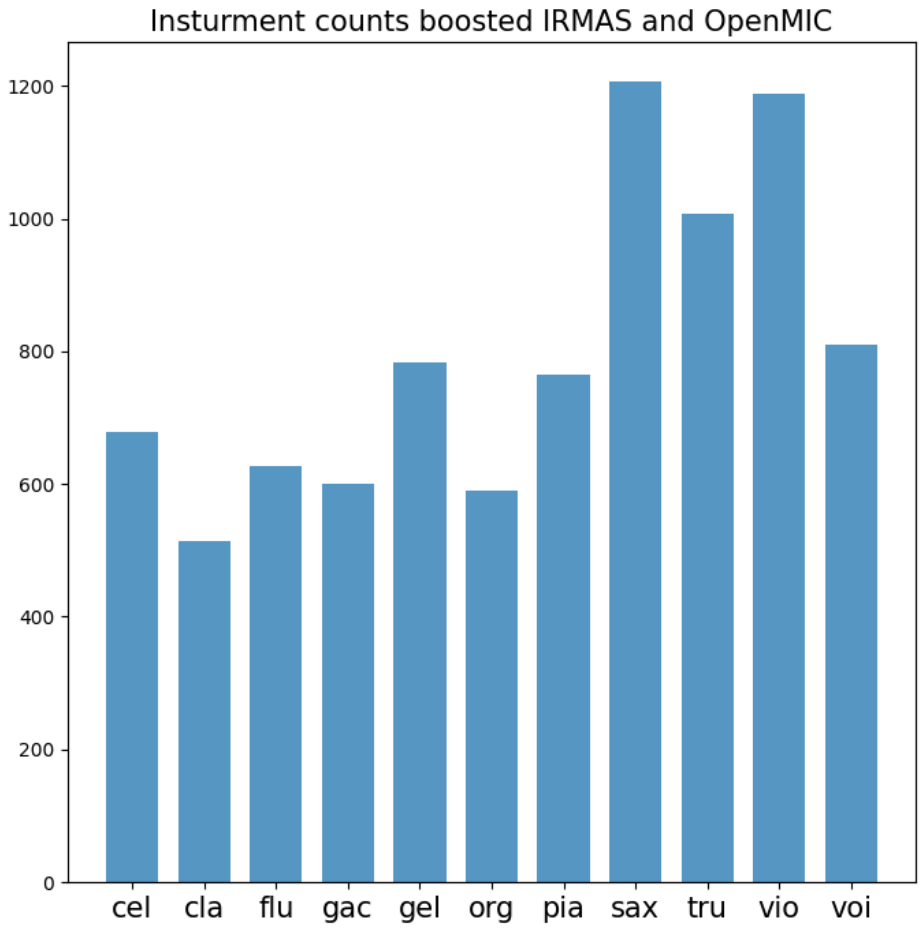


Figure: Boosted IRMAS + OpenMIC instrument distribution.

AudioSet

AudioSet consists of an expanding ontology of 632 audio event classes and a collection of 2,084,320 human-labeled 10-second sound clips drawn from YouTube videos, however we used only the first seconds of each audio track. At the time of this writing it is the largest audio dataset available. It has well curated labels and fairly high quality sound tracks. Among the 632 audio event classes we were interested in only 11 that corresponded to

IRMAS instruments. We scraped a total of 15174 single instrument tracks using the following script: https://github.com/aoifemcdonagh/audioset-processing We also had to remove repeating tracks or low quality tracks and the details on data cleaning are provided in the following section. Repeating tracks What we are left with is 13050 tracks with the following instrument distribution. This part of the data was downloaded fairly late throughout the project so most results we present were not trained using this data. When we combine the Boosted IRMAS, OpenMIC and AudioSet we end up with the *Ultimate dataset*.

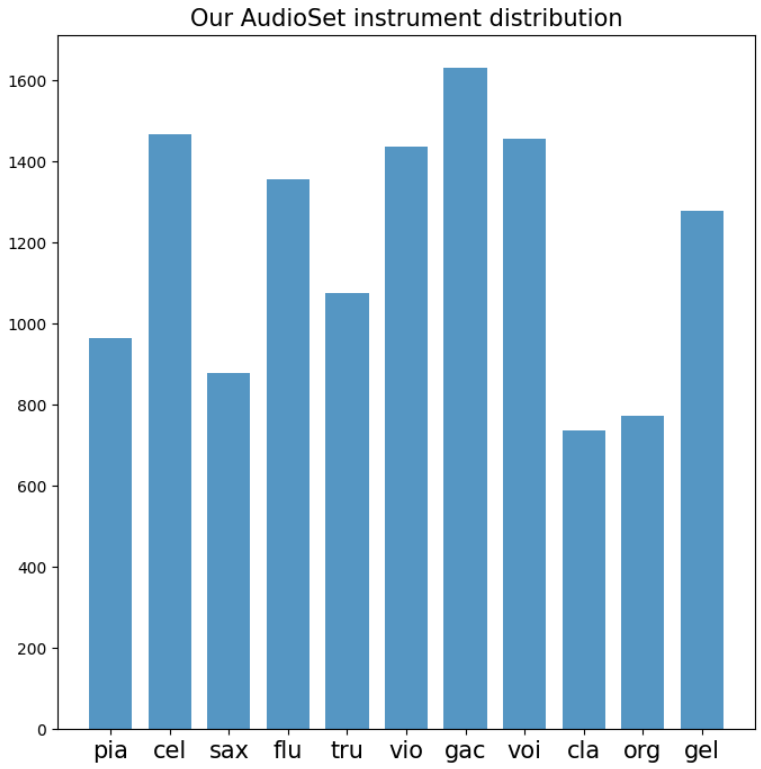


Figure: Our AudioSet instrument distribution.

Handling difficult examples

Obvious outliers

While IRMAS is mostly well curated this is not true for OpenMIC or AudioSet. So it falls to us to curate the data as well as possible, it's as uncle Ben said: *With great data comes great responsibility*. First we wish to handle files with faulty audio; we removed the tracks that last less than a second as well as audio files that are *silent.* We considered an audio silent if its amplitude is below 0.001 more than 80% of the time. For this definition to make sense the audio needs to be normalized so the values in the waveform are between -1 and 1.

Embedding outliers

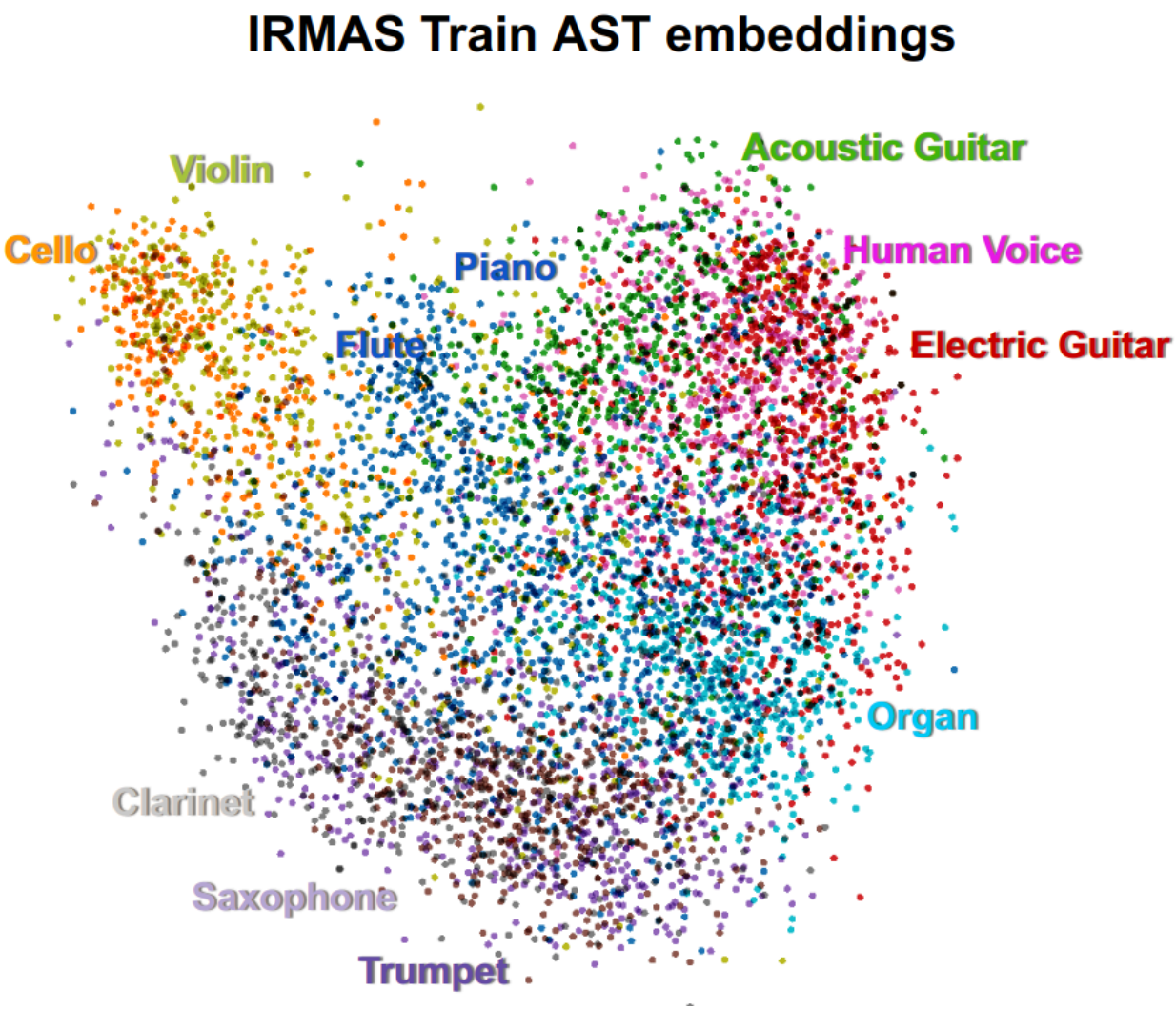


Figure: AST feature embeddings for every file in the IRMAS dataset projected on their principal components of variance (PCA). The CleanLabs classifier is trained in this (unprojected) embedding space to find outliers. It's interesting to note that even in the projected space instruments cluster intuitively, cello and violin are close together, as well as both guitar are in nearby clusters.

Every dataset contains a non negligible amount of mislabeled, noisy and anomalous data. If not taken care of these examples can have a significant detrimental effect on the models performance. These examples are much harder to find than the ones mentioned in the previous subsection, so in order to remove the noisy or mislabeled data points we used the cleanlab tool.

CleanLab is a data centric model used for identifying difficult examples. The procedure is as follows. First we need some representation of our data, and to this end we used an Audio Spectrogram Transformer (AST) model to produce an embedded representation of the audio. More will be said on the AST itself in its own chapter, but here we only use it to produce embeddings. Figure above shows the PCA projection of the embedded data points. The advantages of the AST model is that it was pretrained on the AudioSet dataset and can produce meaningful representations for various possible labels for music, aside from only instruments. Once we have the embedding we train a classifier to produce probability scores for each example, in our case we used logistic regression. The classifier is trained several times using k-cross validation (in our case k=10), furthermore the label probability scores are produced for the withheld part of the dataset. Once this is repeated several times we have a label probability score for all of our data points. The data points are then ranked in the ascending order of their label probability score, see Figures below for openMIC and IRMAS datasets. The probability score is between 0 and 1, so we decided to discard all of the data points with particularly low probability scores.

This procedure was performed on IRMAS and AudioSet separately by discarding labels with probability scores lower than 0.11 for IRMAS and 0.13 for AudioSet.

OpenMIC guitar cleaning

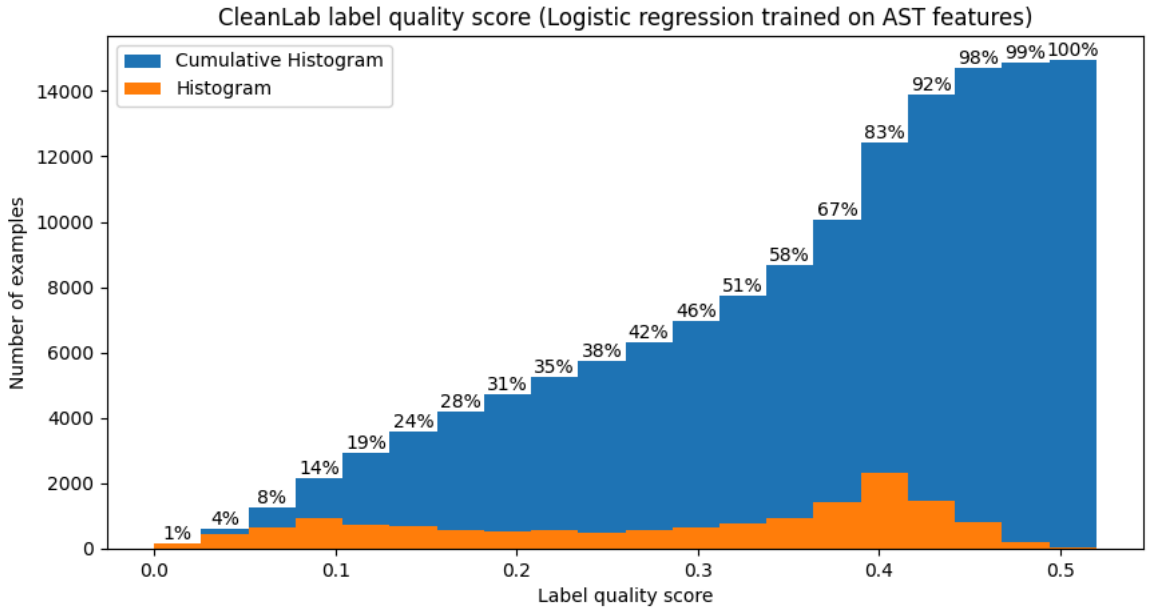


Figure: AudioSet label quality scores, all audio tracks with score lower than 0.13 are removed from the dataset.

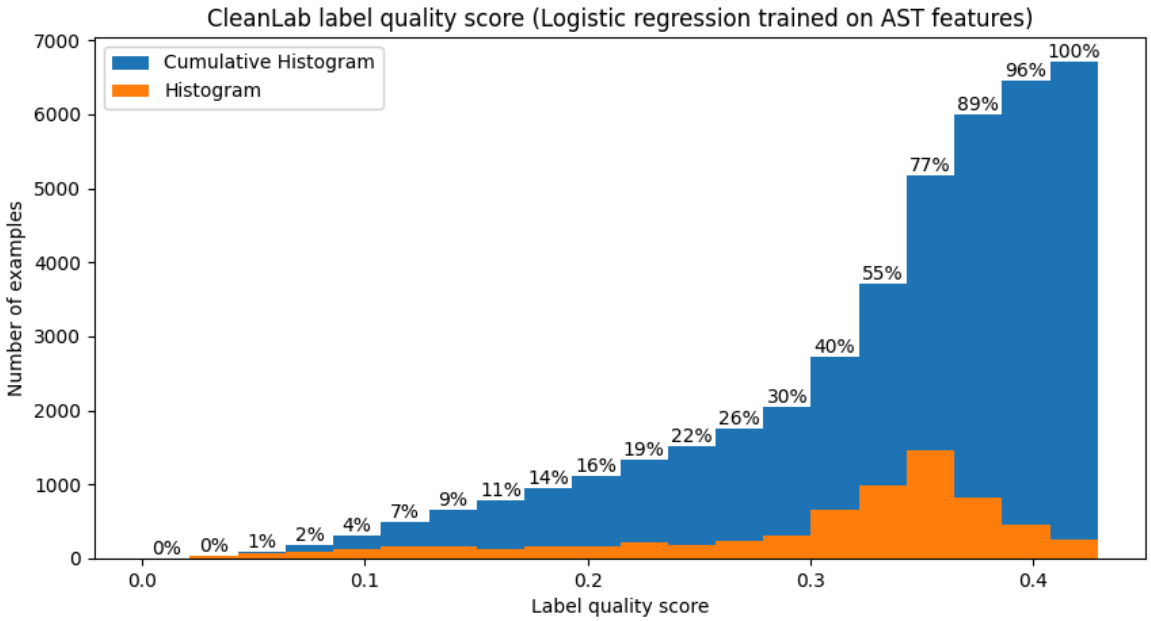


Figure: IRMAS dataset ranked by label quality score, tracks with score lower than 0.11 are removed from the dataset.

Table: Number of examples removed from dataset

Dataset	Percent dropped	cel	cla	flu	gac	gel	org	pia	sax	tru	vio
IRMAS	0.074	35	36	45	50	68	37	65	40	58	44
AudioSet	0.156	272	0	292	286	155	194	169	195	239	297

Lastly, we present the training evolution of a ConvNext network to showcase a slight improvement which is gained by removing poorly labeled examples.

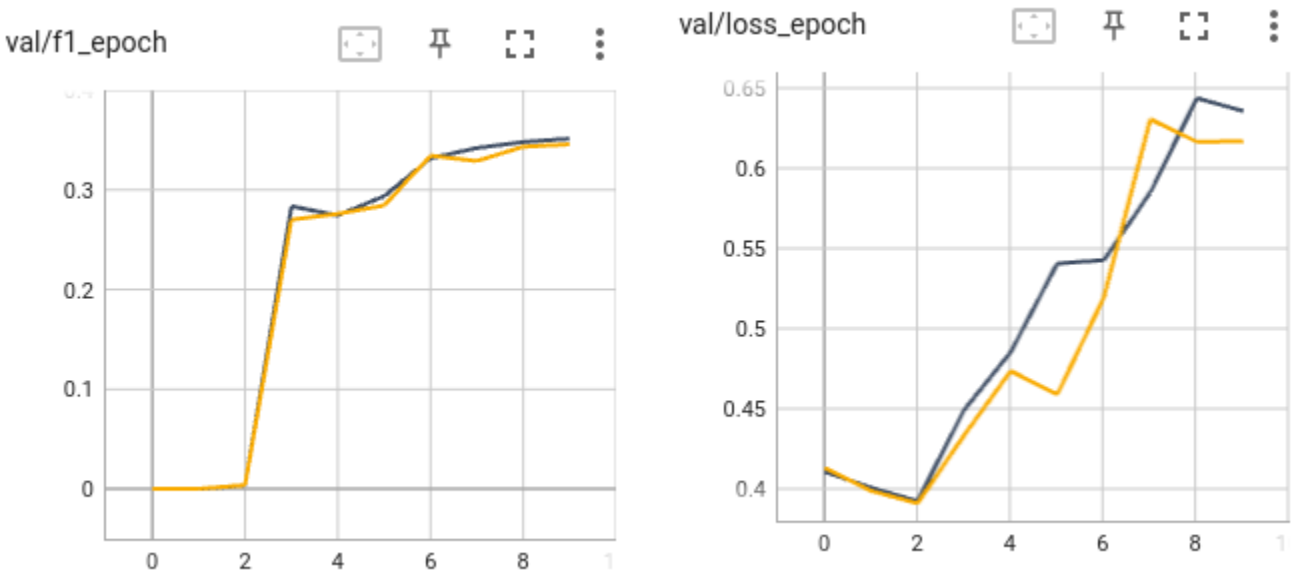
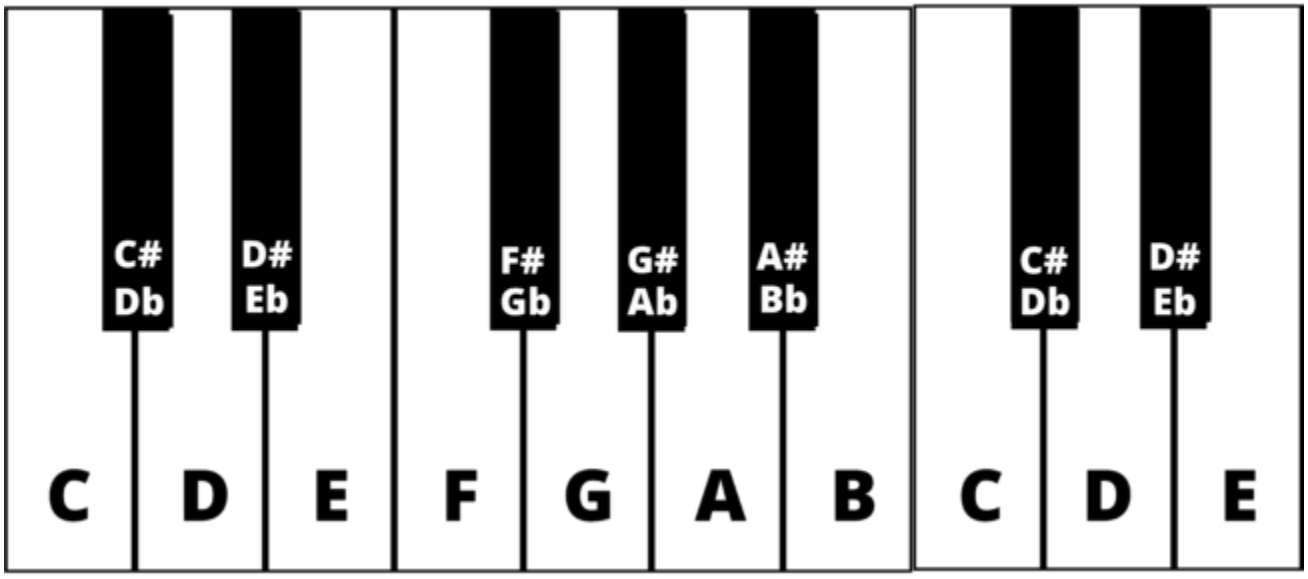


Figure: A slight improvement of the F1 metric can be seen by removing difficult examples from the IRMAS single instrument dataset. The model in question is ConvNeXt in both cases, they were trained with no augmentations, with the same learning rate of 1e-4. The black curve represents the model trained without outliers, while the orange curve represents the model trained on the original dataset.

What about the music?

Instead of just focusing on the audio aspect of the data, we wanted to take a look at the musical aspect of it. In order to understand the terminology, there is one important term to get acquainted with, and that is tonality. Tonality (or key) is defined as the arrangement of the pitches and chords in a piece of music. In western music, there are 12 tones in an octave, usually denoted by letters from A to G. A keen reader would notice that there are only 7 letters between A and G, both ends included, which is why sharp and flat notes are introduced. Therefore, the possible notes in an octave are: C, C#/Db, D, D#/Eb, E, F, F#/Gb, G, G#/Ab, A, A#/Bb and B. The picture below denotes (pun haha) the notes on a piano. In the context of tonality, the note that best describes the tonality is the root of the key.



It is important to understand the other aspect of tonality which is whether a minor or major key is used. People usually describe songs in major keys as happy, cheerful or bright sounding, while songs in minor keys are

usually described as sad or melancholic. Since it's hard to encapsulate this aspect in words, an example of a melody in a major key would be "Happy Birthday", "Jingle Bells" or "Never Gonna Give You Up". Examples of songs in a minor key are "What is Love", "Africa" and "Sude mi". A song doesn't have to be in a single key e.g. "Zaustavi se vjetre" verses are in E minor but the chorus is in G major or the aforementioned "Africa" which uses 3 minor keys. Taking these two aspects of tonality into account, we can come to the conclusion there are possible 12 minor keys and 12 major keys. The key of the song is determined using the Krumhansl-Schmuckler key-finding algorithm, which correlates the prominence of each pitch class in a sample with typical profiles of a major key and a minor key, and returns the key with the highest correlation coefficient.

Now that the terminology is explained and the reader has basic intuition of the tonality we can take a look at the data. We had some assumptions regarding the key of the songs in the dataset. For the classical genre, we didn't have any expectation, while these are our assumptions for the rest:

- Country folk - more major than minor
- Jazz Blues - more minor than major
- Latin soul - more major
- Pop rock - more minor

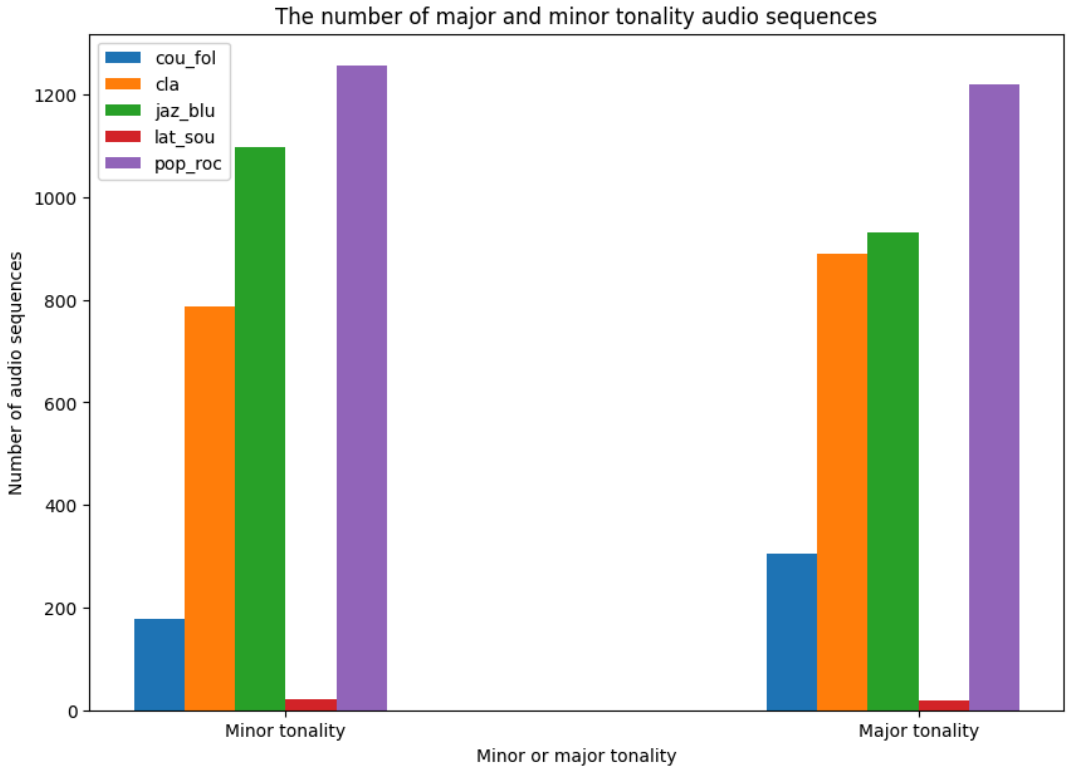


Figure :The major/minor aspect of the key is fairly balanced, with 3341 minor audio sequences and 3364 major audio sequences. Next, we decided to take a look at the key with respect to genre.

We can see that our assumptions for the country folk and jazz blues genres hold, while the pop rock genre hypothesis barely holds. We speculate this is because the dataset creators took a simplified approach when defining genres and that the pop rock genre is actually a mixture of multiple genres. It doesn't make much sense to reflect on the latin soul assumption since there are only 42 audio sequences of this genre.

We can now take a look at both aspects of genre. The analysis uses the assumption that an audio sequence is in the same key, even though this is not necessarily true, although it is sensible due to the short length of audio sequences. The plot below shows a histogram of audio sequences for each tonality. We can clearly see that some roots stand out in the major group like C, D, E, G and A. For the minor key, the most frequent roots are C, D, G and F. An interesting thing to note is all of the most frequent keys aren't sharp or flat keys. Why is this? Let's take a look at the simplest scale: the C major scale. It consists of the notes: C, D, E, F, G, A and B. This key doesn't have a single sharp or flat in it. Conveniently, the C major scale notes correspond to the white notes on the piano, meaning that it is somewhat

easier to play in this key. The same principle can be extended to different instruments. For the major key group that stands out, the key with the most sharps or flats is E major, and the number of sharps in this key is 4. For the minor key group that stands out, the key with the most sharps or flats is F minor, and the number of flats in this key is 4. This insight indicates that composers pick keys to make the songs easier to play.

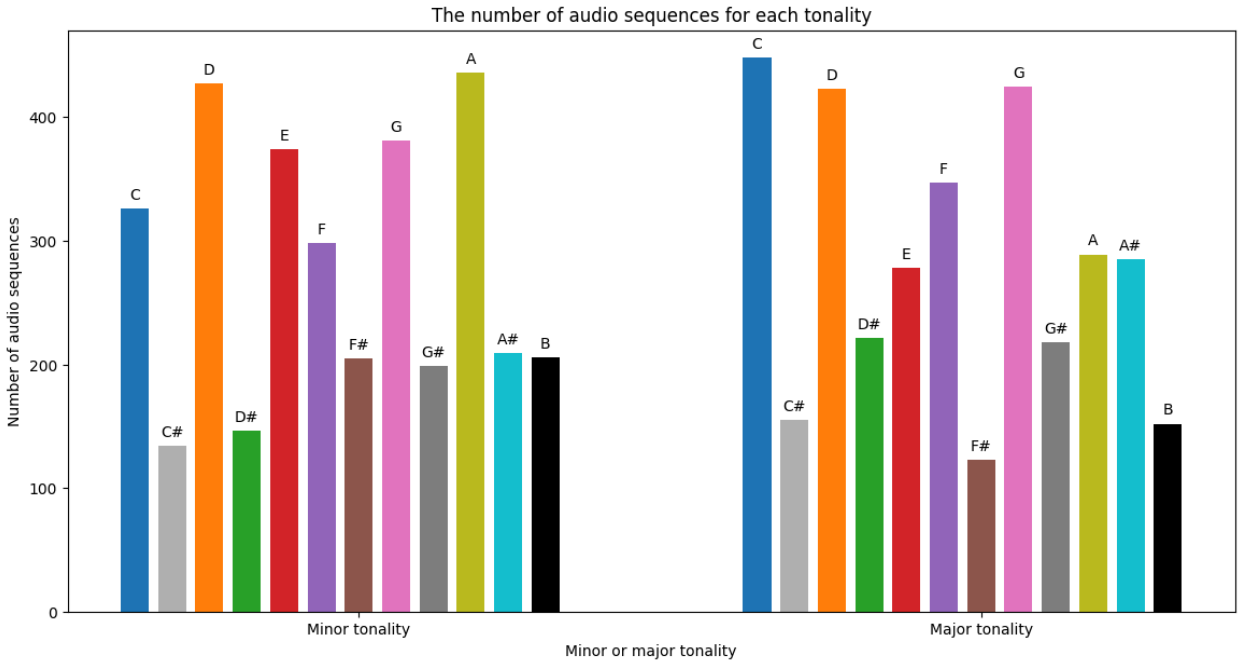


Figure: The counts of keys per major/minor tonality in the single-instrument dataset

It is important to distinguish the difference between the most frequent tone and the root of the key, because the two do not have to correspond. Again, to understand the terminology, a bit more theory. If we take a look at two pairs of two notes e.g. C and D, and E and F#, we can see that on the piano, those two notes are equally apart. A way to measure this relative distance between two notes is by using intervals. Below, we can see the plot that compares the distribution of intervals between the root of the key and the most frequent tone. We can see that the tonic interval and the perfect 5th interval stand out the most. These intervals have a very full and rich sound, without giving any hints about the key being a major or minor key.

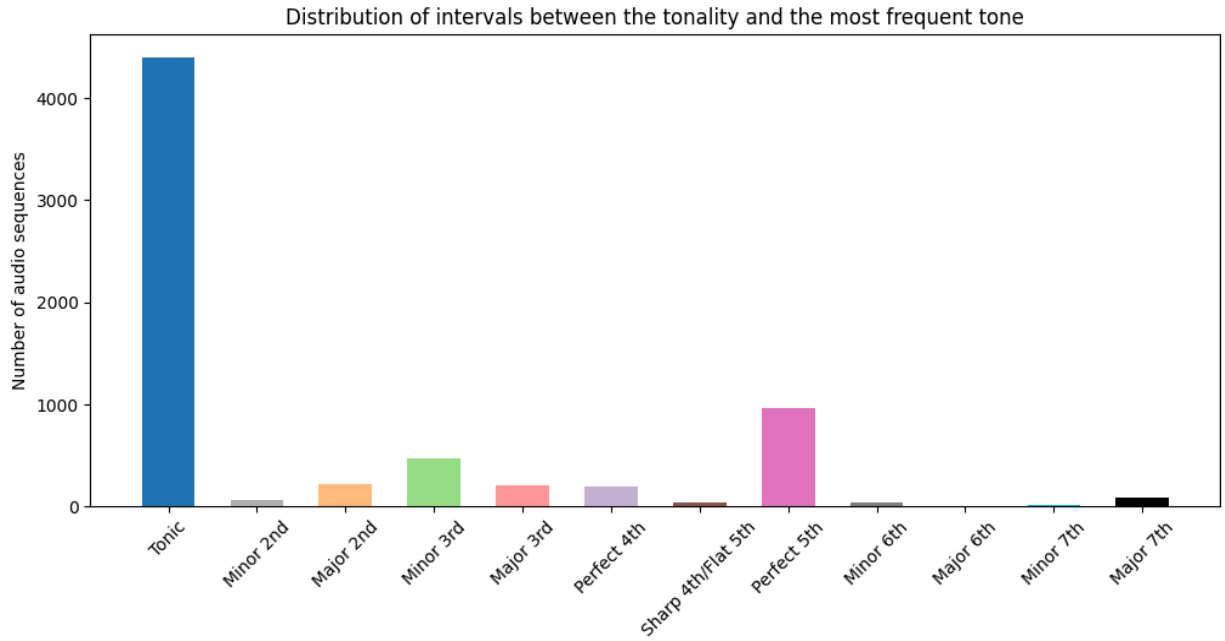


Figure: The distribution of intervals between the totality and the most frequent tone

Finally, we can measure the percussive elements in music as well. By computing a spectral flux onset strength envelope we can determine the audio's onset strength. The onset strength can then be used to estimate the tempo by calculating the onset correlation. We can see the tempo distribution for the train and the validation split below.

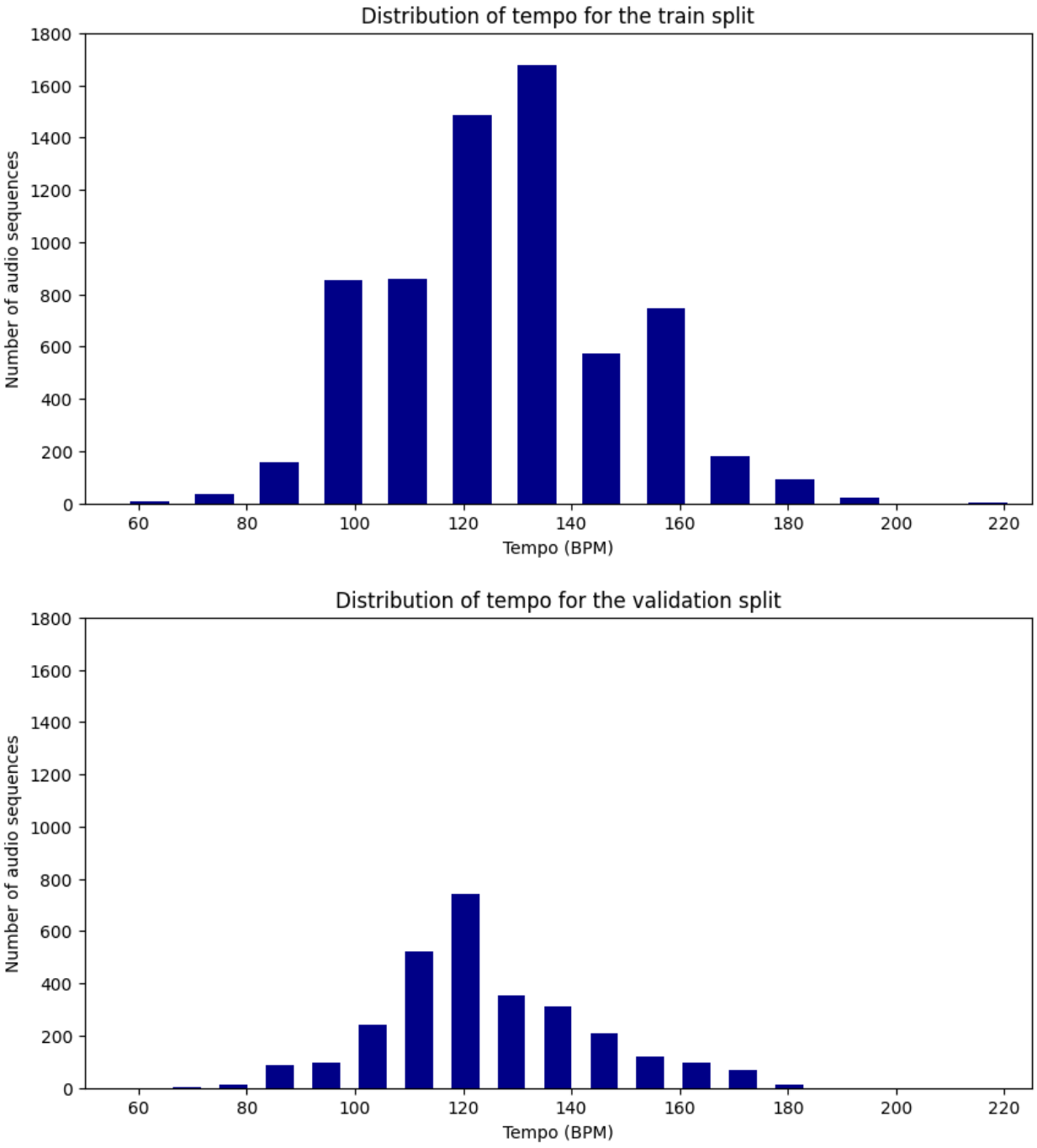


Figure: Tempo distributions for the IRMAS single-instrument (above) and IRMAS multi-instrument (below) datasets

The distributions are very similar. The train distribution isn't skewed, while the validation split distribution is slightly positively skewed. The mean and standard deviation of the train distribution are 126.90 +/- 21.468, while the mean and standard deviation of the validation split are 123.66 +/- 19.215.

3. Feature extraction & Preprocessing

This, and the following sections describe the transformations we performed on the data on its way to the model. This section focuses on how we handle the waveform and which spectral transformations we perform, while the next section is dedicated to augmentations only.

Stereo to mono

When the audio is loaded from .wav .ogg or an .mp3 file it typically comes in two channels, representing stereo sound. This sound is typically played from multiple sources and can give the sound an impression of depth. However, the model doesn't care about sound depth, those things only matter to us humans, furthermore the waveforms are mostly the same in both channels so we converted them to a single channel called mono.

Resampling the data

The raw data in an audio file is defined by the amplitude value of the sound, and the sample rate. With this information we can properly reconstruct the sound present in the file. The sampling rate defines the time scale of the audio file; it defines how many points are taken in a single second. It also defines the maximum frequency of a sound that can be saved in the audio file. Most of our training data is sampled at 44100 Hz, and initially we used that very sampling rate for training. Things changed when we started using models that were pre trained on audio, such as AST. A great majority of them are trained with audio that has a sampling rate of 16000 Hz. Although it initially seemed like a great reduction of information, we've found that the mel spectrograms (our core representations of input data) do not change noticeably after the said resampling.

Furthermore we've realized that the resampling process itself slowed down the training substantially, and since we intended to run a large number of experiments this problem needed to be dealt with. So we completely resampled the entire training data and saved it for future use. Every new piece of data we acquired we checked its sampling rate and resampled when needed to 16000 Hz. This in fact sped up the training procedure substantially, and while the exact number depended on the model at hand, the typical acceleration was up to a factor of 2.

Normalization

Audio tracks from various datasets and sources can vary according to their power output, figure below depicts this distribution. We do not want the model to get signals that vary greatly by amplitude so we decided to normalize each audio track. The audio normalization was done using librosas built in function normalize which is the equivalent to min-max normalization. While this does solve the problem of variable power in audio tracks it is sensitive to noisy spikes in the audio, a single point can squeeze the waveform too much. A better way of normalizing is normalization by volume, unfortunately we did not implement this feature in time.

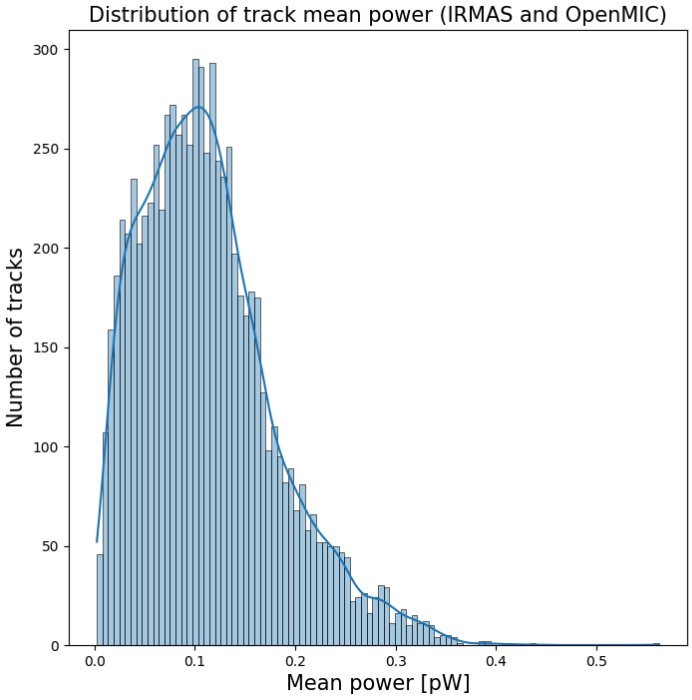


Figure: Distribution of audio tracks by mean power output.

Mel spectrogram

Based on the latest audio classifier papers Mel spectrogram proved to be a state-of-the-art feature so we decided to employ this valuable feature in our models. Mel spectrograms are constructed in a few steps. First, we split the waveform into several overlapping windows, then we compute the Fourier transform of the signal present in each of the windows. What we end up with is a regular spectrogram which gives us information about the spectral and temporal features of a given signal. If we further explored the obtained spectrogram, we would soon notice that its frequency, given in linear scale, is unnatural in the sense that the human ear doesn't perceive it in that way. Studies have shown that the sounds humans perceive as equally distant are not really equally distant in the frequency range. For example, an average person would say that the sounds with frequencies of 200 Hz and 300 Hz are perceptually more separated than the sounds with frequencies between 5200 Hz and 5300 Hz but, as we can clearly see, they are equally distant in the frequency range. Obviously, linear scale in frequency is not really appropriate to describe audio spectrograms so we have to devise a new scale that would be more in line with the perceptual scale of pitches judged by listeners. The new scale is logarithmic in nature and it is called the Mel scale. The reference point between this scale and normal frequency measurement is defined by assigning a perceptual pitch of 1000 mels to a 1000 Hz tone.

The idea of a Mel spectrogram is to bin frequencies in such a way that higher frequencies get wider bins and lower get bins that are more narrow, in this way we can model the logarithmic nature of human hearing. There is a special procedure to do this and it is a result of a broad research. As we have already mentioned, there is a formula that is logarithmic in nature that converts between mel frequencies and frequencies in Hertz. Using that formula, we construct the mel banks which are shown in the figure below.

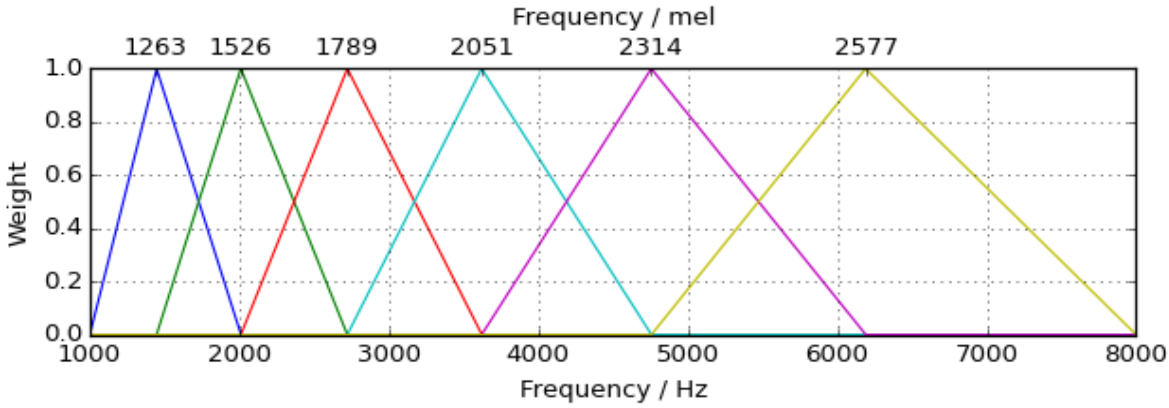


Figure: Mel spectrogram filter banks

We use these bins called the Mel filter banks as a filter on a signal. When we perform this filter, we obtain a spectrogram which is considerably more in line with human perception. That spectrogram is called a Mel spectrogram. Additionally, we convert the mel-binned spectrogram from a power representation to a deciBel representation for even more natural representation as the human perception of loudness of sound is not linear, so the dB scale is more suited for the exploration of sounds. In the figure below, the difference between the spectrogram with linear frequency scale and linear amplitude scale and the mel spectrogram with the mel frequency scale and the dB amplitude scale is shown for the same audio file. We can obviously, at least visually, obtain much more information from the second Spectrogram than the first one.

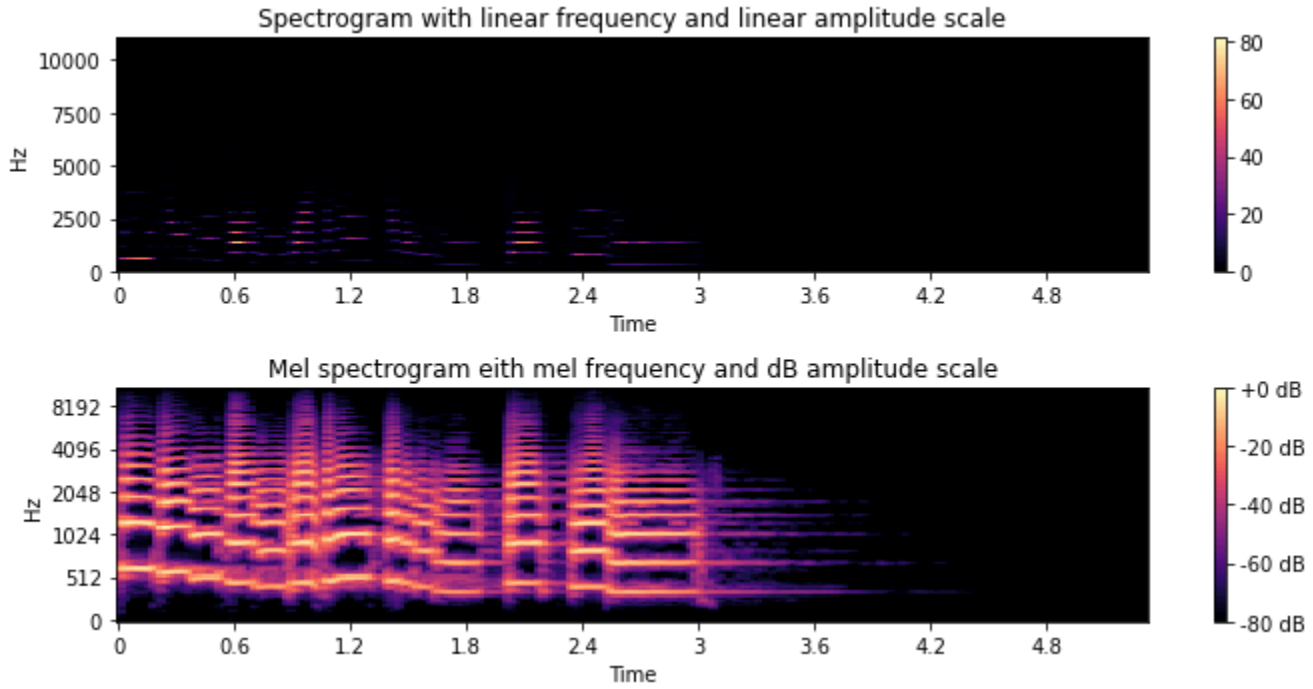


Figure: Comparison between power and dBell representations of the mel spectrograms. Above liner scale, below dBell scale.

Other features: chromagrams, wavelets and abominations

We explored several ways to incorporate spectral features into our model. We will explain each of them briefly, mention their pros and cons and what they might help the model achieve.

The most clear-cut feature is the raw waveform itself, and while they might seem appealing due to their conceptual simplicity, they require far more work than they're worth. For instance any convolutional filter over the waveform needs to cover a wide temporal window, so with a 16000 sampling rate one needs 3200 parameters per

channel for the filter to cover a 0.2 second interval, and that's only the first layer… Nevertheless we explored them as a raw feature for models and will report further in the following sections.

The other end of the spectrum (pun intended) is to use hand crafted spectral features such as chromagrams, spectral centroids, tempograms etc. These features are popular in approaches utilizing SVM models or statistical methods based on KNN-s. However, other areas of deep learning have shown that learned features perform better than hand crafted ones, so the sweet spot most likely lies somewhere in between the raw waveform, and the overly engineered features. We assumed that the best representation of the data is in the form of a mel spectrogram, and due to it being an image of sorts we could borrow a wide variety of tools developed in computer vision.

Other spectral features

Aside from using only Mel spectrograms we attempted to use different similar spectral features called the chromagram and the scalogram.

A chromagram is produced in a similar fashion to the Mel spectrogram albeit with different frequency banks (bins). The idea behind them is to bin frequencies according to their octave pitch, the concept of pitch is thoroughly described in the section *What about music*. The recipe for creating a chromagram is as follows:

1. Create a Fourier spectrogram of the waveform
2. Create a filter bank matrix (see figure #neki broj)
3. Perform matrix multiplication on spectrogram with the filter bank
4. Et Voila, enjoy your chromagram!

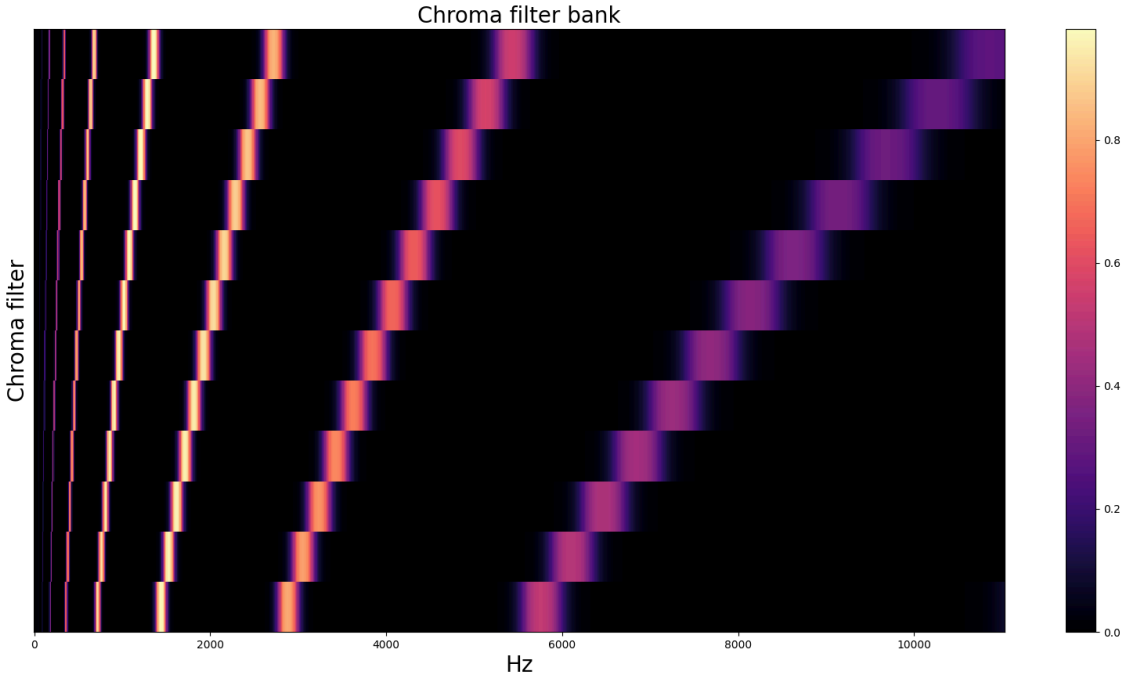


Figure: Chromagram filter banks.

It's worth mentioning that the filter banks attempt to bin the energy content of the spectrogram to separate pitch classes. Although this feature seems nice and intuitive it did not achieve better performance, compared to the Mel spectrogram, we believe that this is due to the large information loss after the binning is performed, and that the same instruments may cover the same pitch classes. We do recognise the importance of this feature in genre identification tasks.

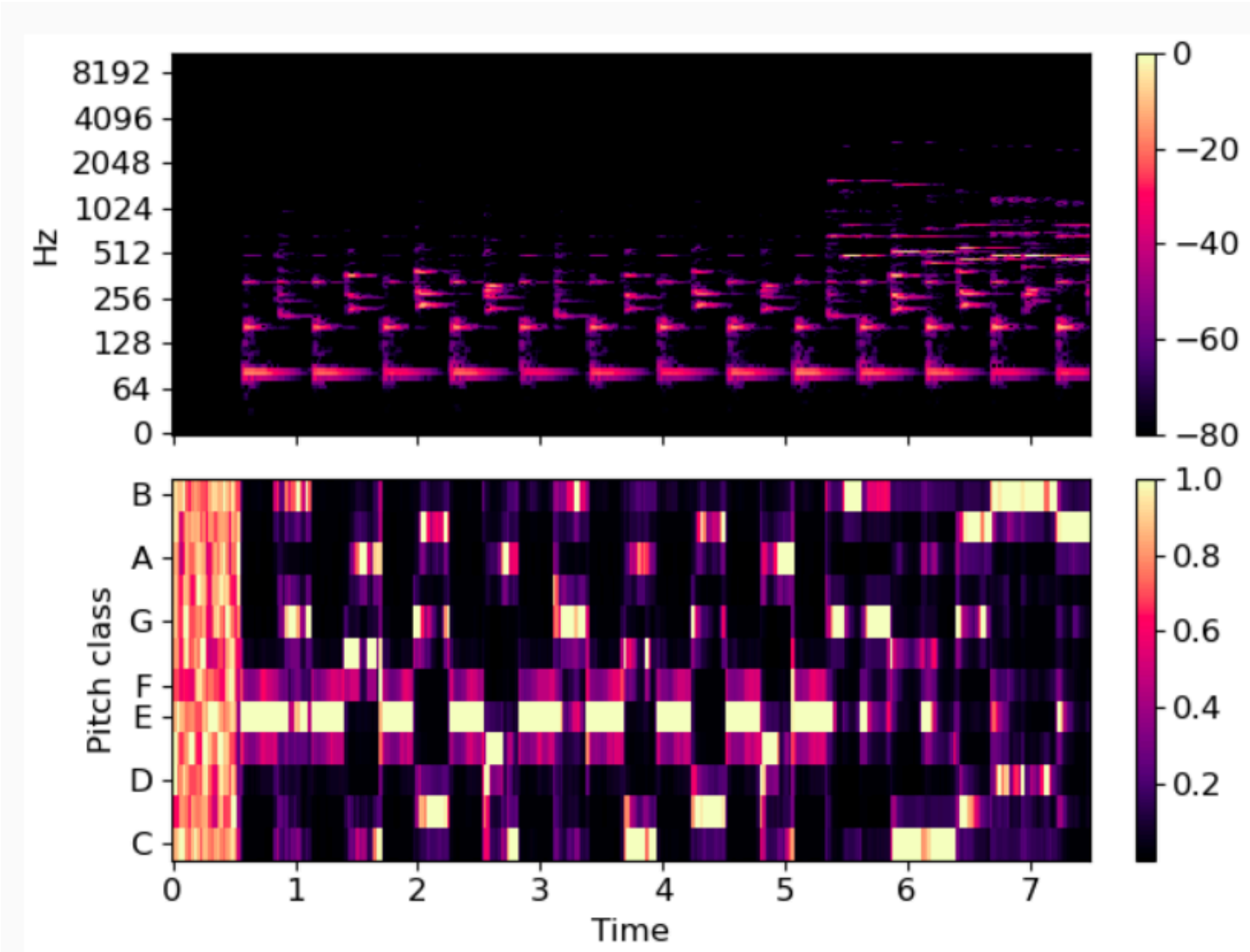


Figure : Comparison of a spectrogram and chromagram of Tchaikovsky's Nutcracker.

One more transformation was considered and that was the wavelet transformation that produces so-called scalograms. The main idea from wavelets is not to capture events that happen at different frequencies rather, to capture events that happen at different scales. We chose 128 wavelet filter kernels, gaussian modulated sine waves with different scales. These kernels are convolved over the waveform to acquire a scalogram. To this end we even implemented our own Pytorch module that performs this operation. The resulting scalogram differs from the one acquired by the continuous wavelet transform typically found in the (Pywavelet packages).

The major pro of scalograms and wavelet transforms is that they possess the optimal time-frequency resolution tradeoff i.e. at high frequencies (low scales) the frequency resolution is worse, and the time resolution is better, likewise the opposite holds. This sort of mimics the human hearing perception. The con is that the usage of scalograms as input features yielded bad results.

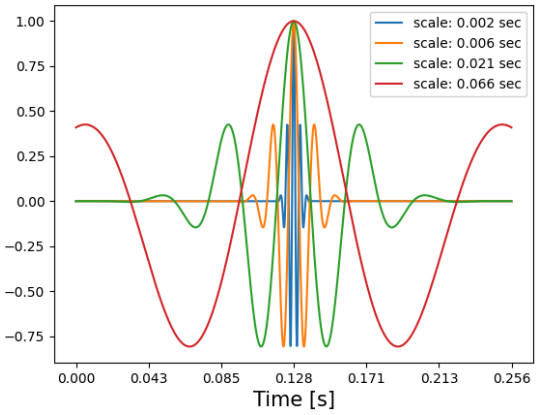


Figure: Wavelet filters.

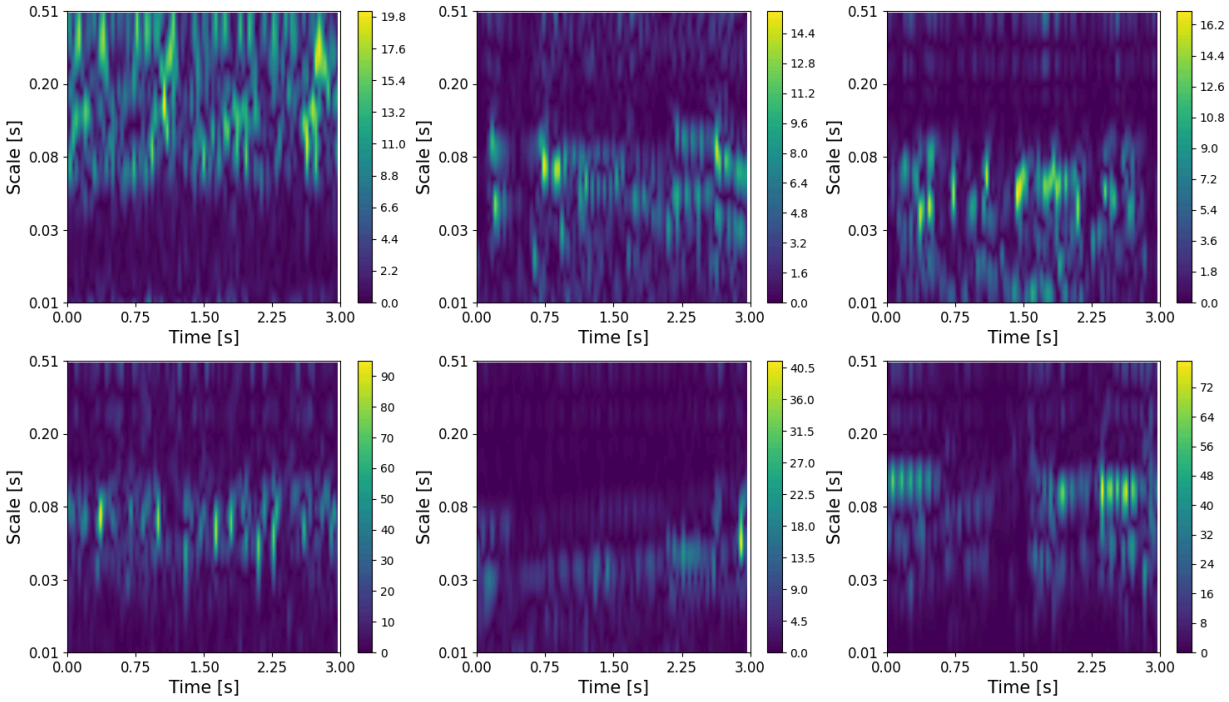


Figure : Scalograms for various instruments.

Three channel spectrogram

A last resort approach in using scalograms and chromagrams was to perhaps include them as different RBG channels in the input of a convolutional network. This only slowed down the training procedure and gave admittedly poor results. What else could we expect from this god awful image? We suspect that the reason lies in the fact that a Mel spectrogram, a chromagram and scalogram all represent different quantities and therefore should not be just added with a convolution kernel. A more sensible approach would be to construct three separate convolutions that extract vector representations of Mel spectrogram, chromagram and scalogram, and then concatenate or add the resulting three vectors; unfortunately we did not explore this approach.

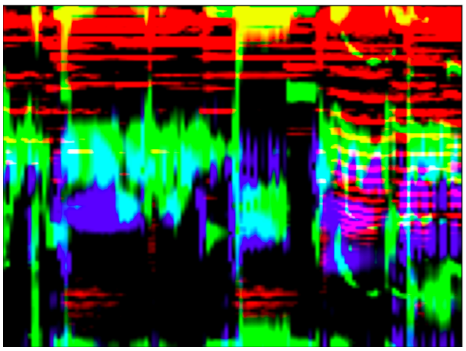


Figure: Mel spectrogram, chromagram and scalogram as RGB channels of an image.

Chunking

A crucial piece of data preprocessing is the so-called chunking. We either chunk the waveform or the spectrogram depending on which one the model takes as input. The chunking is only done along the time dimension, and the whole reason for performing this operation is that we want the model to train and perform inference on data with fixed length. This happens to be especially useful during inference since the main goal of the model is to recognize the instruments present in an audio. If we have a song that is usually saxophone with a voice interjecting in short intervals, a model working on the entire audio at once might not notice the voice in the short intervals.
For a prime example of a song that has this property please check out this amazing performance on *America's got talent*: https://www.youtube.com/watch?v=mVCC0MBT6bE. Instead we chunk the audio file (or melspectrogram) into smaller subsections that are all 3 seconds long, and allow the model to find the instruments in each chunk. This adds flexibility to the inference procedure, i.e. it can be done either on patches of 3 seconds or on entire files. In the latter case we find the prediction for each chunk and return the union of the predictions. For instance if the audio is 9 seconds long and the model finds a piano in the first patch, a violin and cello in the second and a voice and piano in the last chunk, the prediction is: piano, violin, voice and cello.

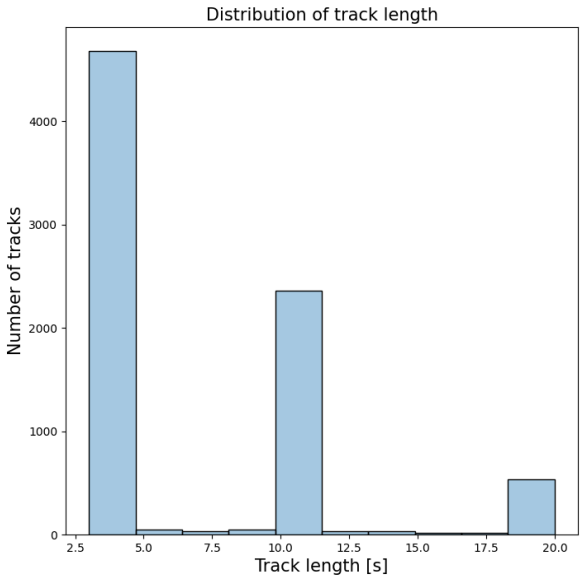


Figure: Distribution of track length in the Boosted IRMAS + OpenMIC set. The peak at 10 seconds corresponds to the OpenMIC tracks.

This may become a hindrance during training when one is dealing with weakly labeled data such as IRMAS (there is no information when the instrument is actually playing), if the whole 20 second audio is labeled with a voice and guitar it is not necessary that both instruments are present in every instant of the song. However, these cases are few and are overshadowed by the correctly labeled chunks

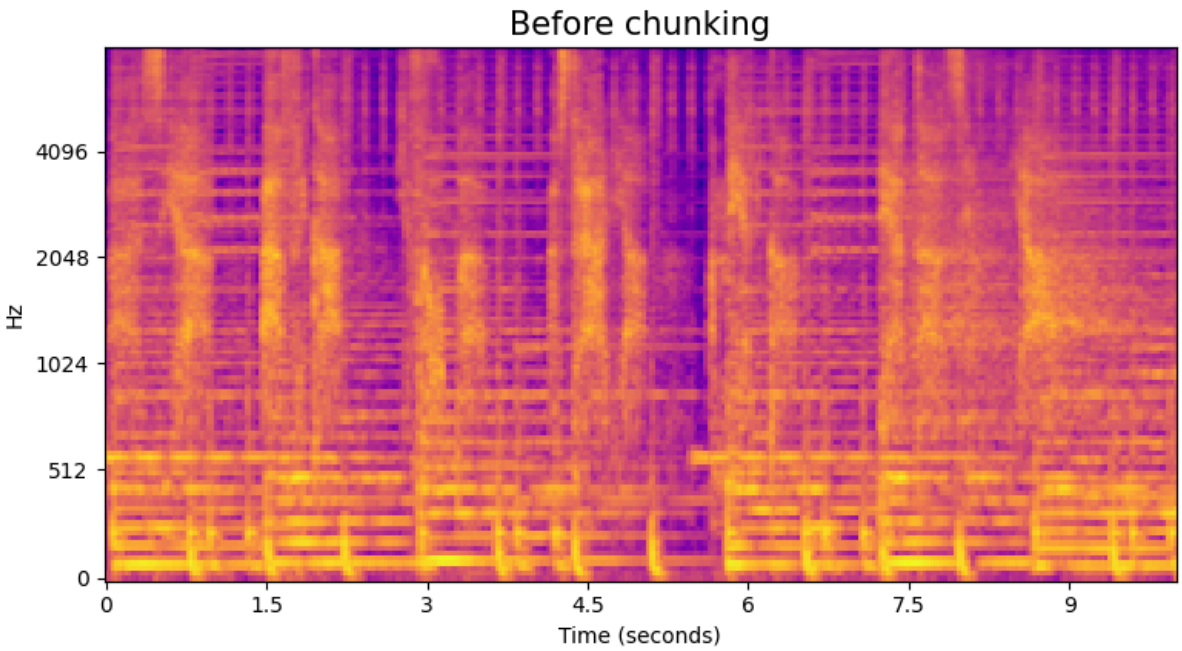


Figure: A 10 second long spectrogram before chunking.

We divide the mel spectrogram's temporal dimension into chunks that are 3 sec long, the frequential dimension is left unchanged. For instance if we have a spectrogram that is 10 sec long, then we will be left with 4 spectrograms that are 3 sec long. The additional mel spectrograms are stored in the batch dimension, i.e. chunking increases the batch size implicitly. To keep track of which chunk came from which file, by adding a group index to each mel spectrogram, the group indices are the same for each mel spectrogram that originated from the same file.

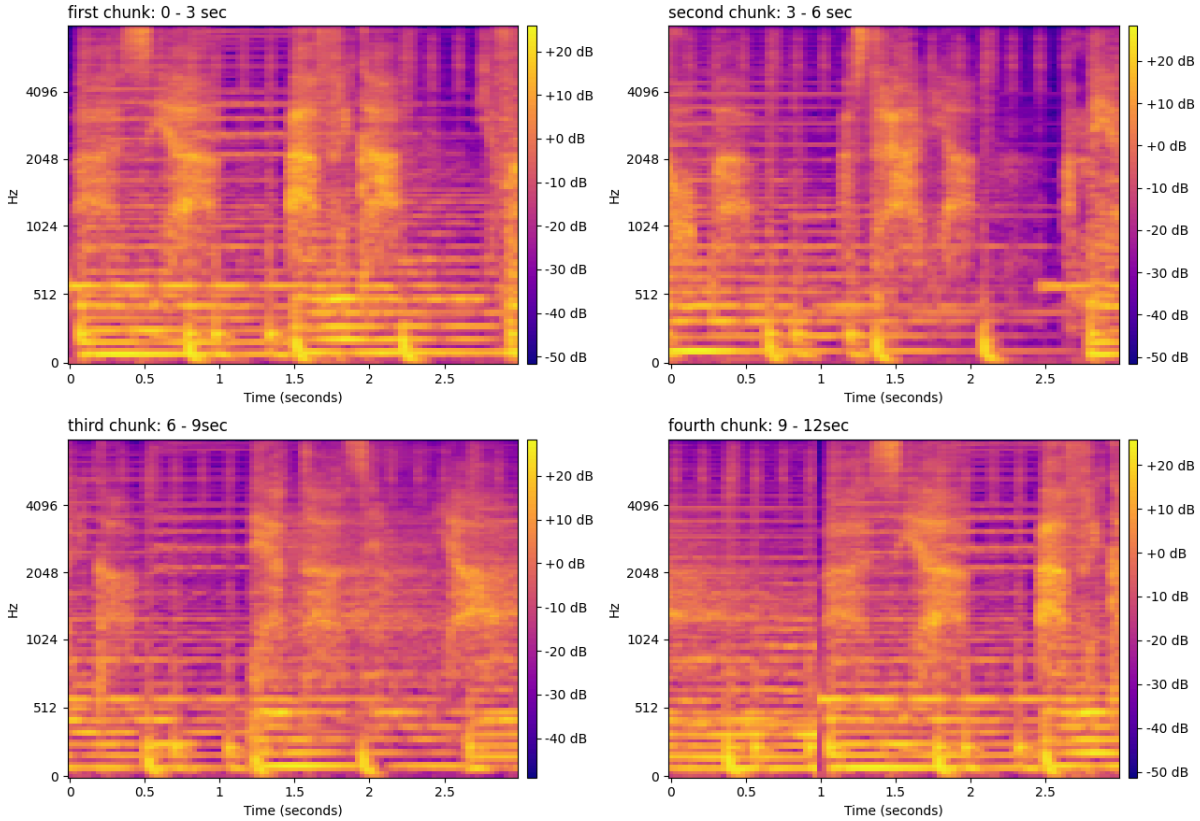


Figure: Four resulting mel spectrogram chunks each 3 second long. The last chunk is only 1 second long without extension, thus it is extended to 3 second length by repeating the entire original mel spectrogram. This can be clearly seen as a sharp vertical line in the last chunk.

An attentive reader will notice that if the temporal length of the audio is not divisible by 3 seconds then we are left with a fractional number of mel spectrograms. To be more exact there is one spectrogram that is shorter than 3 seconds, so what to do with it?

The simplest solution is zero padding, however this means that we will often save a large number of zeros, as well as pass a large number of meaningless zeros to the model. Instead we decided to repeat the audio. In the figures above we can see how the chunking is performed for an audio that is 10 seconds long. There are no problems with the first three chunks, however with the last chunk we are left with only 1 second of mel spectrogram content, so we repeat the first 2 seconds of the first chunk on top of it, as if the audio restarted itself.

4. Augmentations

Adding waveforms - Sum with N

A powerful, yet cheap augmentation is waveform addition which we call *sum with N.* The idea is to take multiple waveforms containing different instruments and overlay them together. We attempted several different types of waveform addition and here we will explain both the procedure and the effects of these augmentations on the data distribution as well as model performances.

The simplest way of adding is called sum with N, and let's say N=2 for simplicity, here we first sample a waveform and check which instrument is present, say a piano. Next we choose N distinct instruments that are different from the initial instrument, in our example we sample instruments that are different from a piano, and we sample 2 of them, let's say a violin and a cello. Now that we have decided which instruments to add we randomly sample an audio file for each instrument. One last step before adding the instruments, we have to normalize each waveform before adding them to the resulting waveform which is then divided by the total number of instruments so as not to create overly loud instances. The labels are binary vectors and to get the resulting label after addition we perform the logical OR operation between them.

The benefits of this augmentation are the following:

- It's a effective way to get properly labeled multi instrument data
- It makes the model more robust to noise
- It fixes the label distribution in the dataset
- It can help the model unlearn certain correlations between instruments

We believe a comment is necessary for each of the points, although it is a way to get multi-labeled data it is not clear that it will help the model find background instruments which are by default less loud and often hard even for humans to distinguish.

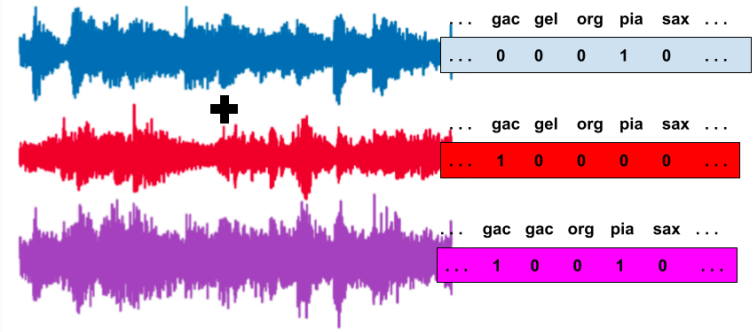


Figure: Example of waveform adding. The waveforms are added and meaned, while the labels are joined with the logical or operation.

There's an interesting interplay between adding instruments and resilience to noise. Adding waveforms without careful mixing will inadvertently by training the model in this regime it will learn to be more robust to various noise, however adding too many instruments makes the model (and humans for that matter) incapable of distinguishing the instruments correctly.

Another effect this way of adding has is it levels the instrument distribution more towards the uniform distribution, this can be seen on figure below. Probabilities of instruments being drawn are shown in histograms after summing with 0, 1, 2, 3, 4, and 5. A steady progression towards the uniform is noticeable, however as we've already mentioned adding too many results in a cacophony of noise. The sweet spot seems to be around sum with 2 and sum with 3.

Lastly, it is well known that some instruments are more likely to be played together than others, for instance a violin is likely to be played alongside a cello (see figure [ova s korelacijama instrumenata]), while an electric guitar and a clarinet are an unlikely combo. Nevertheless we do not want to allow the model to learn these sorts of correlations present in the training data, the reason is that the fraction of the dataset containing multiple instruments is small and that these correlations might be spurious. By randomly adding instruments these correlations are reduced.

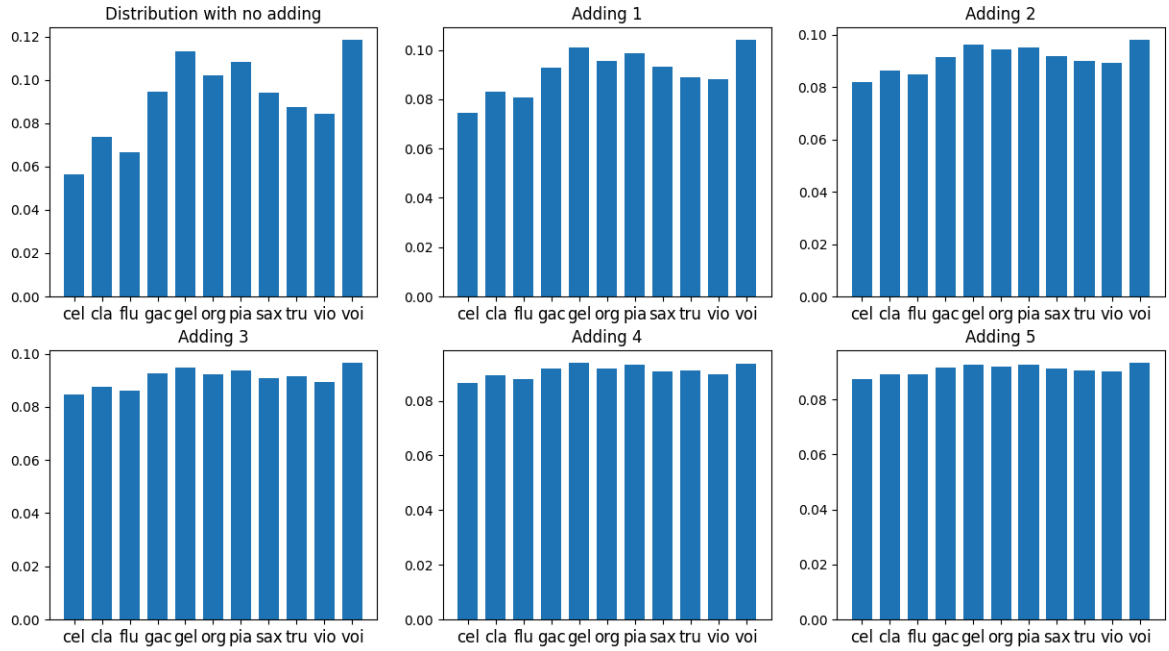


Figure : IRMAS single instrument label distribution change for various values of N in sum with N.

Another way of adding instruments was also explored and this was called exponential adding. Early on we noticed that the number of tracks containing large amounts of instruments was low. It was highest around 1 or 2 instruments per track and then it dropped exponentially. On the figure below we can see a histogram proving this assumption true,the bins drop linearly in a logarithmic scale, which is the signature behavior of the exponential. In order to make the added waveforms more natural we decided to randomly sample a number between 0 and 5 from an exponential distribution. This number represented the number of different instruments to add. However, this approach was not deployed since the uniform sum with N worked better. Furthermore the effect of making the label distribution more uniform is much lower when compared to sum with N.

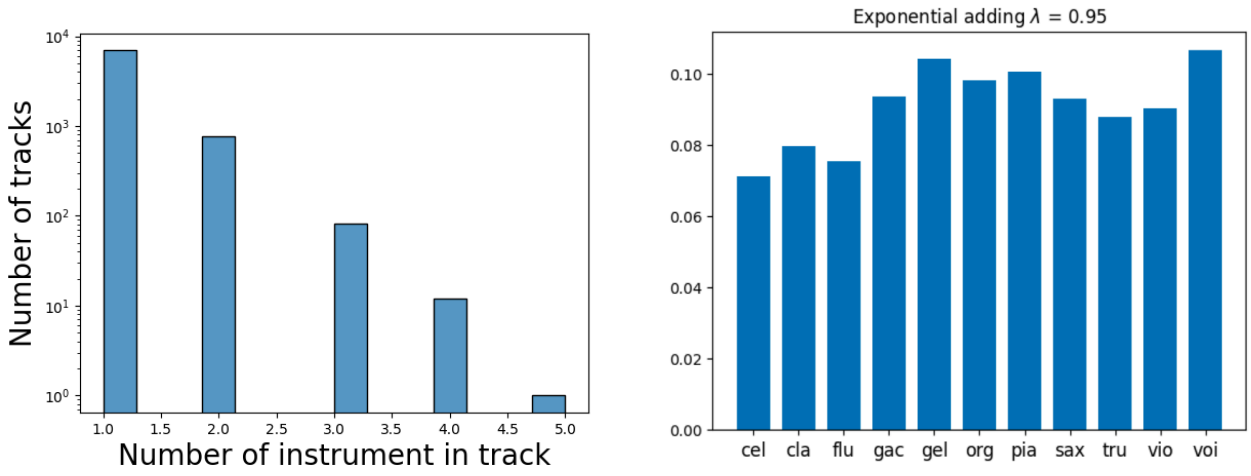


Figure: The left figure shows a histogram of tracks binned by their number of instruments played, the y-axis is in logarithmic scale, so we can clearly see an exponential drop in the number of instruments present in an audio track. On the right the figure represents the probability of sampling an instrument label after exponential adding is applied, the distribution is pushed slightly towards the uniform.

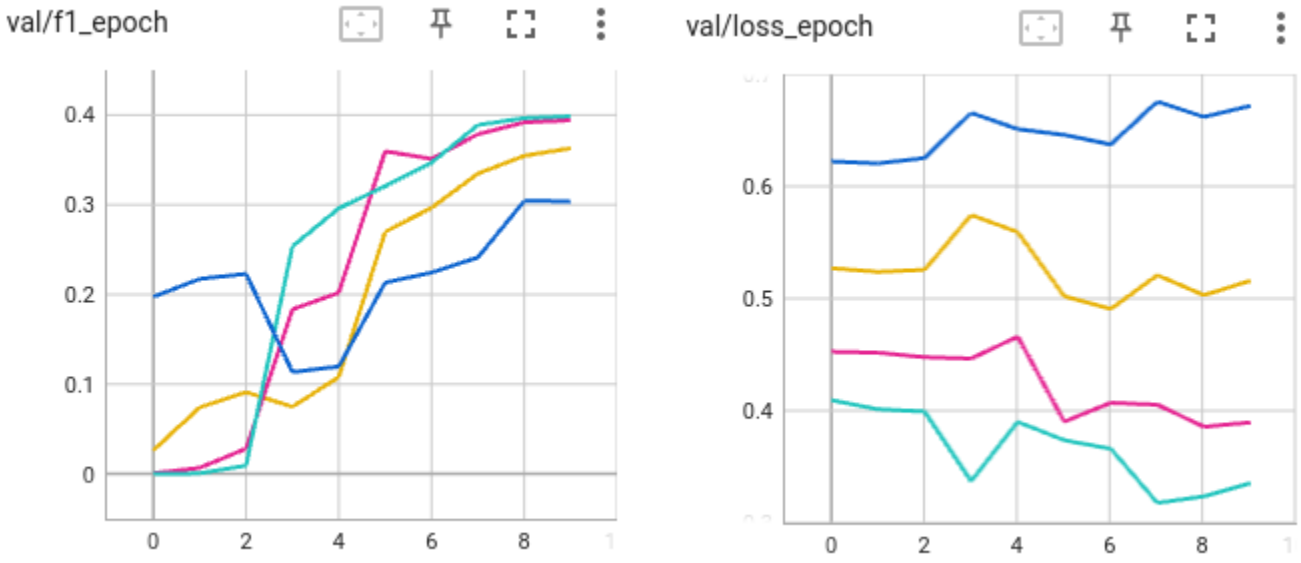


Figure: ConvNEXT training evolution for different waveform adding regimes. We can see how sum with 2 and 3 perform best while adding a third or fourth instrument into the *picture* degrades the performance. This is the interplay between augmentation and increases in noise we mentioned.

Concat N samples

Certain models are trained on audio longer than 3 seconds and to use them properly they need to be given spectrograms that are 10 seconds long, one such model is the all important AST. The procedure here is similar to the one present in sum with N. We take several audio tracks and concatenate them along the temporal dimension, as for the labels we again perform, and logical OR operation to obtain the joint labels. While this may seem as a necessity it actually serves as an augmentation as well. It produces new multilabel data, which is in short supply by itself.

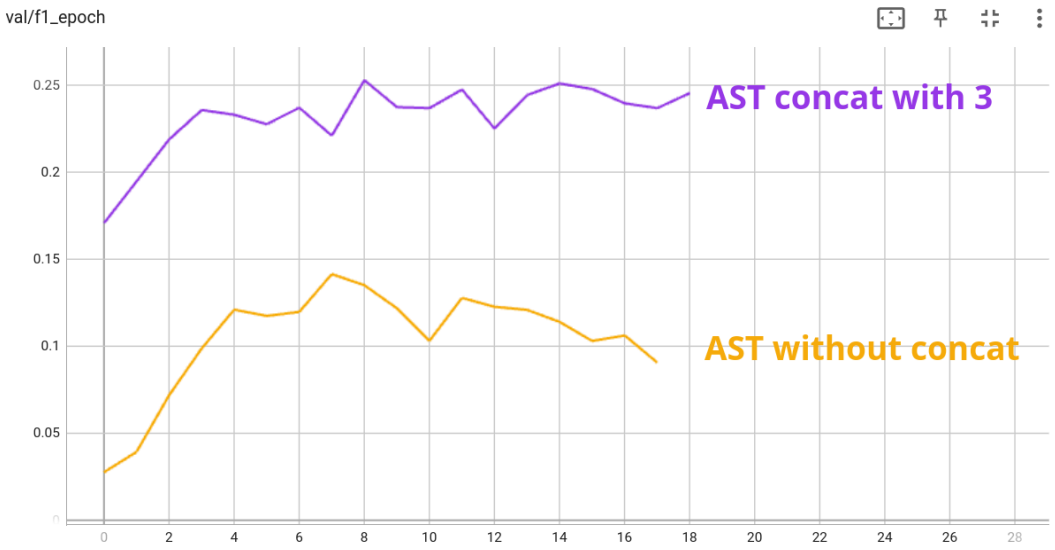


Figure: AST performance with and without concat N samples.

Waveform augmentations

We also performed several standard audio augmentations, we will list them here, explain how they work and comment on their effects.

Background noise

If this model is truly going to be an general audio extractor it has to handle audios that were recorded in all sorts of environments, this sort of robustness can be obtained with the background noise augmentations. We believe this augmentation offers more to the model than a simple gaussian nose does, the latter only emulates poor sound quality, which we have plenty of already present in the dataset. We used sounds generated from the *ESC-50: Dataset for Environmental Sound Classification* (https://github.com/karolpiczak/ESC-50). The augmentation dataset contains 2000 audio files of various environmental noise ranging from rain and wind to laughter and car horns.

Time stretch

Time stretching is a waveform augmentation that takes the sampling rate and modifies it by a factor between 0.8 and 1.25, by making sure that the pitch and tempo stay the same. The whole augmentation requires multiple steps that ensure that the pitch and tempo stay preserved. They are a bit complicated and for further details we refer to: https://en.wikipedia.org/wiki/Audio_time_stretching_and_pitch_scaling#Frame-based_approach

Time inversion

Although the sound is inherently serial, there are certainly aspects of the sound musical instruments produce that are invariant to time inversions. Play a single note and you will find it difficult to pre-determine whether the note was played in reverse or not. To this end we perform time inversions on the waveform with a certain probability, with hopes that the model will learn these invertible features of sound.

Time masking

TIme masking hides a certain random sub interval of the entire waveform that it is given. Musicians often make abrupt pauses for dramatic effect while performing and this augmentation simulates abrupt pauses and starts off the track.

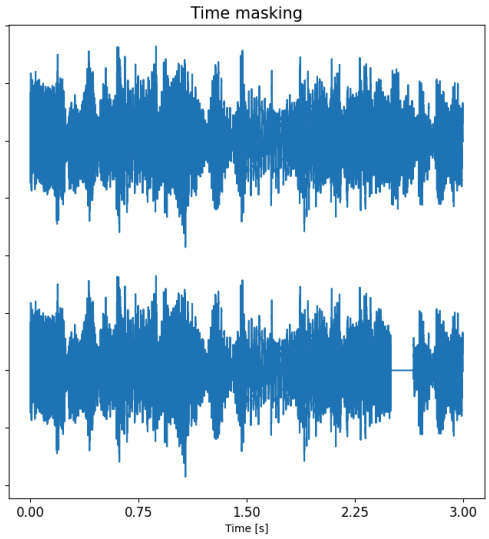


Figure: The effect of time masking on the waveform.

Colored noise

The concept of noise color refers to its power along the frequency spectrum, for instance brown noise influences lower frequencies far more than the pink noise does, on the other hand the pink noise influences (or is more present) at higher frequencies. The important note here is that differently colored noises sound different to humans and affect the different parts of the spectrum. This can make certain timbers more noisy than the others.

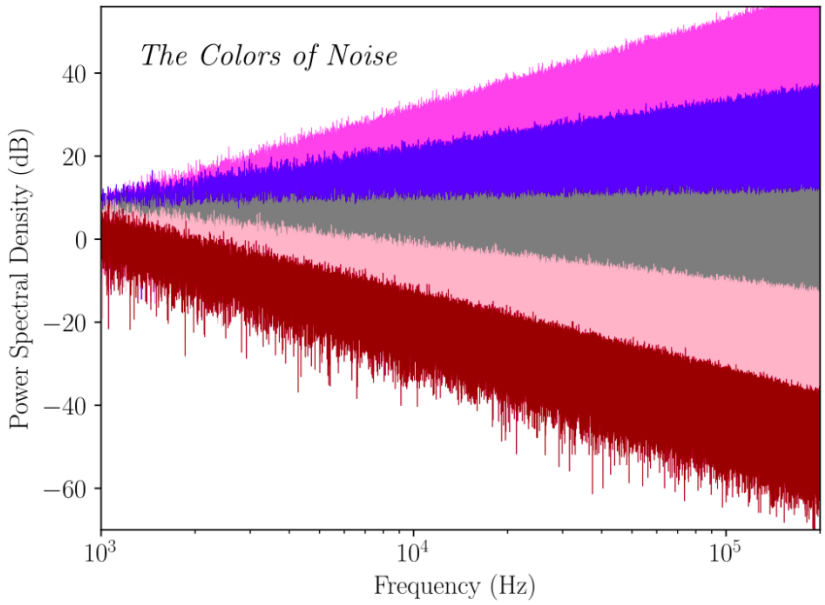


Figure : Colored noise differs in its power throughout the spectrum. The pink noise mostly affects the higher frequencies, while the brown noise is focused more on the low end of the frequency spectrum.

Spectrogram augmentations

Aside from augmenting the waveform itself we can perform augmentations on the spectrogram, which directly impacts the frequency content of the sound. The reason spectrogram augmentations make sense is that they distort the sound but the instruments are still recognizable, providing a way to add a different kind of regularization. We designed three spectrogram augmentations: frequency mask, random erasing and random pixel erasing.

The frequency mask augmentation is an augmentation that takes an arbitrary spectrogram and covers a certain range of frequency bins. Since the spectrogram's vertical axis represents the frequency bins, the resulting spectrogram will have a horizontal blacked out line of a certain width. The width of the horizontal line is determined by a hyperparameter which we set to 30. This setup means that our audio sequences will be augmented so that a certain number of mel frequency bins between 0 and 30 will be excluded. It's important to take into account the frequencies of musical notes when talking about this augmentation. If we consider the note A_4 at frequency 440 Hz, the A_5 (the A in the next octave) will have a frequency of 880 Hz. We can notice that the frequency in Hz doubles when considering two notes in consequent octaves. This also introduces the discrepancy between the relative distance of musical notes. The difference between the frequency of two notes in one octave versus the next also doubles. The problem with a spectrogram is that it bins the frequencies in a linear fashion. This means masking 30 bins in the higher frequencies and masking 30 bins in the lower frequencies would cover the same amount of frequencies, but not the same amount of notes. Mel spectrogram avoids the discrepancy problem by working on mel bins, which cover a larger amount of frequencies when working with higher frequencies in comparison to covering a smaller amount of frequencies when working with lower frequencies.

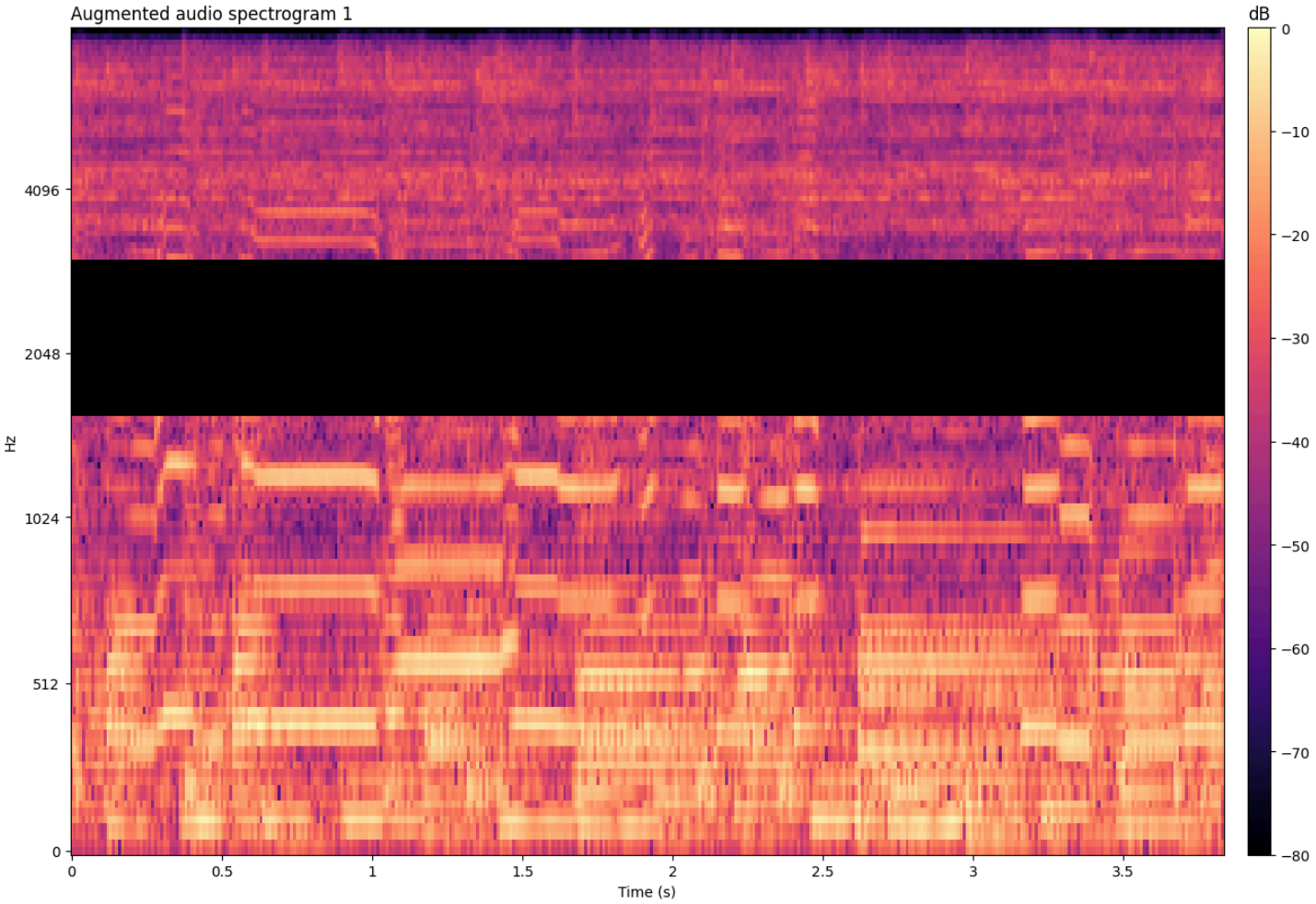


Figure: The image shows a spectrogram with the applied frequency mask augmentation. The figure above depicts applying the frequency masking on a spectrogram. The frequency masking augmentation imitates removing a smaller range of frequencies throughout the whole audio sequence. The way it affects the models is they're less prone to focusing on connecting frequencies with instruments.

Our next augmentation is the random erasing augmentation, which randomly selects a rectangle region in an image and erases its pixels. The augmentation is done only on a percentage of the images, where the scale and ratio of the erased area is also adjustable. The figure below shows the effect of the random erasing augmentation. The random erasing augmentation simulates distorting a wider range of frequencies in a short time period.

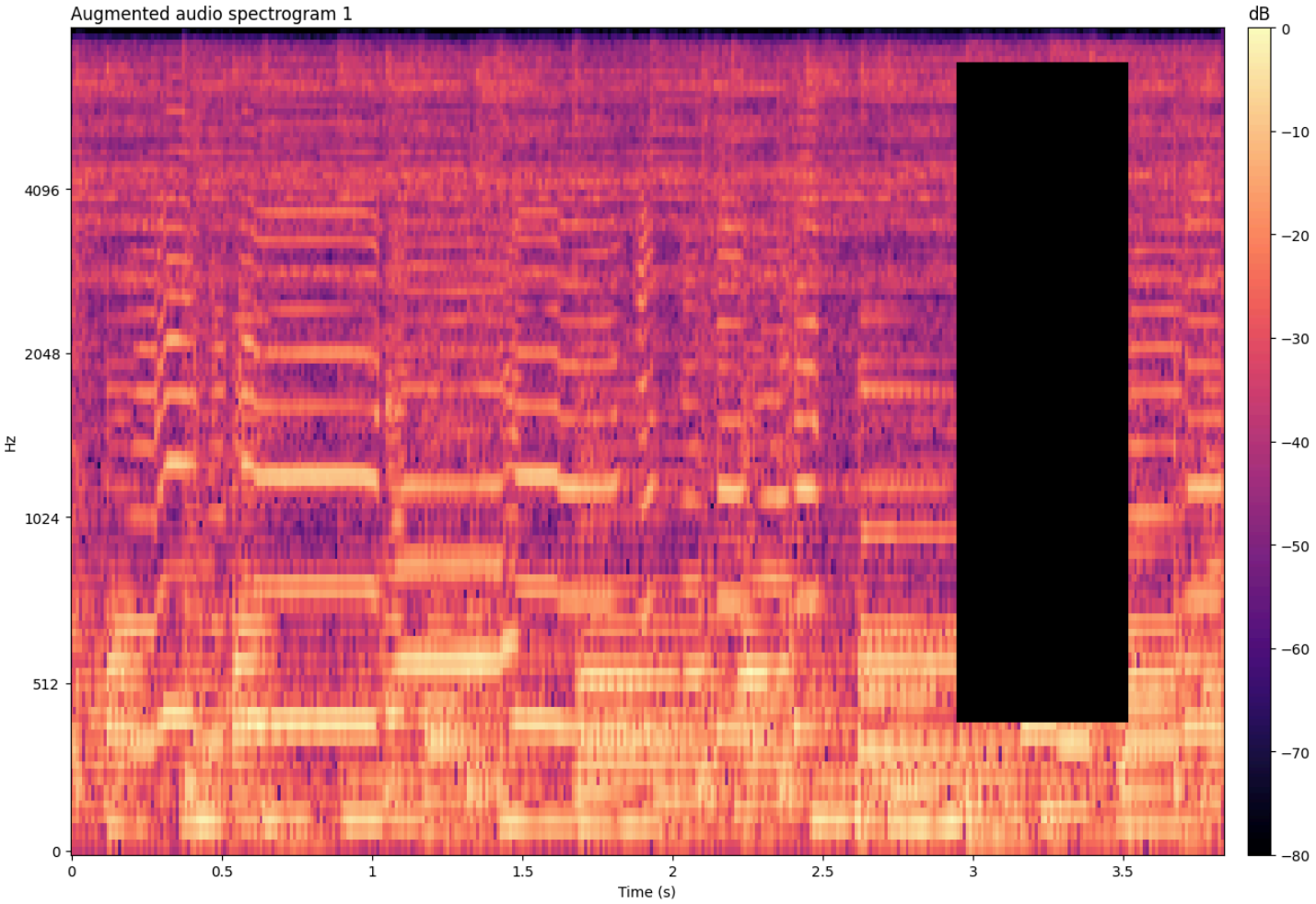


Figure: The image shows a spectrogram that uses the random erasing augmentation.

Our final spectrogram augmentation is the random pixel erasing augmentation. The figure below displays the effect of the augmentation. We can see that random pixels are chosen and then blacked out. This augmentation's effect on the audio isn't noticeable, but it affects the convolutional models by providing a regularization effect to make the model focus less on certain pixels.

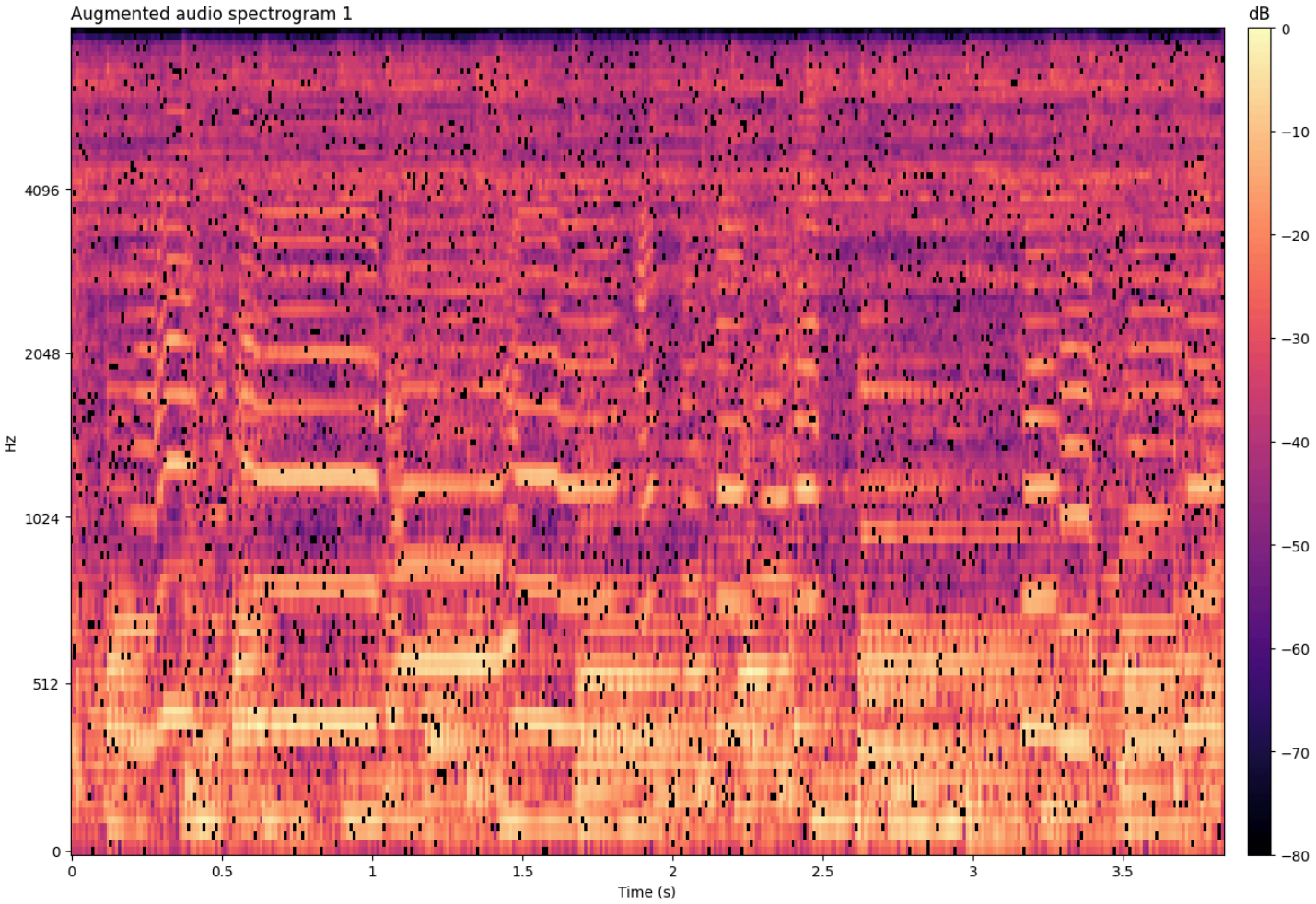


Figure: The image shows the effect of the random pixel erasing augmentation.

Finally, we provide a table for the waveform and spectrogram augmentations in order to compare them in a controlled setting. We use a ConvNeXt small model trained for 10 epochs with an augmented mel spectrogram. For comparison, the augmentations are applied individually. We excluded the sum with N and concatenate with N augmentations as their results were displayed previously.

Augmentation	Validation
No augmentation	0.346
Background noise	0.34
Time stretch	0.347
Time shift	**0.352**
Pitch	0.349
Color noise	0.343
Reversed audio	0.346
Time mask	0.351
Frequency mask	0.349
Random erasing	0.344
Random pixel erasing	0.350

Table: The results of training a ConvNeXt small model with an augmented mel spectrogram

We can see minor improvements for the time stretch, time shift, pitch, time mask, frequency mask and random pixel erasing augmentations, while other augmentations have a negative effect on the validation F1 score. Surprisingly, the time shift augmentation improves the score by the most.

5. Modeling

Loss Functions and Problem formulation

Before considering any model architecture we needed to clearly formulate what we want the model to accomplish. This is done by selecting an appropriate loss function. Instrument retrieval can be formulated as a multi label binary classification problem, or in plain english we assume each instance has a num_instruments (11 in our case) binary labels which encode the presence or absence of a particular instrument.

Binary Cross Entropy

The obvious way of telling a model what to do in this case would be using binary cross entropy as a loss function. Also instead of using softmax as a way to estimate class probabilities we use the sigmoid activation function on the model outputs. When properly trained, this approach can get the model to around 90% accuracy, except the model performs horribly and predicts that there are no instruments every time! This is due to the nature of the musical data, most songs contain only a small number of instruments, usually one or two but rarely more than six, i.e. the majority of labels for each song are set to 0. So any model trained in this fashion can achieve astounding accuracy simply by outputting zeros all the time. Two conclusions can be drawn from this: one, accuracy is a poor metric to evaluate the performance of the model in this task, and two we need to manually incorporate some mechanism to force the model to increase its recall. To solve the first we discarded the accuracy as a metric and instead focused on the average macro F1 score, as well as the per instrument F1 score.

Binary cross entropy with positive weights

The problem with the aforementioned approach is that any model will encounter a negative label (0) far more often than a positive (1), to mitigate this we need a way to increase the influence of positive labels on the loss function. Luckily for us the Pytorch implementation of binary cross entropy has an argument that does just that, the pos_weight (positive weight) argument.

$$l_n = -w_n \left[y_n \cdot \log \sigma(x_n) + (1 - y_n) \cdot \log(1 - \sigma(x_n)) \right],$$

The pos weight for the i-th class is calculated as number of negatives divided by the number of positives, here negatives refer to all the samples that have this label set to 0 i.e. the instrument in question is not present, while the positives refer to the number of examples that contain that instrument. For instance if we're dealing with a dataset with 200 tracks containing only guitars, 50 containing guitars and pianos, 100 containing only pianos, and 300 ones with violins, then the number of positives for guitars are 250 and the number of negatives are 400, yielding a positive weight of 400/250 = 1.6

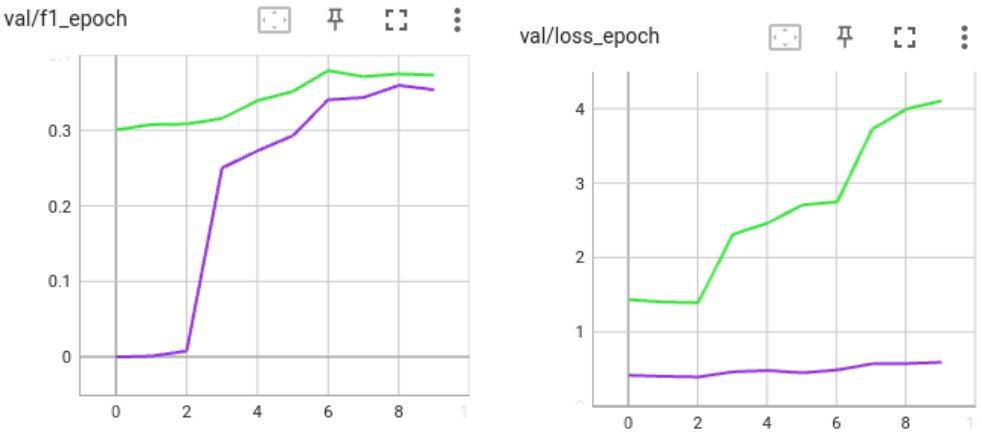


Figure : The left and right figure show ConvNEXT small training with no augmentations and same hyperparameters but the purple is without using positive weights, while the green is with the use of

positive weights. The effect on the training is noticeable: it slightly improves the F1 metric and more importantly it has a significant effect on the speed of training convergence.

Focal Loss

While including positive weights did indeed increase the F1 score it did so in by increasing recall and sacrificing precision, and while there typically exists a tradeoff between these two scores, we attempted to fix this as well. A trained model can correctly identify positive examples but it does this too often, a solution can be found in the form of the so-called focal loss.

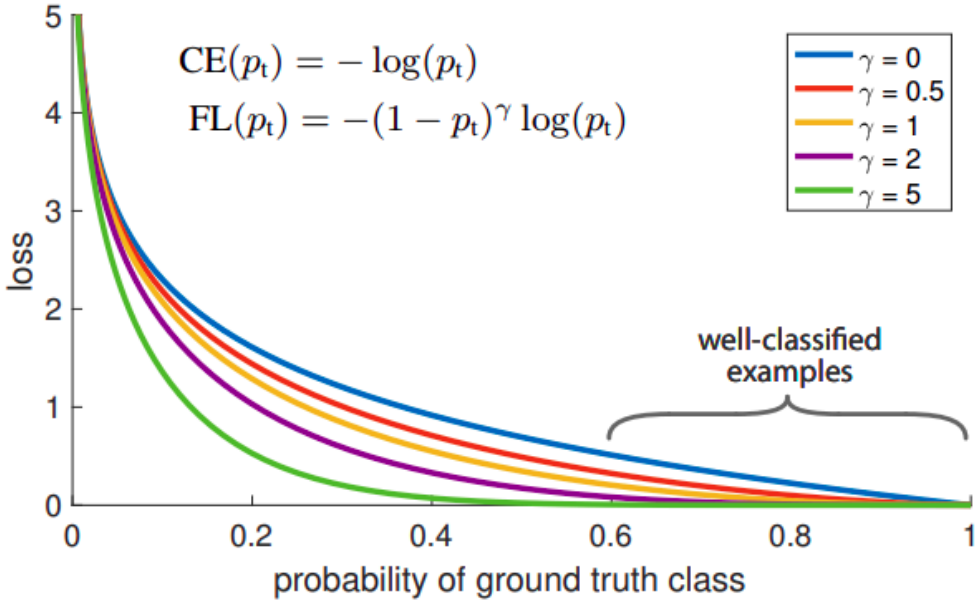


Figure : Borrowed from https://arxiv.org/abs/1708.02002. The figure shows how focal loss affects the shape of the loss function depending on the classification probability. High probability scores are less and less influential in the overall loss.

During the optimization the model can decrease the loss by increasing its confidence in examples that are already well classified, this can lead to overfitting if not to memorization of particular examples from the training set. To avoid this the focal loss places an additional factor that reduces the influence of well classified examples in the overall loss. This way if the model predicts a high probability of a positive example that example will no longer contribute significantly to the overall loss.

The figure below shows the validation loss and validation F1 metric across 10 epochs. All the hyperparameters were fixed except for the γ hyperparameter in the focal loss function. The setup we used included the small ConvNeXt model with a learning rate of 1e-4 and without any augmentations. The orange curve represents the model with the γ hyperparameter set to 0.5, which also achieves the highest F1 score being 0.3748. The model with γ set to 0.3 achieves the second highest F1 score, which is 0.3516. The model that uses the binary cross entropy loss achieves an F1 score of 0.3456, which is worse than the model with γ set to 0.1, with an F1 score of 0.3490. This insight suggests that the focal loss is better suited for solving the problem of multilabel instrument classification, and that increasing the value of the γ hyperparameter benefits the results. An important detail to mention is we unfreeze the backbone at the third epoch, which is why there is such a boost in the performance but also an increase in the loss.

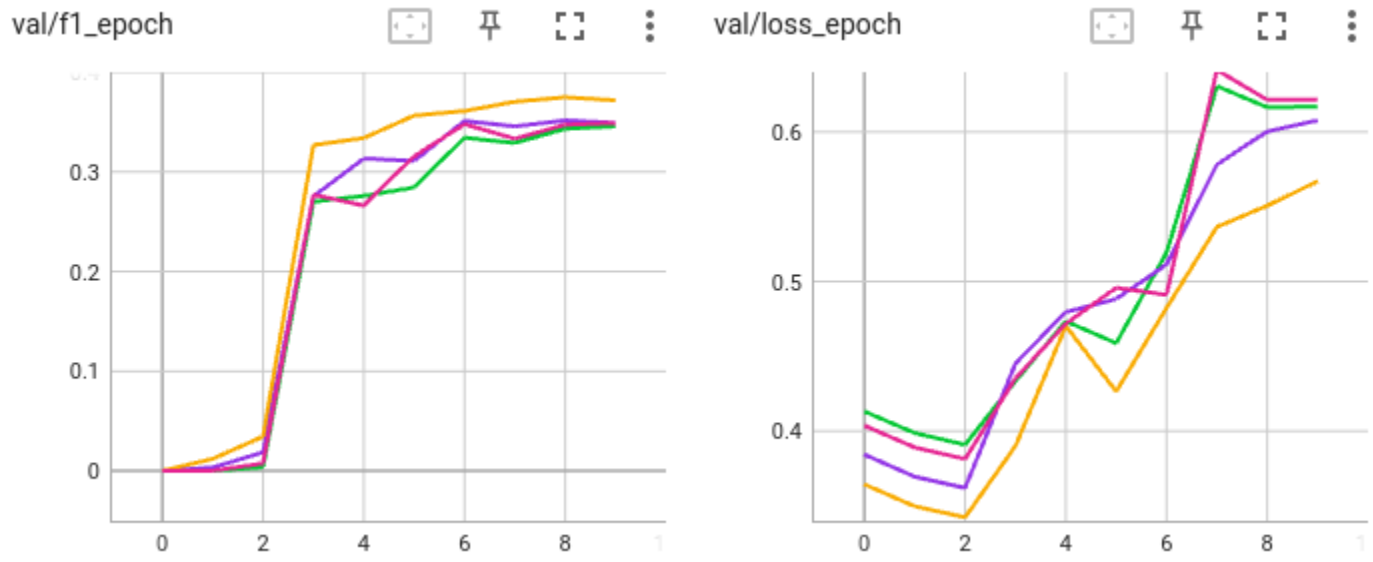


Figure: The validation loss and F1 score across 10 epochs of training using the ConvNeXt small backbone. The color of the curve represents the value of the γ hyperparameter used in the focal loss. Orange represents γ 0.5, purple 0.3, pink 0.1, and green represents the binary cross entropy loss, which is equivalent to setting γ to 0. All other hyperparameters are fixed.

Family Loss (auxiliary loss)

In order to further assist the model in correctly extracting instruments present in songs, we decided to add a sense of perceptual similarity of certain instruments to the loss function. Although there are many ways of doing this we found that the simplest way to group the instruments by instrument group. The groups are the following: brass instruments, guitars, percussions, woodwind instruments, and vocals. The idea behind this is that it is much easier to confuse a cello with a violin, than a cello with a trumpet. That is, instruments of the same family are more likely to be perceptually similar. Thus, alongside a binary cross entropy, or focal loss devoted to particular instruments we add to that an instrument family loss.

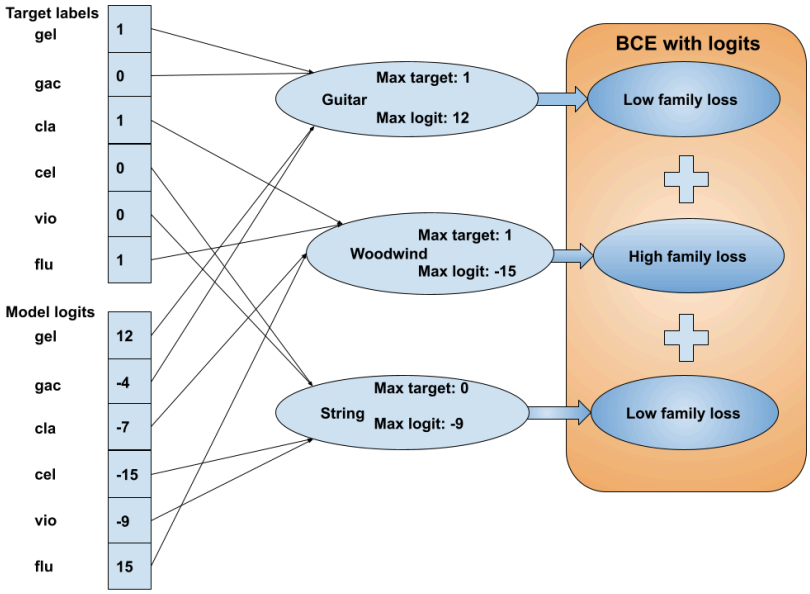


Figure: The figure shows how the instrument family loss is calculated, it indicates the presence or absence of an instrument family in the audio. We group the targets and logits according to family groups

and take the maximal logit/prediction from each group, these maximus are passed to the binary cross entropy.

A scheme of how the family loss is calculated is given in figure above, first all of the target labels and logits are grouped according to their respective groups, then we only take the maximum of these labels since it encodes the presence or absence of an instrument group, as well as the maximum logit. Those are then passed to another binary cross entropy (BCE) with logits loss, which is then added to the original classification loss multiplied by some factor. We can see on the figure above that the family loss did not in fact improve a lot on the model performance.

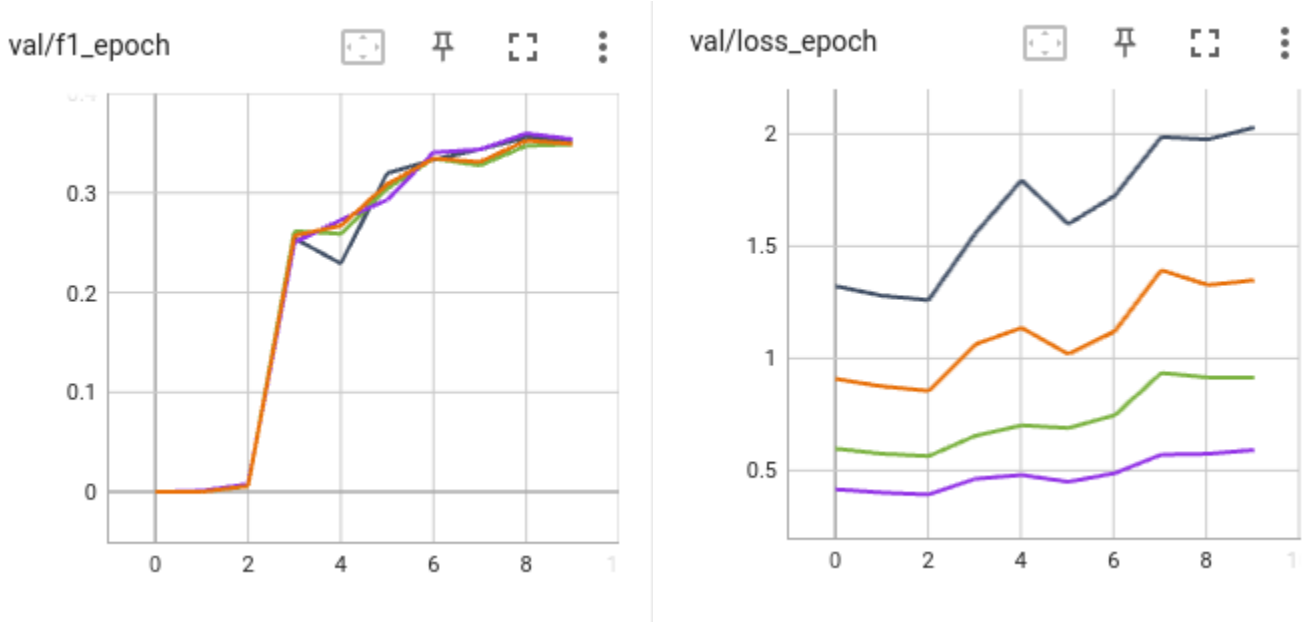


Figure : The three lines show the training progress of ConvNEXT with varying degrees of family importance. They are as follows: the purple line is the pure BCE loss, while the green, orange and gray have the family loss 0.3, 0.8 and 1.5 respectively. Although we found the idea to be sound, unfortunately it did not improve the models ability to classify the instruments significantly.

Classification heads

Most models we employed used a tunable backbone on top of which we stacked a classification head, here we briefly discuss the head configurations we experimented with. We've mostly used two types of classifier heads, which we've dubbed DeepHead and AttentionHead. A DeepHead is a configurable fully connected feedforward neural network, meaning one can specify the number dimensions each layer should have, for example if the module is given a list containing [128,64,11] it will produce three layers with 128, 64 and 11 neurons.

Most features that our backbones provide contain a temporal component to them. For instance if a convolutional backbone is applied to a mel spectrogram we may expect an output with a shape (N,C,F,T) where N is the batch size, C the number of channels, and F, T come from the original mel spectrogram height and width. Here the last component T holds the temporal information. We can always flatten the features, or perhaps perform a global pooling on F and T, however in doing so we throw away any temporal information the feature map might hold, which might be bad since sound is inherently serial in nature. A solution to this conundrum is the AttentionHead.

Our AttentionHead has almost the same configuration as DeepHead does except it has an additional attention layer that performs a learnable pooling across the temporal dimension, this allows the model to attend to the important time intervals for instrument classification.

Fluffy

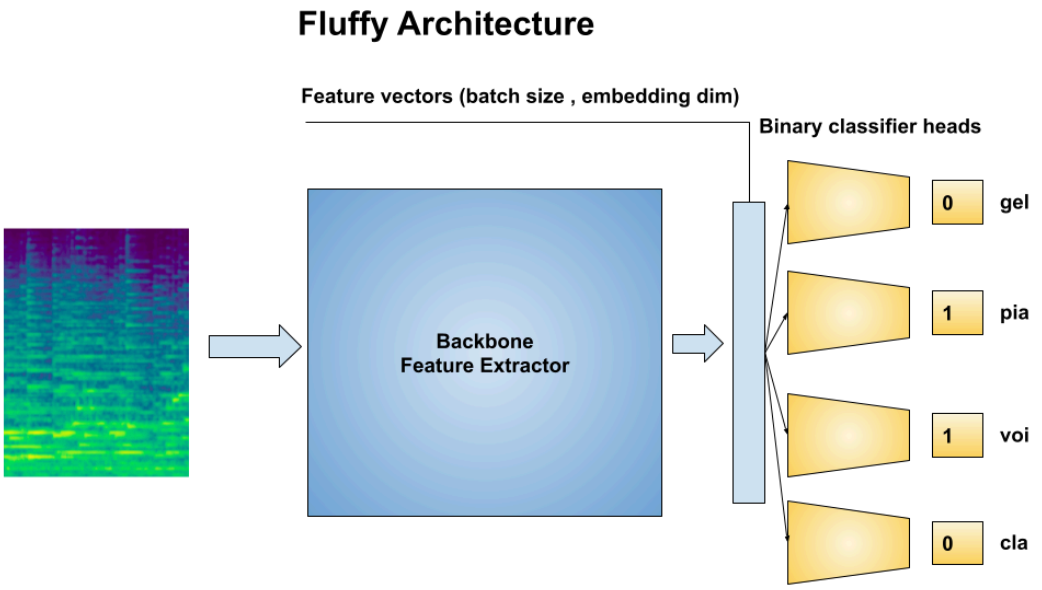


Figure: Fluffy model architecture, an arbitrary feature extracting backbone produces an embedded representation of the data. The embeddings are fed to the multiple instrument classification heads, each searching for a single instrument.

The usual strategy when using feature extracting backbones is to add a linear layer that serves as a classifier head. One can even make this head have an arbitrary depth, and we call this type of architecture Single head, since it uses a single fully connected network on top of an existing backbone.
However, it is not at all obvious that this is the best approach when making an instrument retrieval model. A possible issue is that the weights of the single head architecture are all interconnected and therefore the model may learn spurious or noisy correlations between instruments. Furthermore most of the work done in the field of instrument recognition is based on predominant instrument identification. When all of this is taken into consideration we decided it might be to create a smaller classifier head for each of the instrument labels, that is to allow the model to have multiple heads. Each head is responsible for finding its own instrument.

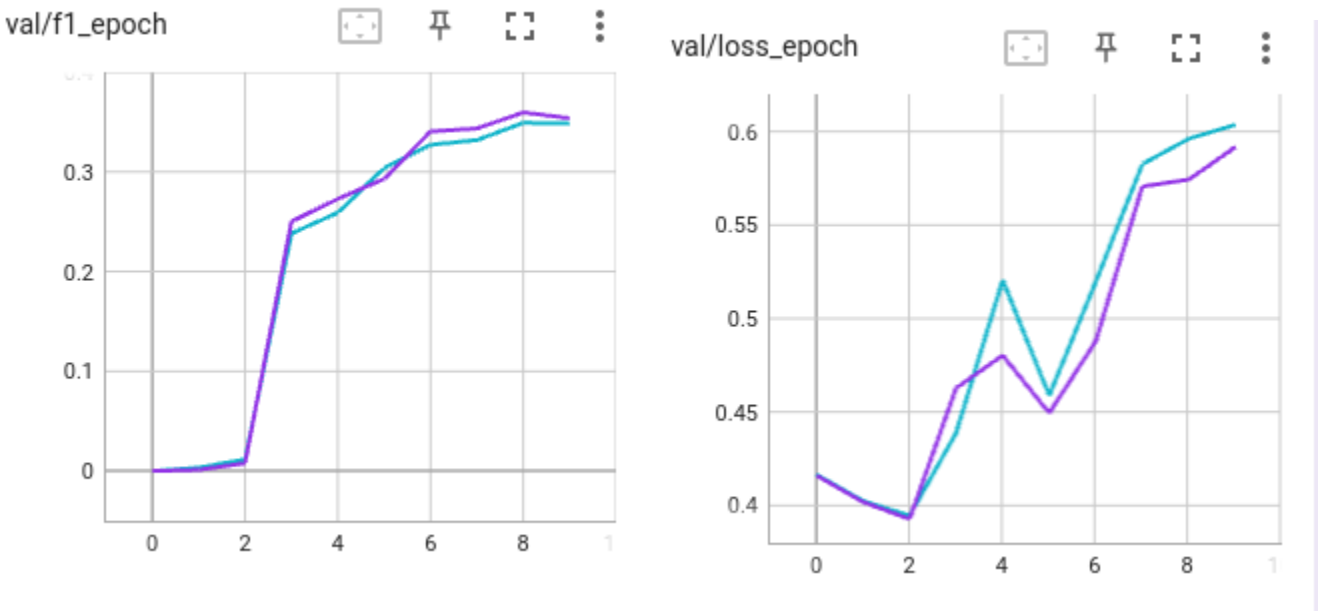


Figure: Comparisons between the Fluffy and SingleHead classifier head. In this experiment Fluffy seems to have a slightly lower performance; this might be due to the fact that the produced features are not general enough.

Thus the Fluffy architecture was born, the fact that we had multiple heads on a single backbone made us think of a three headed dog, and while Cerberus sounded too sinister we decided to name it Fluffy after Hagrid's three headed dog from the Harry Potter series. We hypothesize that this division of labor might make the model more robust to noise and spurious correlations that might be present in the training set.

While this approach might seem unconventional, we name several of its advantages. Firstly it offers a lot of flexibility, the backbone can be pre-trained with a different procedure, or on a different task altogether. For instance the backbone can be trained using metric learning to extract single instrument features. Next, even though having separate instrument heads increases the number of the parameters the model has this is not a significant increase. Furthermore, once a model has been trained to distinguish IRMAS instruments, adding a new instrument to the model consists of simply adding a new head, which can be trained with most of the model frozen. Lastly if each classifier head is an Attention Head it can increase the interpretability of the model, and can help with labeling track segments. For instance if we have a track consisting of a guitar and human voice, we can look at the attention scores for each head at every timestep and determine which timestamps are most insightful for instrument detection.

Backbones

Most of our modeling approaches reside on a simple formula: we use a pre-trained backbone that serves as a feature extractor, and stick a shallow classifier head on the features that the backbone provides. The only exceptions to this scheme were the LSTM and SVM baselines. ConvNEXT MobileNet AST. We first train the model's head with a frozen backbone which forces the weights of the head to depend on the features of a pre-trained backbone. After a few epochs, we unfreeze the "trainable backbone" which is a subset of the backbone. In most cases a trainable part of the backbone becomes the whole backbone, but in case of larger models such AST it might not be necessary to finetune the whole backbone. In this case, we leave the non-trainable backbone frozen forever. The only exception is that we don't freeze batch norms in any layer whatsoever. For each model, we specify two lists of parameters. One defines the trainable backbone, and the other head of the model. Parameters that are in neither list are considered a part of the non-trainable backbone.

Epoch	Non-trainable backbone	Trainable Backbone	Head	Epoch	Non-trainable backbone
n=0	frozen	frozen	trainable	n=0	frozen
n=1	frozen	frozen	trainable	n=1	frozen
n=2 (finetune_head_epochs)	frozen	trainable	trainable	n=2 (finetune_head_epochs)	frozen

CNN models

Convolutional neural networks

Our first approach for this problem were convolutional neural networks. Our architectures consist of a backbone and a classifier head. The classifier head consists of a linear layer, layer normalization, ReLU (rectified linear unit) and a final linear layer. The backbones we opted for are ResNeXt50, EfficientNet V2 small, ConvNeXt small and MobileNet V3 large. For our experiments we mostly stuck with the smaller version of these architectures for quicker experimentation with different hyperparameters. In contrast, we actually noticed using bigger versions of the previously mentioned architectures achieved worse results..

At first, our approach consisted of creating a spectrogram given an audio sequence and resizing it to the size the model expects. Since the spectrogram size varies depending on the length of the audio sequence, we noticed this wasn't the right approach because it meant that we stretch the pixels by a different factor.

Our modification to tackle this was simple: take an audio sequence, transform it into a spectrogram that is equivalent to a 20 second spectrogram. Since there is such a discrepancy between the training and validation set, this meant spectrograms from the training set contain around 85% black pixels. We decided to change our approach once again.

Taking this problem into consideration, we turned to chunking. Given a spectrogram of any width, we can divide it into fixed size spectrograms. Given that the training set audio sequences are exactly 3 seconds long, we opted for spectrograms that represent 3 second long audio sequences. Since the validation set consists of audio

sequences of different lengths, our system would divide it into 3-second chunks and pad the last chunk with black pixels. After predicting the instruments in each of these chunks, the final prediction consisted of all of the instruments predicted in the chunks. We thought this was a more sophisticated way of solving this problem, and that it would benefit our validation and test metrics.

Another modification was taking into account the stretch of the width of the spectrogram. A way to tackle this was dynamically calculating the length of the audio sequence it represents depending on the sampling rate and the hop size. This meant that we only had to resize the height of our spectrograms. Since these sequences were usually longer than 3 seconds, we decided to repeat the beginning of the audio sequence to account for the black pixels that would otherwise occur. Besides, a way to completely avoid resizing is also possible. It includes setting a certain hyperparameter e.g. n_mels for the mel spectrogram to the height of the image that the model expects.

Finally, we noticed our resizing didn't work properly for some models, even though training was working. As we delved deeper into the problem, we noticed that all of these architectures are actually invariant to the dimensions of the input. This is because there is a pooling layer over the height and width of the final feature map, providing a representation that is equal to the number of channels. In fact it turned out that our best convolutional models made use of this insight. This was more of a feature rather than a bug.

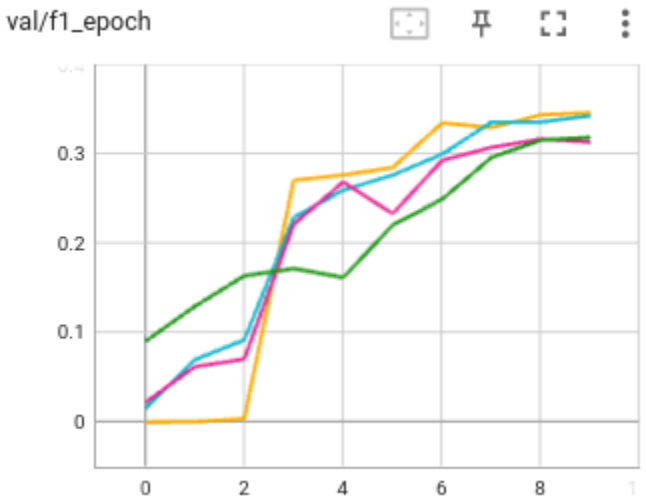


Figure: The performance of different convolutional models ConvNeXt small in orange, EfficientNet V2 s is in blue, ResNeXt 50 32x4d in magenta, and MobileNet V3 large is in green. All models were trained with the same data and in the same regime.

Pretraining

A thing to take into consideration before training convolutional models for any application is that they're usually pretrained on the ImageNet dataset. The dataset covers a wide range of classes: everyday objects, vehicles, animals, flowers, etc. Unfortunately one of the classes is not a spectrogram, so we decided to take matters into our own hands.

We designed a pretraining procedure inspired by the ELECTRA pretraining approach used in natural language processing. Given an example, we consider a different random audio sequence from the dataset. We transform both sequences into spectrograms. Next, we chunk both images into k x k patches, where we set k to 4. Then, a certain amount of patches in the first image gets shuffled. The amount of patches is controlled by a hyperparameter denoted by the shuffle probability. Next, we use another hyperparameter called replace probability. It controls the amount of patches in the first spectrogram that are replaced by patches located at the same positions from the second spectrogram. Finally, we also add a black patch probability, which completely blacks out a certain amount of patches. The pretraining objective is three-way classification. The model is required to learn to distinguish shuffled, replaced and original patches. We use the cross entropy loss and calculate it only over the patches which aren't blacked out. Although we implemented the whole pretraining procedure we didn't use it to train the model because of the time constraint.

Audio Spectrogram Transformer (AST)

The Audio Spectrogram Transformer is a large state of the art model primarily designed and trained for audio event classification. One of its advantages is that it was trained on the complete AudioSet and thus is capable of distinguishing a staggering amount of sounds. It works by first extracting overlapped patches from the spectrogram, these patches are then linearly projected to generate vectorized inputs to the transformer, also before the transformer encoders the vectors corresponding to patches were positionally embedded. The transformer encoder follows the typical architecture present in transformer models, the last layer is a linear classification layer that outputs the logits for AudioSet classes.

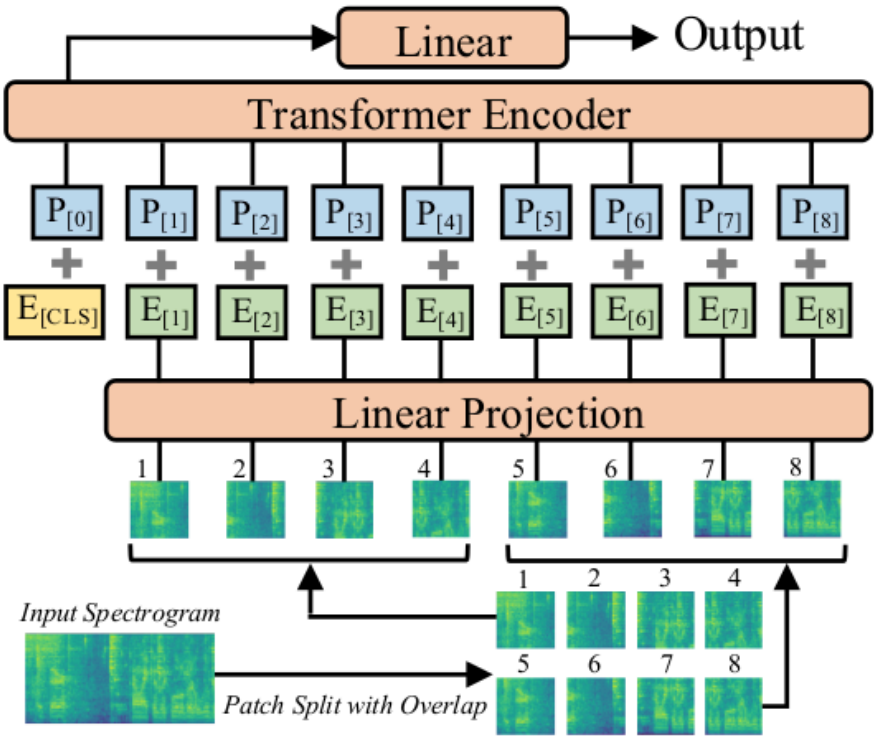


Figure: AST architecture overview

As we've mentioned these features were used for data cleaning and managing guitars in the case of OpenMIC, however AST can also be used for classification, and due to its size and large pool of data it was trained on it can achieve F1 scores on IRMAS data of 0.51 to 0.54 without any additional finetuning. This was done by finding the classification logits corresponding to IRMAS instruments using them for instrument retrieval. Other class logits were ignored.

MobNet

Our last try revolved around the responsibility of the model's parameters. A big problem with our approach is we use a single backbone for eleven tasks, meaning the parameters need to be very general to solve the 11 tasks accurately. A way to solve the problem of the responsibility of parameters is by using different backbones for each of the tasks, but this meant the model would have an enormous number of parameters. Here, the MobileNet V3 large architecture comes into play, since it only has 5.4 million parameters. Our idea was to create a module with 11 different MobileNet models where each of the models is responsible for a single instrument, which we called MobNet. For comparison, this model is smaller than the ConvNeXt small model (50 million parameters), which we used in most of our experiments.

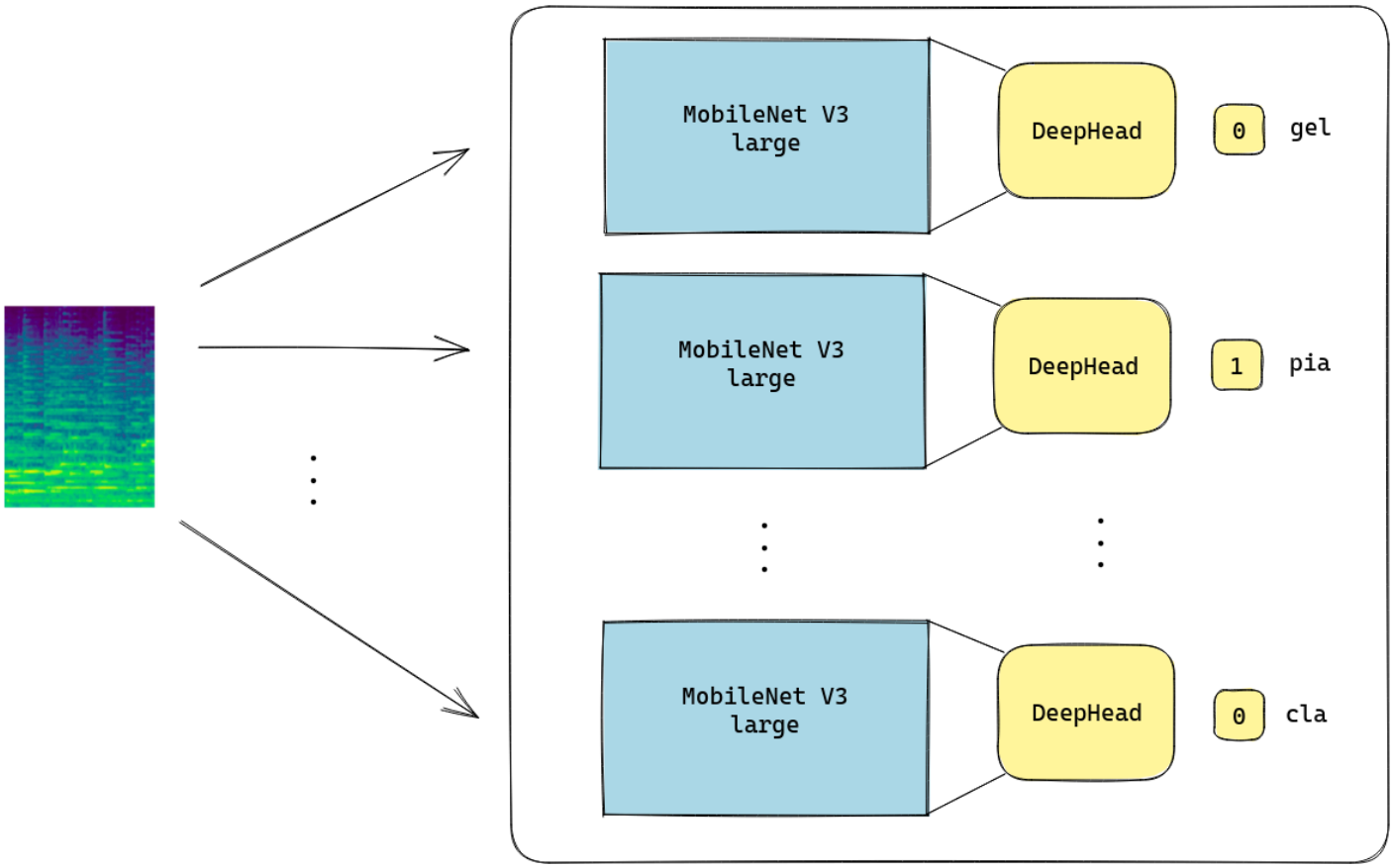


Figure: The MobNet model, consisting of 11 MobileNet V3 large models, followed by a DeepHead module.

Unfortunately, the downside of this kind of architecture is that 11 forward passes through the backbone are required as well as 11 forward passes through the classification head. However, an upside is that we can easily add a new instrument by simply training another MobileNet model. Unfortunately, this model turned out to be surprisingly bad, achieving only a 0.2433 validation F1 score.

Training details:

Finding optimal decision threshold

As we've mentioned in the introduction during training we kept the decision threshold fixed at 0.5, later when several promising models emerged we decided it was necessary to find the decision threshold that will ensure the best F1 metric on the validation set. To this end we made another validation set that was representative of the one given by the competition organizers. The data was from the validation set except we implemented a strong sum with N ranging from 4 to 6, which is similar to the data presented in the official validation set. Next we generated prediction probabilities with our models in question and for every model a search was performed to find the best decision threshold. We sampled 1000 evenly spaced thresholds from the [0,1] interval, and once the best was found, let say its value is *p,* we next sample 100 thresholds from the interval [*p* - 0.1, p + 0.1] uniformly to finegrain the decision boundary.

Optimizers and schedulers

Optimizers

The optimization algorithm often determines if the model will find a suitable optimum at all; in all of our experiments we've used either the Adam or the AdamW optimization algorithms. Adam (short for Adaptive Moment Estimation) combines the benefits of two other optimization algorithms, namely, RMSprop and momentum-based optimization. It maintains an exponentially decaying average of past gradients (first moment) and an exponentially decaying average of past squared gradients (second moment) to adaptively adjust the learning rate for each parameter. The update step in Adam involves calculating the gradients of the network's parameters, scaling them based on the running averages of the first and second moments, and applying the scaled gradients to update the weights. The learning rate is adjusted for each parameter individually.

AdamW extends the Adam optimizer by incorporating a technique called weight decay. Weight decay is a regularization technique that penalizes large weights during training to prevent overfitting. It is typically implemented by adding a term proportional to the weight values to the loss function, which encourages the network to learn smaller weights. The key difference in AdamW lies in how weight decay is applied. In standard Adam, the weight decay term is included in the calculation of the gradients during the update step. However, this approach has been shown to have some undesirable effects, such as overly shrinking the weights and reducing the effectiveness of the adaptive learning rate. In AdamW, weight decay is applied directly to the weights after the parameter update step. This means that the weight decay does not contribute to the calculation of gradients, allowing the adaptive learning rate mechanism of Adam to function as intended. The weight decay term acts independently on the weights, preventing their growth during training without interfering with the adaptation of the learning rate. In summary, the main difference between Adam and AdamW lies in the handling of weight decay. Adam incorporates weight decay in the gradient calculations, potentially affecting the adaptive learning rate behavior, while AdamW applies weight decay directly to the weights after the parameter update, preserving the adaptive learning rate properties. AdamW is generally preferred over Adam when weight decay is desired to regularize the model.

Schedulers

The One Cycle Learning Rate Scheduler is a technique for dynamically adjusting the learning rate during training to improve model performance. It was introduced by Leslie Smith in his paper "Cyclical Learning Rates for Training Neural Networks". The basic idea behind this technique is to start with a relatively low learning rate, gradually increase it to a maximum value, and then gradually decrease it back to a low value.

1. The One Cycle LR Scheduler consists of three phases:
2. Warmup phase: In this phase, the learning rate is gradually increased from a very low value to the maximum learning rate over a few iterations. This allows the model to gently ramp up to its maximum learning rate.
3. Annealing phase: In this phase, the learning rate is gradually decreased from the maximum learning rate to a low value over a few iterations. This allows the model to gradually converge to the optimal solution.
4. Final phase: In this phase, the learning rate is kept constant at a very low value to fine-tune the model.
5. The cycle is called "one cycle" because it consists of one full cycle of the warmup, annealing, and final phases. The cycle length and the maximum learning rate are hyperparameters that can be tuned based on the specific problem and model architecture.
6. The One Cycle LR Scheduler has been shown to improve the training speed and final accuracy of deep learning models. It helps to prevent the model from getting stuck in local minima by periodically increasing the learning rate, allowing it to explore a larger portion of the parameter space.

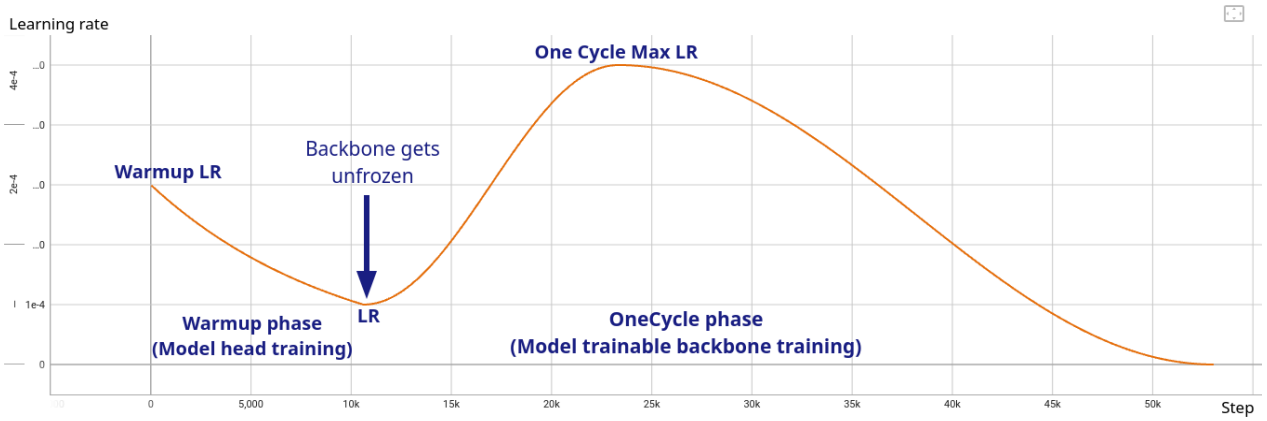


Figure: One cycle cerning rate scheduler procedure.

Naive models:

As every investigation goes, before we try deep learning techniques and the state-of-the-art models, we should try and see how some naive and shallow models work on this problem. By examination of some earlier work on this problem, the Support Vector Machine (SVM) model has shown to be a great way to tackle this problem. Now, the first problem we notice is that the SVM is a classification algorithm, i.e. it returns one of two values and we need a 11-dimensional vector as an output. For this purpose, we should modify a simple SVM algorithm to return an 11-dimensional vector which tells us which ones of the given instruments are present in the given audio file. There are many potential ways to do this, but we decided to go with the simplest one here. We simply build a model called MultilabelSVM which consists of 11 different simple SVMs. Each of the 11 SVMs acts as a binary classifier for each instrument telling if a given instrument is present in a file or not. In principle, we could set the kernel and other hyperparameters individually for every SVM, but as we aim at a simple model, we will set the same hyperparameters for each of 11 SVMs so that MultilabelSVM takes these parameters and forwards it to its 11 SVMs it consists of. Each model is trained on the IRMAS Training data and labels and it is tested on one third of the IRMAS test data as the whole test data took too long to load. Testing on one third of the test data should not be a problem because, as we have shown earlier, there is so much data in the test dataset that even one third of that data should give accurate metrics of a model. Also, training is done on raw features as well as the scaled one. We use the MinMaxScaler which scales all the features to be between 0 and 1. In the end, the comparison is made between these two approaches.

Features used are the ones already described in the classical features analysis section. There are a total of 28 of them and they consist of the classical features from the *librosa* package (such as the spectral centroid, zero crossing rate and similar) and from the *antropy* package (various types of entropies).

The first and the simplest model is the one with the linear kernel meaning the decision boundary will be linear. Obviously, we expect the model to work better with the more complex kernels, but as we again aim for simplicity, we will try how the linear kernel works for this example. As far as other hyperparameters are concerned, only regularization parameter C for the L2 regularization was changed. In the table below we present the main metrics and results while additional information as the metrics for each instrument and the model architecture can be seen in the attached MultilabelSVM code.

Linear kernel:

Scaling and C	Accuracy	Hamming distance	F1 score	Precision	Recall
Raw features (C=1000)	0.84	0.16	0.1	0.27	0.1
Scaled features (C=1)	0.84	0.16	0.08	0.39	0.05

Second model is the one with the radial basis function (rbf) kernel. If we take a look at the data clusters shown in the beginning of the document, we can get some intuition about the features. Although those figures do not show real features, they are sufficiently insightful to see that the clusters are shaped in a way that is definitively more appropriate to separate with the rbf function than a linear function. So, we decide to use a rbf function and also we use the L2 regularization parameter C in order to get more accurate models. We present the results for the rbf kernel in the table below.

Radial basis function (rbf) kernel:

Scaling and C	Accuracy	Hamming distance	F1 score	Precision	Recall
Raw features (C=10 000)	0.84	0.16	0.02	0.08	0.01
Scaled features (C=250)	0.79	0.21	0.19	0.32	0.2

Although the MultilabelSVM model works fairly well regarding the fact that it is a very simple model, we can improve it even more in multiple ways. For example, here we use only the IRMAS Training and Test datasets but we could use other aforementioned datasets and different splits of the IRMAS dataset.

Additionally, If we take a look at the PCA clusters in the feature analysis chapter, we can see the approximate shape of the clusters for every instrument so we could construct a new model which would have 11 single SVM heads for each instrument but they would have custom kernels and other hyperparameters based on the shape of the clusters and other information about the data. In that way, classification quality could be optimized for every instrument.

Another obvious improvement is using some more complex model which would allow single SVM heads to interact in some way, i.e. to use a model in which different SVM classifiers aren't independent but linked in some way. This was not explored further throughout the project.

Model graveyard

In this section we briefly mention a couple of custom architectures we tried to implement, but due to the much lower performance compared to existing architectures were not deployed.

LSTM

The first deep architecture we took into consideration was the LSTM model on Mel spectrograms. When the waveform is chunked on 3 seconds and the hop length of the fourier transform is set to 512, we are left with a timeseries of 130 time steps. Each vector in the series has n_ mels = 128 components. Due to the serial nature of sound we decided a good baseline for deep architectures would be an LSTM architecture with Bahdanau attention. Aa a deep baseline it managed a F1 metric of 0.2.

ConvLSTM

An obvious issue with the LSTM model on spectrograms is that the time series is far too long for the information to properly propagate through all 130 hidden states. A possible solution would be to pool the temporal and frequential information of the mel spectrogram and then implement the LSTM. The pooling is done with convolutional layers, thus the name ConvLSTM. Although it achieved slightly better results than the bare LSTM it did so after more training and with far slower convergence.

Spectral convolution with attentive Fluffy (Good boy)

A simplification of the ConvLSTM is a convolutional layer similar to the one used in ConvLSTM except we do not use the LSTM to extract temporal information rather we use a Fluffy configuration with attention heads. This managed to beat the ConvLSTM in terms of F1 metric reaching up to 0.35. Initially this made us very enthusiastic about the Fluffy architecture.

Wav2Vec CNN with Fluffy

A possible issue with the previous model was that the entire model was trained from scratch, and no pre-trained layers were used. To this end we used the Wav2Vec model that consists of feature extracting one dimensional convolutions that slide on the raw waveform which is followed by a transformer architecture. We removed the transformer and used Fluffy heads with attention in conjunction with the pre-trained Wav2Vec feature extractor. The performance was comparable to the spectral convolution with Fluffy, at a significant increase in the number of parameters.

6. Results

Our final section is concerned with the results we achieved through experimentation. We focused on experimentation with convolutional models, since they generally have less parameters than the transformer models. First, we will consider the best models regarding the validation F1 score using the IRMAS multi-instrument dataset, after which the official metric will be considered.

The best validation F1 scores

The figure below represents the validation F1 score of the 5 best models while training over 10 epochs. Most of our best performing models regarding the F1 score use the ConvNeXt small architecture. We can see the biggest increase in the validation F1 score in the third epoch, where the backbone gets unfrozen.

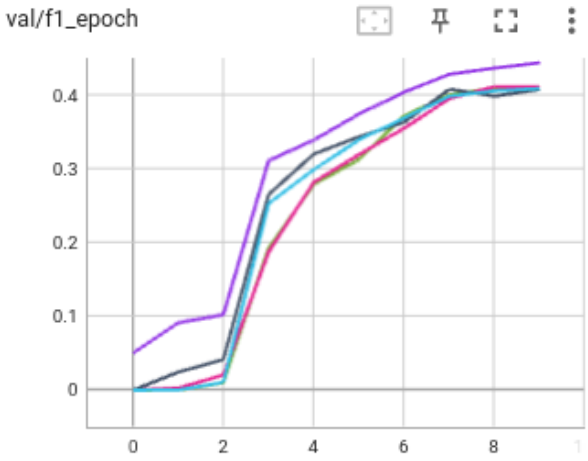


Figure: The validation F1 score of the 5 best models across 10 epochs of training.

Our best performing model is the ConvNeXt small model that uses two datasets: the IRMAS single-instrument + AudioSet. The learning rate is set to 1e-4, which we found works best across all convolutional architectures. The batch size was set to 16, which ensured the best ratio of stochasticity and accurate gradients. An interesting insight we found is that using a bigger image size turned out to be beneficial. The usual image size used for the ConvNeXt small model is 224 x 224, but we used 384 x 384. As mentioned previously, this was a bug we missed, but in the end it turned out to improve performance. We speculate this performance boost happens due to lack of chunking occurring in the training procedure. To make sure the model is robust we use the Adam optimizer with decoupled weight decay regularization, best known as AdamW, with the weight decay hyperparameter set to 1e-2. An important detail is that we use the Fluffy architecture for this model, as it proved to consistently achieve better results than its default counterpart. The model uses only two augmentations: summation of two audio sequences and time shift. Surprisingly, experimenting with a different number of augmentations showed that using more than three augmentations introduced too much noise and worsened our results. The best-performing feature we employed is the mel spectrogram feature, repeated across all channels. This model used our default scheduler, which is the one-cycle learning rate scheduler described in the training details section. As well as the other models, this one uses the ImageNet pretrained weights. We think this had a positive impact even though it wasn't pretrained on spectrograms because the model has knowledge of how to identify features useful for classification. The best validation F1 score this model achieved was 0.4435. Judging by the validation F1 curve (purple curve) across the epochs, this could have been improved by a small margin if we trained it longer.

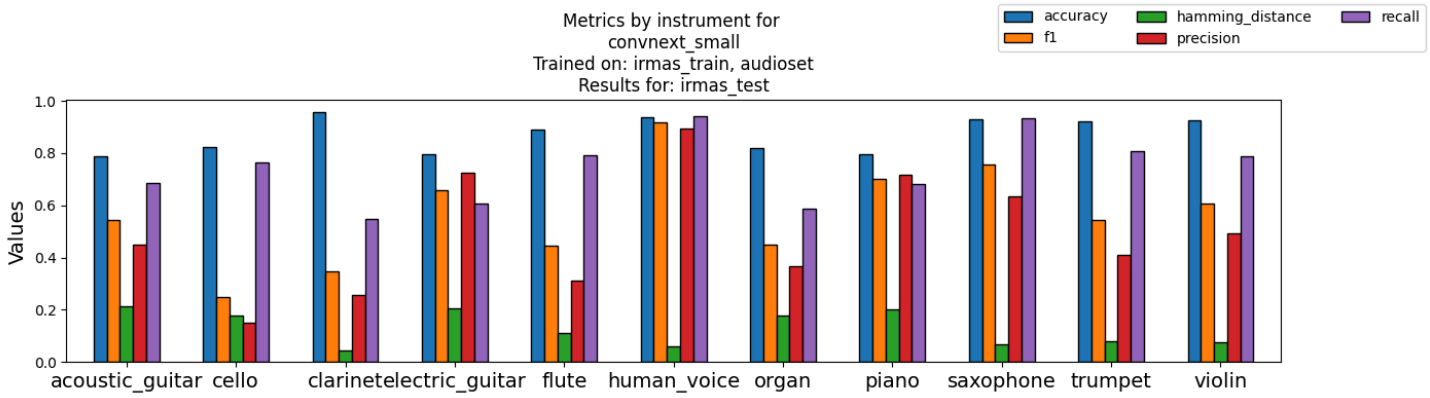


Figure: The metric scores per instrument for the best-performing ConvNeXt small model trained on IRMAS single-instrument + AudioSet.

The figure above shows the performance of our best model in regards to the labels. The highest F1 score is achieved for the human voice label, which most of our models found easy to identify in the audio sequences. Surprisingly, the second highest F1 score is achieved for the saxophone label, which can often be misinterpreted as a clarinet, while the piano label has the third highest F1 score. The model seems to have the most problems with the cello and clarinet labels.

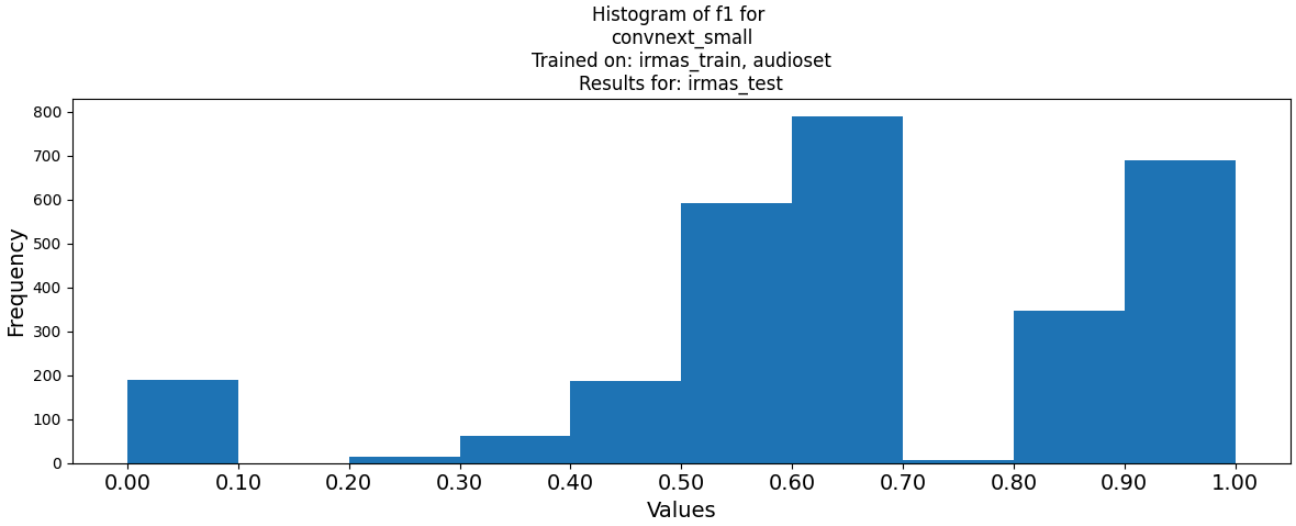


Figure: The distribution of F1 scores per example for the best-performing ConvNeXt small model trained on IRMAS single-instrument + AudioSet.

The figure displays the distribution of F1 scores per example for the best-performing ConvNeXt small model trained on IRMAS singe-instrument + Audioset. The distribution is trimodal, meaning that our model has severe problems with one group of examples, mediocre performance on the second group of examples, and excellent performance on the third group of examples.

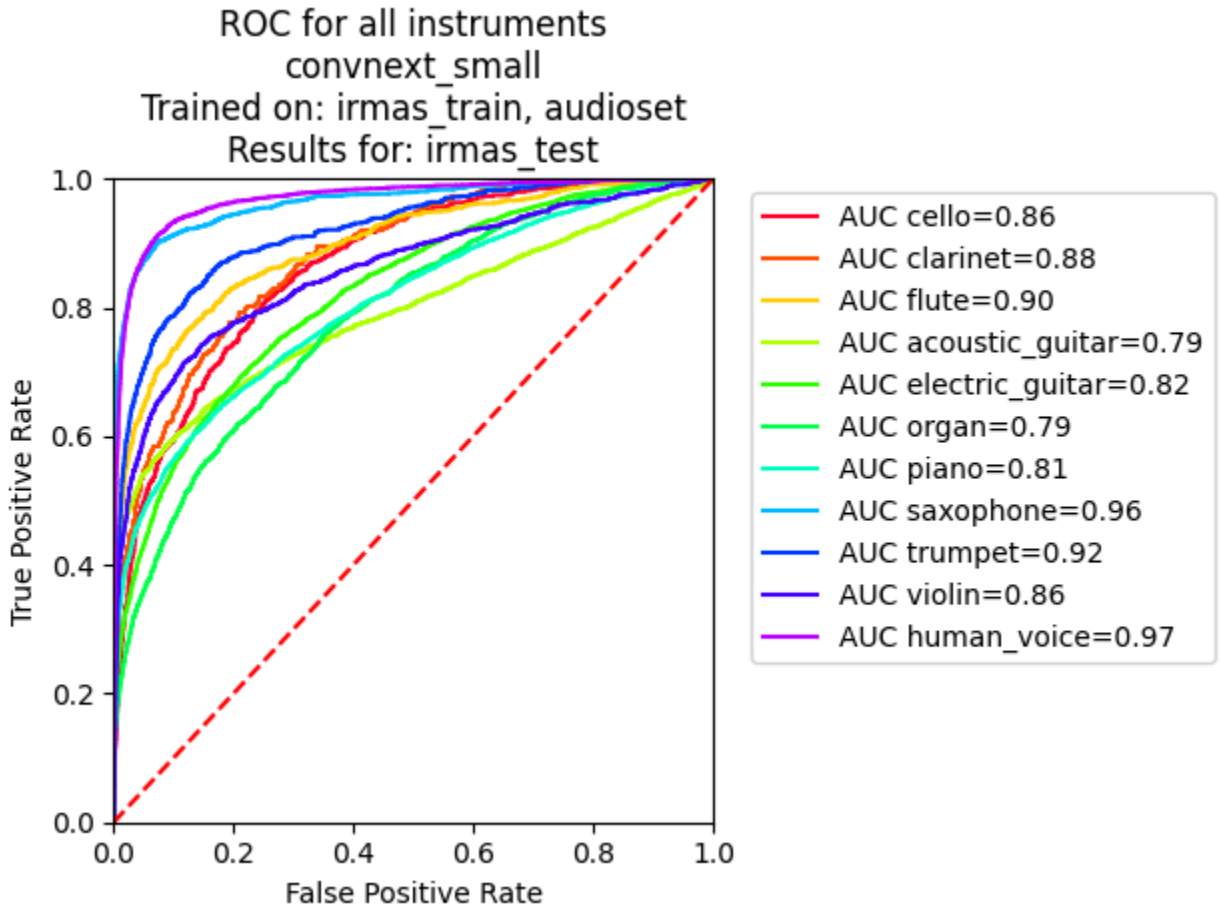


Figure: The ROC curve for the best-performing ConvNeXt small model trained on IRMAS single-instrument + AudioSet.

The final figure represents the ROC curve for the same model. What stands out is that the model has no trouble recognizing the human voice and saxophone, while the most problematic label is the acoustic guitar and the organ. The biggest issues are the acoustic guitar, electric guitar and the organ. We expected a similar outcome, because these instruments have a timbre which is hard to identify when summing the audio sequences, e.g. acoustic guitar can be recognized by the percussive effect that strumming creates, while the organ can be occasionally recognized because by focusing on long and equally loud lower frequencies.

The second best model is the exact same model with the exclusion of the AudioSet data, meaning that the model is trained exclusively on the data provided by the competition organizers. This shows the models' potential when training on more data. This model achieves a validation F1 score of 0.4112.

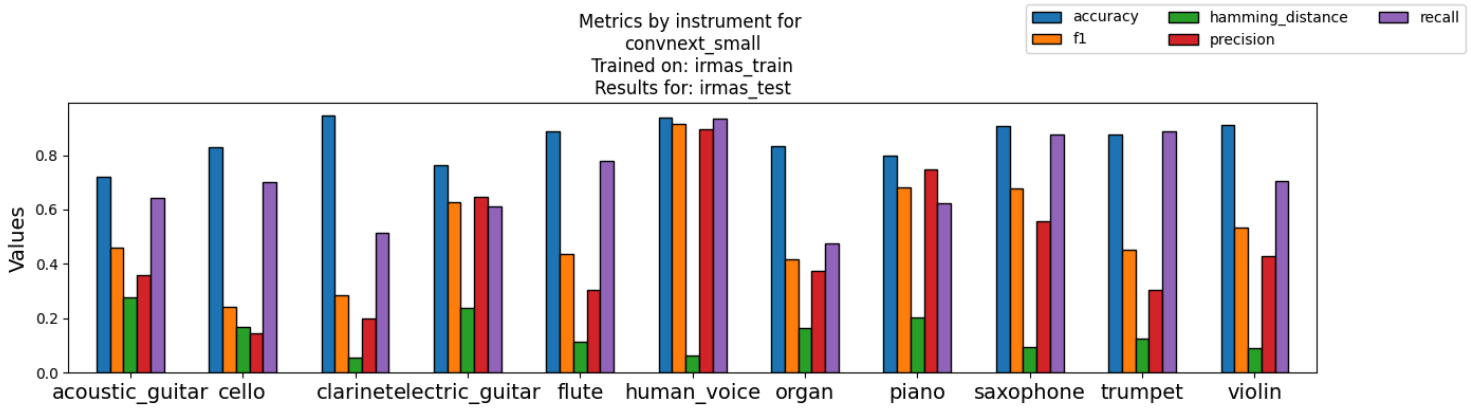


Figure: The metric scores per instrument for the second best-performing ConvNeXt small model, trained exclusively on IRMAS single-instrument.

The figure represents the metric scores for the second best-performing ConvNeXt small model, trained exclusively on IRMAS single-instrument dataset. As well as the previous model, this model works best on the human voice, piano and saxophone labels, although removing the AudioSet dataset diminishes the saxophone F1 score. The most problematic label for this model is cello as well.

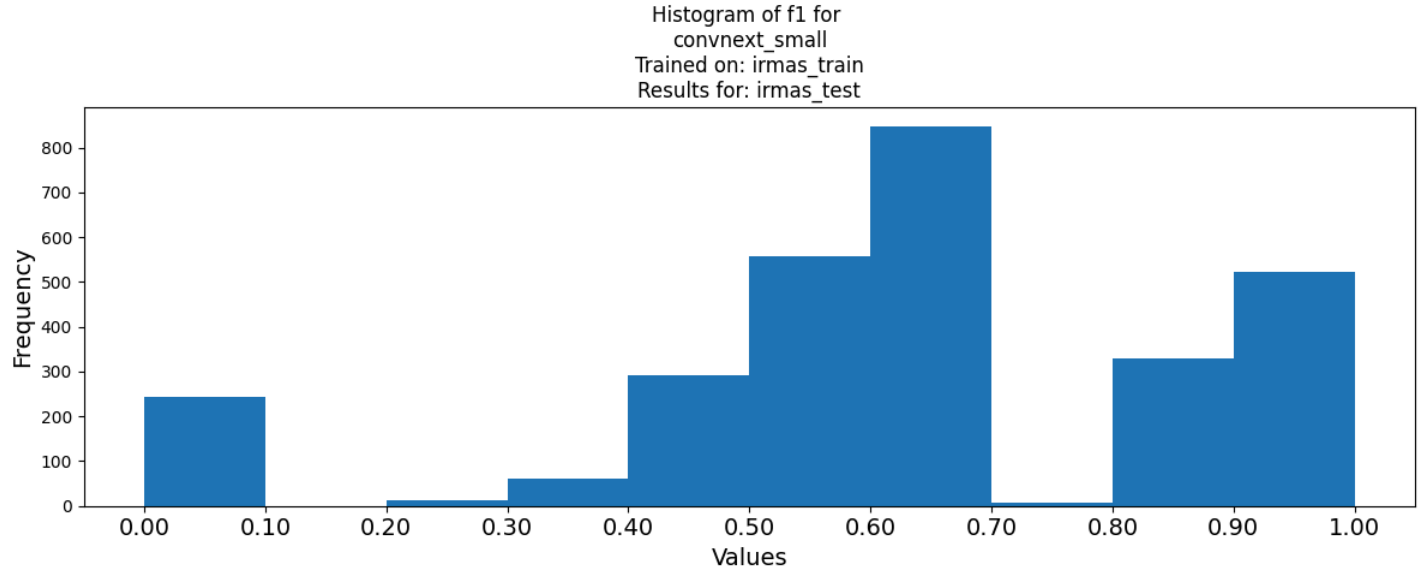


Figure: The distribution of F1 scores per example for the second best-performing ConvNeXt small model trained on IRMAS single-instrument exclusively.

The distribution of F1 scores per example for this model is very similar to the distribution of the previously mentioned model, although we can see that the amount of small F1 scores increased, which seems to be the results of excluding AudioSet.

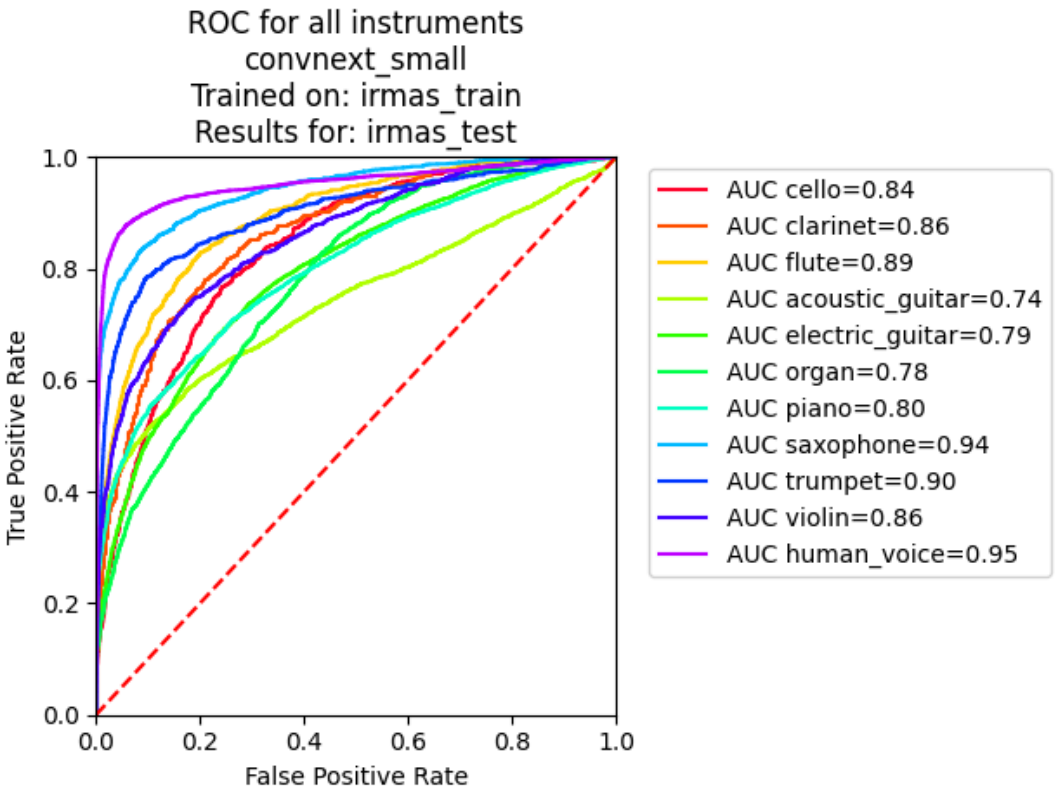


Figure: The ROC curve for the second best-performing ConvNeXt small model trained only using the IRMAS single-instrument dataset.

The ROC curve figure again indicates the deterioration of results with the exclusion of the AudioSet data. The biggest impact can be seen for the acoustic guitar label, where the AUC (area under the curve) is 0.05 smaller. In addition to this, we can see that every value of the AUC is smaller or equal than the previous model, meaning that the AudioSet data is an extremely valuable resource for the task.

The third best-performing model is also a ConvNeXt small model, but without the use of AudioSet and without the use of the Fluffy architecture, while everything else is exactly the same. The validation F1 score this model achieves is 0.4095.

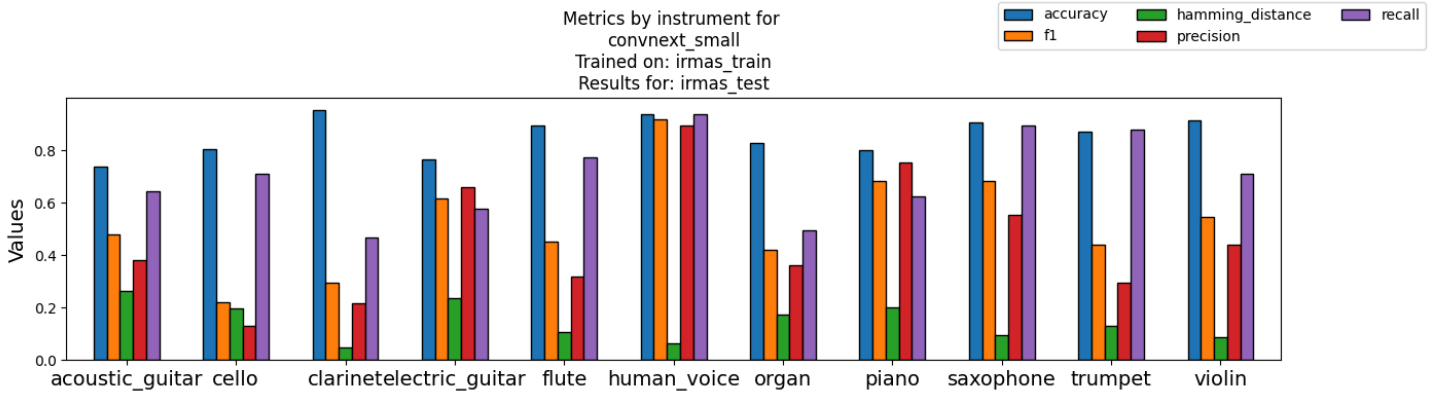


Figure: The metric scores per instrument for the third best-performing ConvNeXt small model trained on IRMAS single-instrument without using the Fluffy architecture.

The figure presents the metric scores achieved by the third best-performing ConvNeXt small model trained exclusively using the IRMAS single-instrument data and without using the Fluffy architecture. All of the metric values roughly stay the same on this figure, which makes sense because the F1 value is much closer to the previous model.

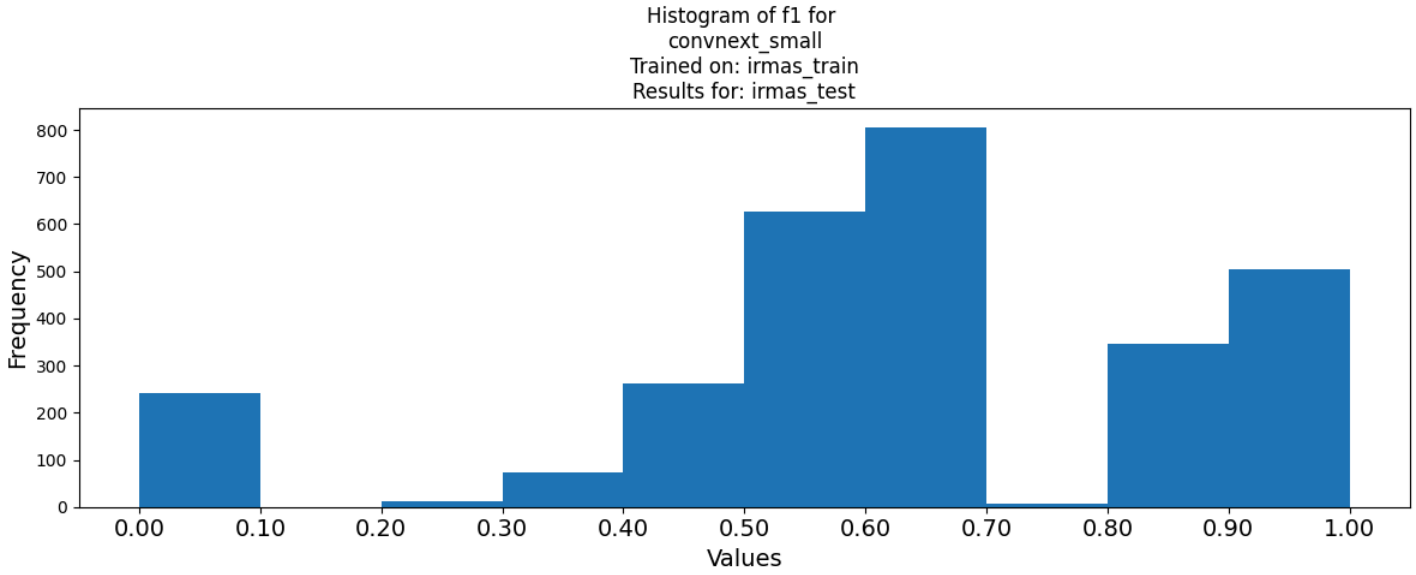


Figure: The distribution of F1 scores per example for the third best-performing ConvNeXt small model trained on IRMAS single-instrument exclusively, without the use of Fluffy architecture.

The figure presents the distribution of F1 scores per example for the third best-performing ConvNeXt small model trained on IRMAS single-instrument exclusively, without the use of Fluffy architecture. The biggest difference compared to the previous model is in the second group of examples, as there is an increase in the number of examples which have an F1 score between 0.5 and 0.6.

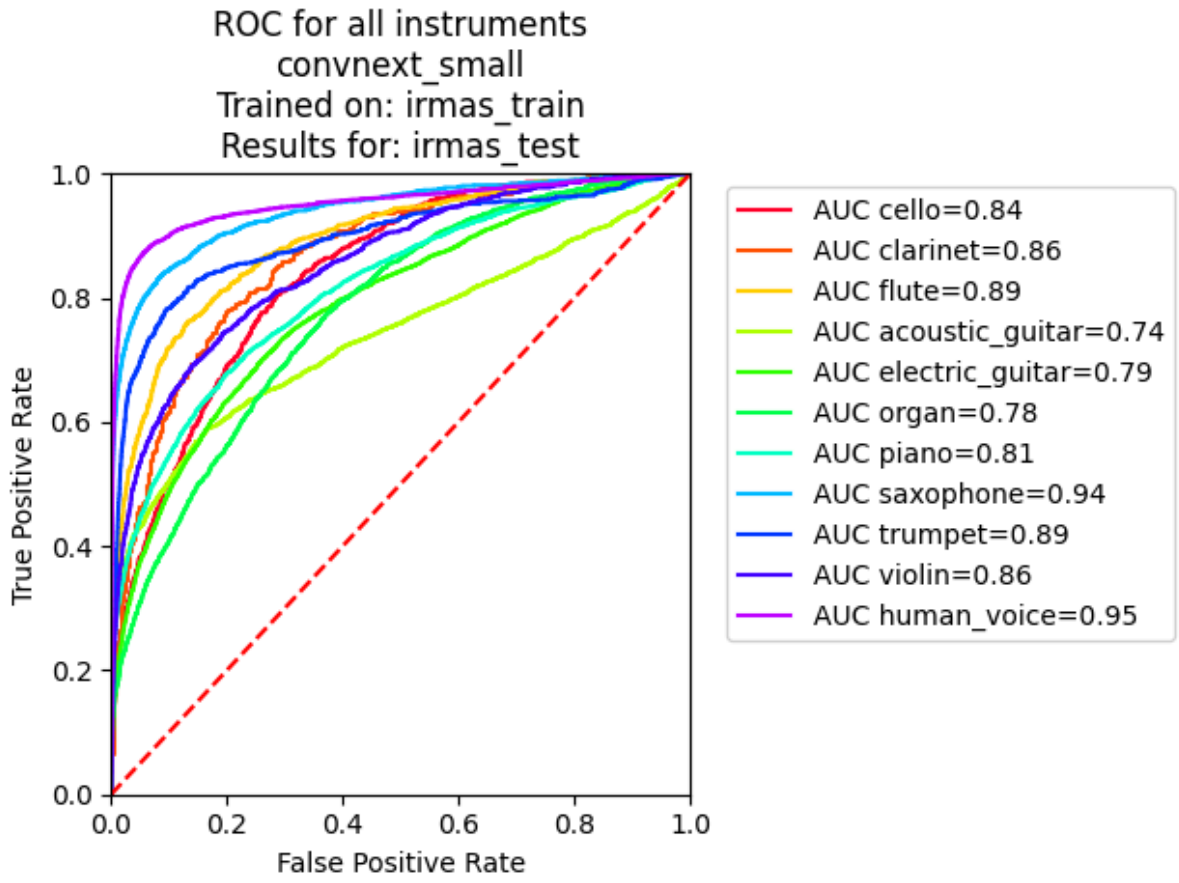


Figure: The ROC curve for the third best-performing ConvNeXt small model trained only using the IRMAS single-instrument dataset, without using the Fluffy architecture.

The ROC curve for the corresponding model can be seen in the figure above. The curves are similar, while the areas under the curve change. The piano label benefits from the exclusion of the Fluffy architecture while it negatively impacts the AUC for the trumpet label.

We've handpicked the 5 best models regarding the validation F1 score and we present their scores in the table.

Table: The top 5 validation F1 scores.

Model	Validation F1
ConvNeXt small + Fluffy, IRMAS + AudioSet, sum 2 + time shift	0.4435
ConvNeXt small + Fluffy, IRMAS, sum 2 + time shift	0.4112
ConvNeXt small, IRMAS, sum 2 + time shift	0.4095
ConvNeXt small, IRMAS, sum 2	0.4084
ConvNeXt base + Fluffy + 224 x 224 image, IRMAS, sum 2	0.4076

The best validation Hamming scores

Even though we mostly refer to the F1 metric as a criterion for model evaluation and comparison, the official metric of this competition is the hamming score (also called hamming accuracy), or just accuracy from now on. In the multi-label setting the hamming loss (distance) is calculated as the mean number of the correctly predicted labels, and the hamming score is 1 - hamming loss.

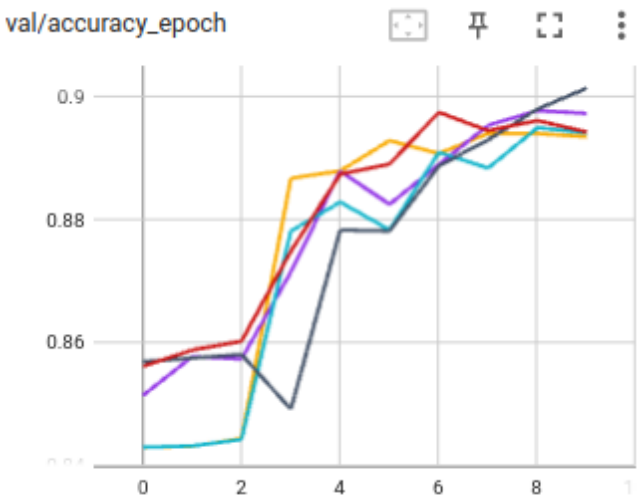


Figure: The validation hamming score of the 5 best models across 10 epochs of training.
ConvNeXt small, batch size 4, IRMAS + OpenMic, sum 2 + time shift , ConvNeXt small + Fluffy, batch size 16, IRMAS + AudioSet, sum 2 + time shift, ConvNeXt small, batch size 6, IRMAS + OpenMic, sum 2, ConvNeXt small, IRMAS, ConvNeXt small, cosine annealing, IRMAS

The figure represents the validation hamming score curves for 5 handpicked models that achieve the highest values.

The best-performing model in regards to the hamming score is the ConvNeXt small model which uses the mel spectrogram repeated over channels to represent audio sequences. The augmentations used for this model are summing two samples and time shifting. The image size used is 384 x 384, even though the model is pretrained on ImageNet using images of size 224 x 224. An interesting insight is that the model with the highest hamming score uses a batch size of 4, as the best-performing model regarding the validation F1 score uses a batch size of 16. The datasets used for this model are the IRMAS single-instrument and OpenMic datasets, while the Fluffy architecture is not included.

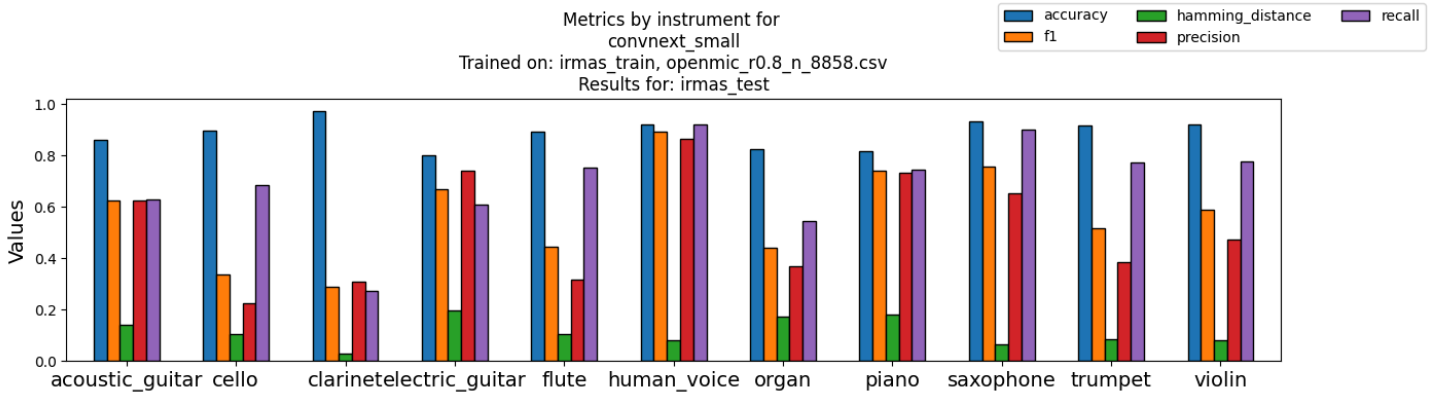


Figure: The metric scores per instrument per instrument for the best-performing ConvNeXt small model trained on IRMAS single-instrument + OpenMic.

The highest validation F1 scores correspond to the same labels as before: human voice, saxophone and piano. The saxophone F1 score is higher than the piano F1 score, which seems to be a recurring theme when using the Fluffy architecture in combination with additional data. The worst F1 score is achieved for the clarinet, which is different than in the case of analyzing the best-performing models in regards to the validation F1 score.

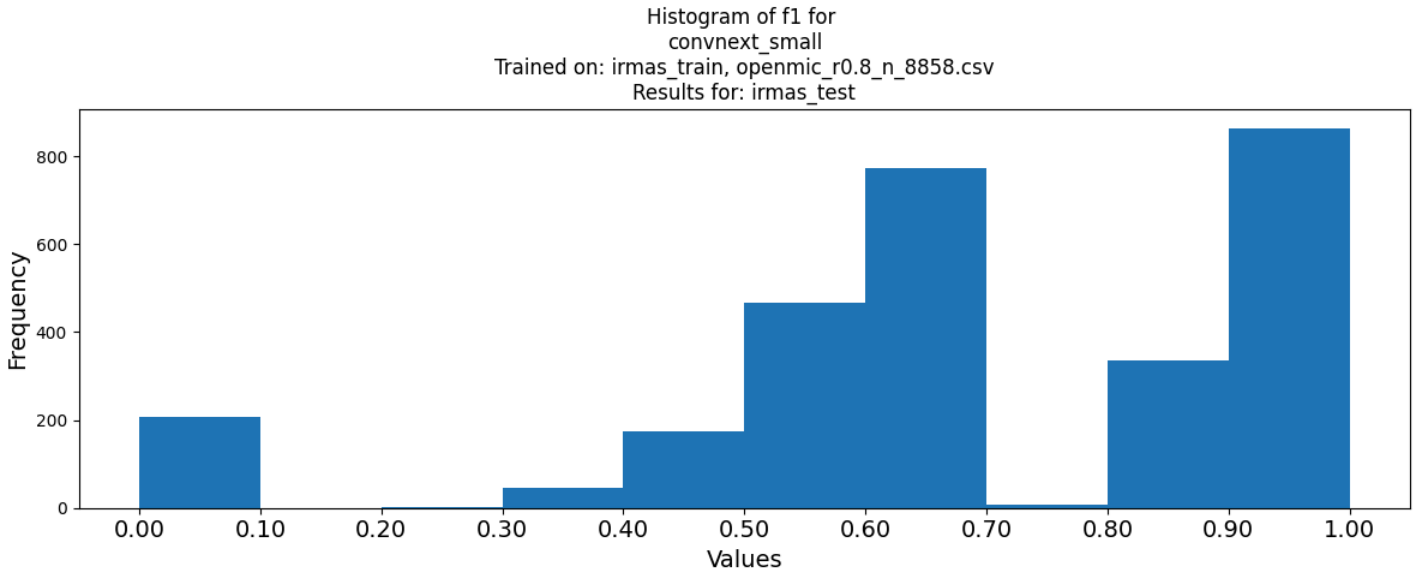


FIgure: The distribution of F1 scores per example for the best-performing ConvNeXt small model trained on IRMAS single-instrument + OpenMIC.

We can see that the distribution of F1 scores in the figure above also contains three groups, but there is a big difference in the second and third group. The amount of examples with a high F1 score is much greater, while the amount of examples with a mediocre F1 score is lower, meaning that this model more accurately predicts a certain sample.

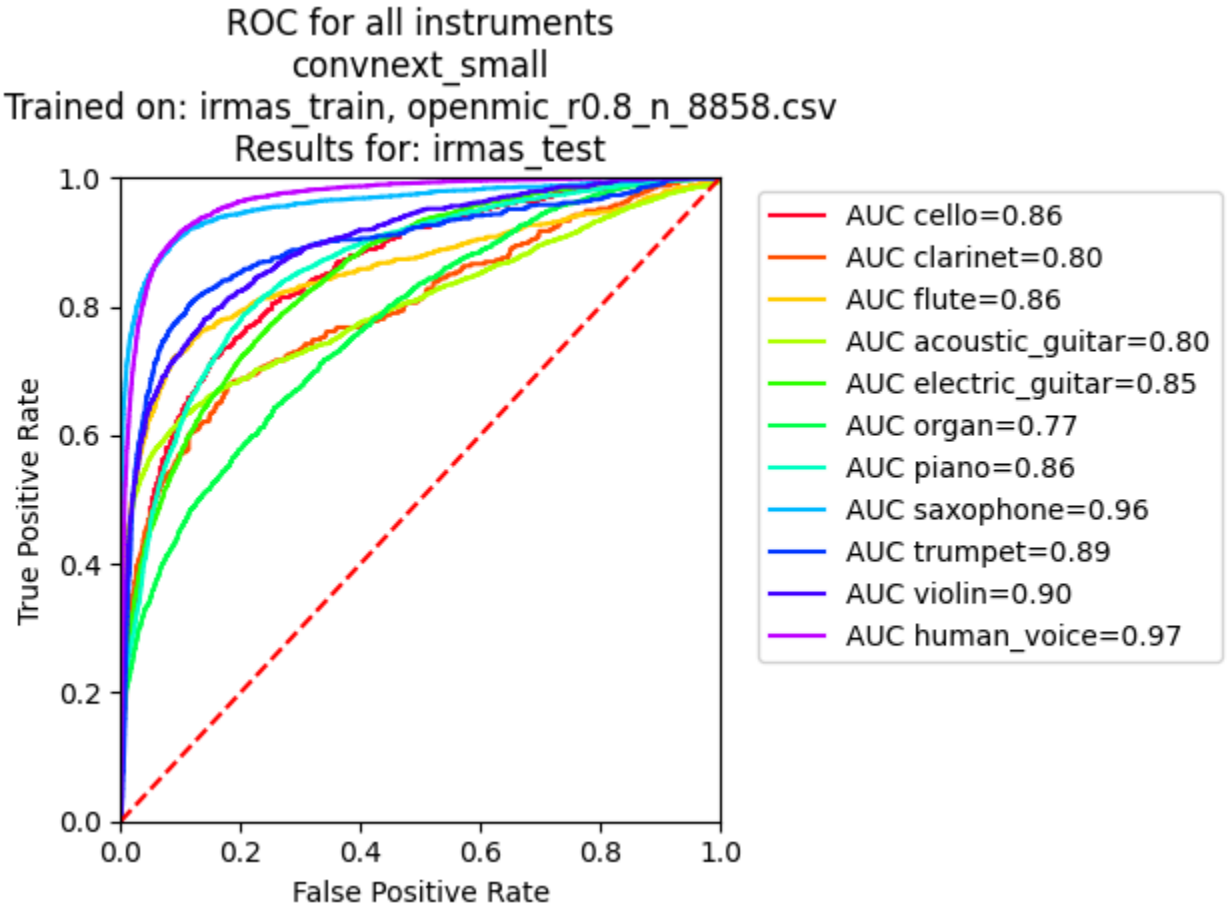


Figure: The ROC curve for the best-performing ConvNeXt small model trained on IRMAS single-instrument + OpenMic.

The ROC curve for the best-performing ConvNeXt small model trained on the IRMAS single-instrument dataset and the OpenMic dataset is shown in the figure above. This model is different in regards to the model with the best F1 as it better recognizes the electric and acoustic guitar, which can be seen in the AUC scores. Another big difference is that the model with the best hamming score has a substantially smaller clarinet AUC, meaning that this model performs much worse on this label.

The second best-performing model regarding the hamming score is the same model with a change in the batch size, the inclusion of the Fluffy architecture and the datasets used. The batch size for this model is 16, while the datasets it was trained on are the IRMAS single-instrument and AudioSet datasets. An important insight is that the inclusion of AudioSet benefits both the validation F1 score and the validation hamming score.

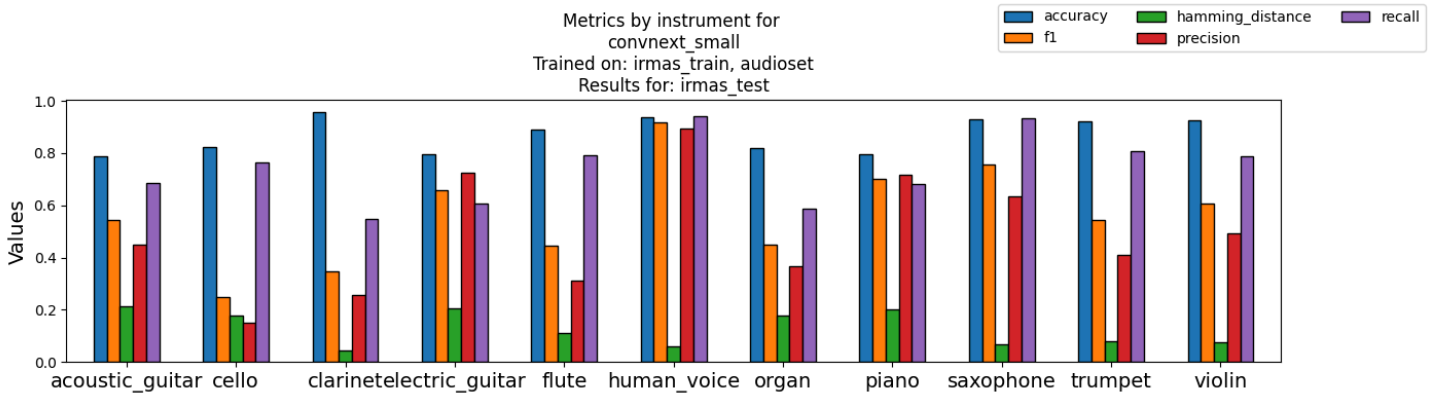


Figure: The metric scores per instrument for the second best-performing ConvNeXt small model trained on IRMAS single-instrument + AudioSet.

The figure above represents the metric scores per instrument for the second best-performing model regarding the validation hamming score. In comparison to the best-performing model, this one has the highest validation F1 score for the same instruments, although the difference in performance between the piano and saxophone labels is more prominent. In contrast to the previous model, this model has the most trouble with the cello label, indicating that the inclusion of AudioSet improves the clarinet F1 score, while the inclusion OpenMic improves the cello F1 score.

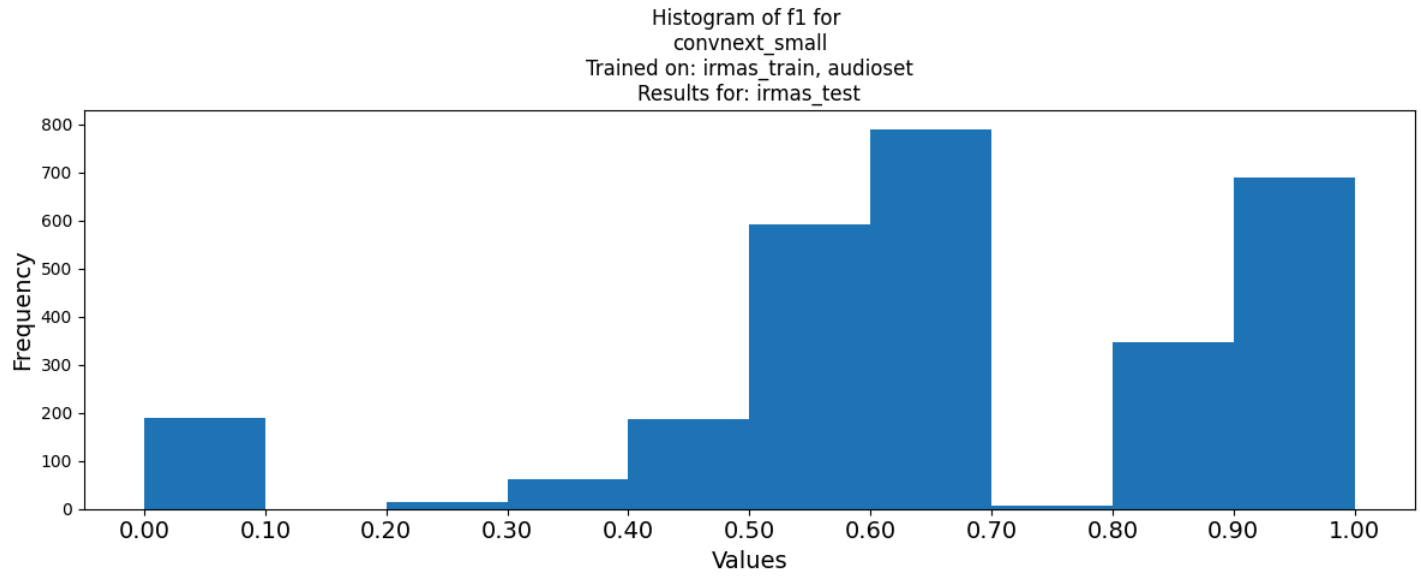


Figure: The distribution of F1 scores per example for second the best-performing ConvNeXt small model trained on IRMAS single-instrument + AudioSet.

The distribution of F1 scores per example is very similar to the previous model. The most prominent difference that can be seen in the figure above is that the group with the mediocre F1 score contains more examples, while the group with the highest F1 score contains less examples.

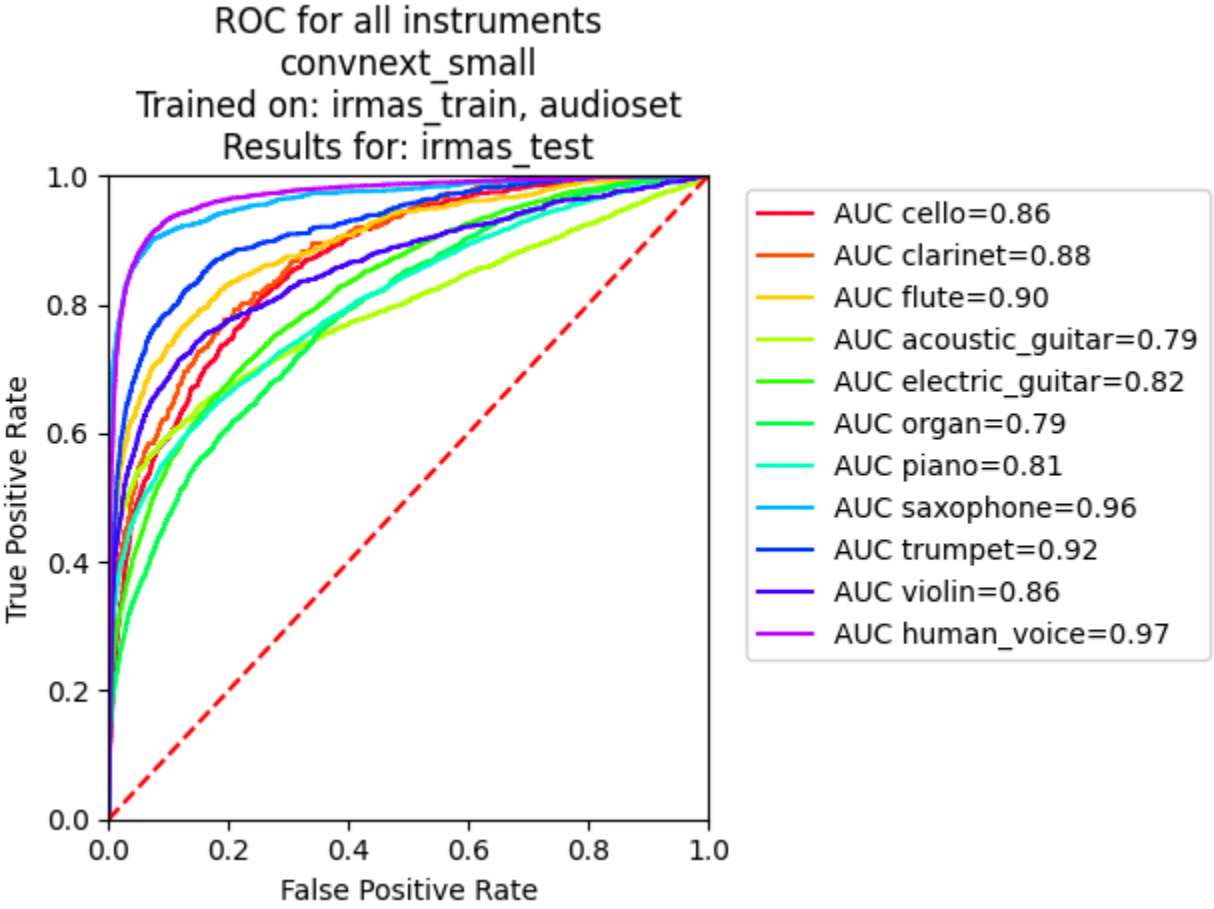


Figure: The ROC curve for the second best-performing ConvNeXt small model trained on IRMAS single-instrument + AudioSet.

The ROC curve for the model can be seen in the figure above. The most noticeable difference is that the AUC for the clarinet label improves by 0.08. This once again confirms that AudioSet improves the performance in regards to the clarinet label. The biggest decline can be seen in the violin AUC, where the difference is equal to 0.04.

The third best-performing model in regards to the hamming score is achieved by the ConvNeXt model in combination with summation of two samples, trained on the IRMAS single-instrument dataset and the OpenMic dataset. Another difference in comparison to the second best-performing model in regards to the hamming score is the batch size. The batch size for training this model was 6, which also suggests that smaller batch sizes benefit the hamming score.

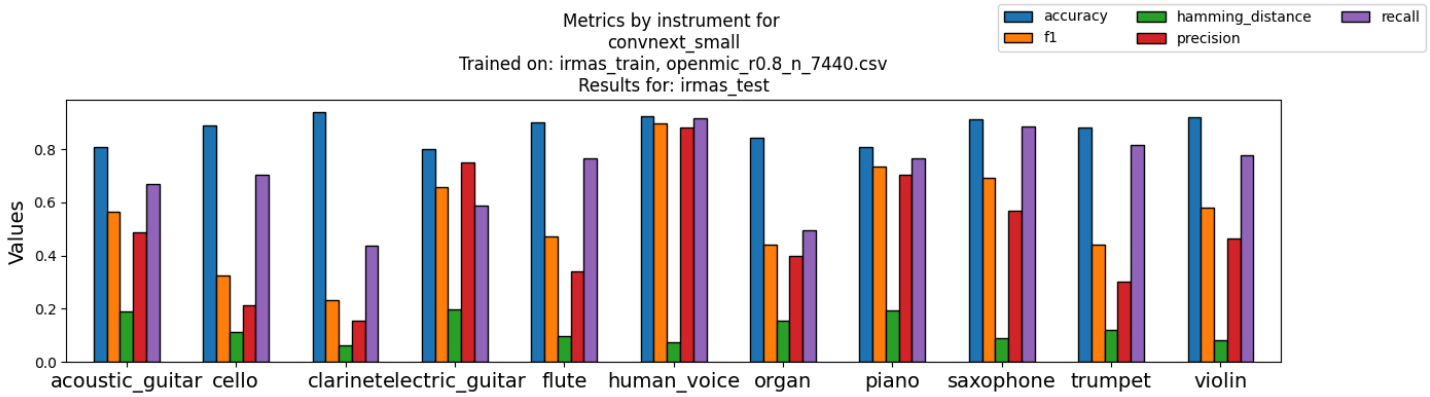


Figure: The metric scores per instrument for the third best-performing ConvNeXt small model trained on IRMAS single-instrument + AudioSet.

The metric scores per instrument for the model are available in the figure above. As in the previous experiments, we can see the improvements in the F1 score for the piano label, which we again attribute to the addition of the OpenMic dataset. The model performs the worst on the clarinet label, same as the other models which use the OpenMic dataset.

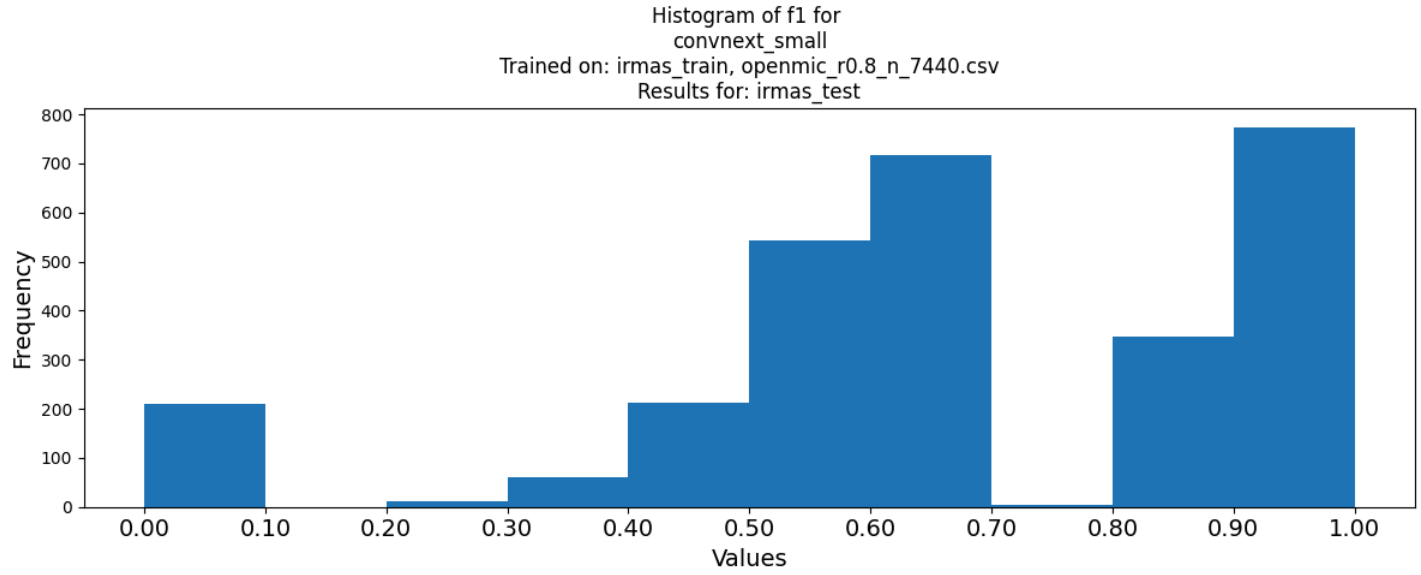


Figure: The distribution of F1 scores per example for the third best-performing ConvNeXt small model trained on IRMAS single-instrument + AudioSet.

The distribution of F1 scores per example for the model can be seen in the figure above. The biggest improvement can be seen in the amount of examples present in the group with the highest F1 score, while the group with the mediocre F1 score consists of less examples.

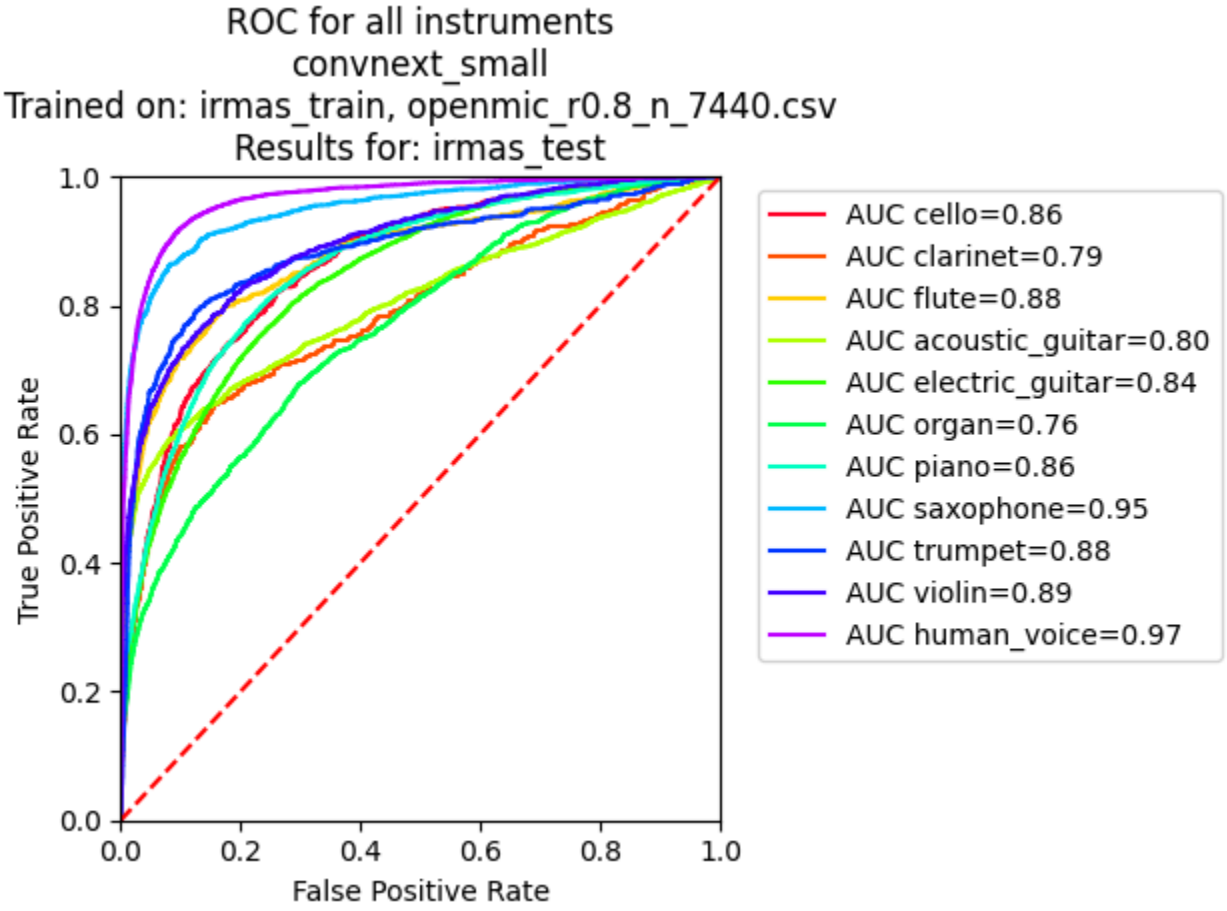


Figure: The ROC curve for the third best-performing ConvNeXt small model trained on IRMAS single-instrument + AudioSet.

The figure above represents the ROC curve for the third best-performing model. The ROC curves and AUC scores for the third best-performing model are very similar to the ROC curves and AUC scores for the best-performing model. The biggest difference in the AUC scores is 0.02, indicating that not much changes in regards to the ROC curves by changing the hyperparameters of the model, while changing the datasets drastically changes the curves and scores.

We've handpicked the 5 best models regarding the validation hamming score and we present their scores in the table.

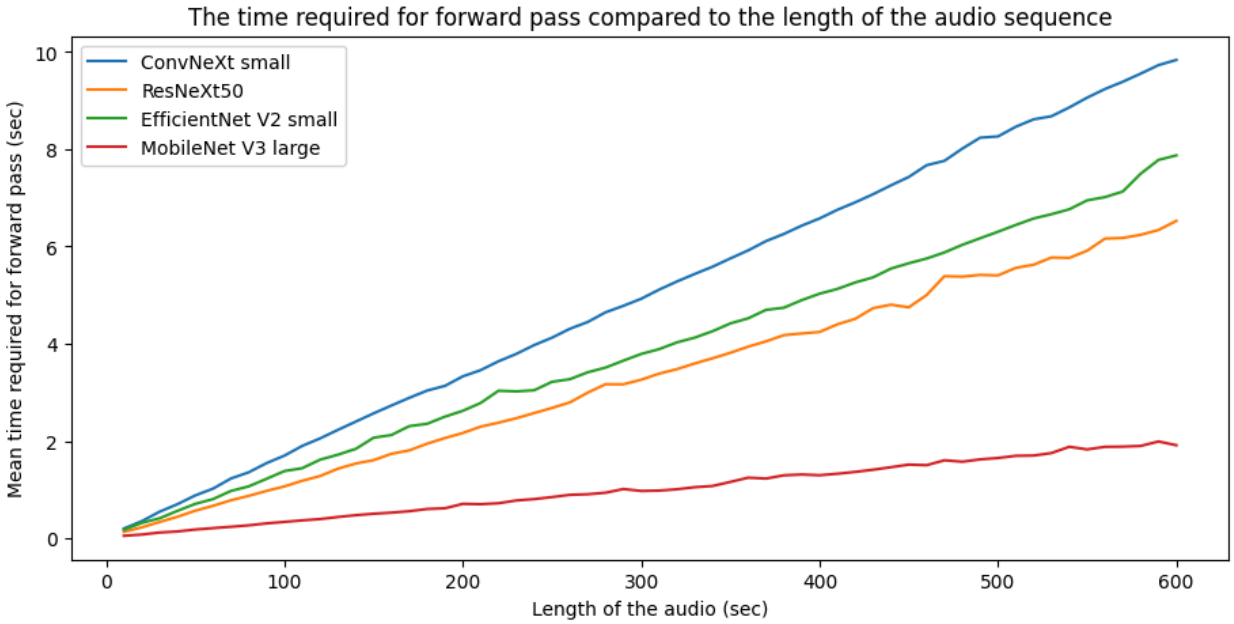
Table: The top 5 validation hamming scores.

Model	Validation hamming score
ConvNeXt small, batch size 4, IRMAS + OpenMic, sum 2 + time shift	0.9014
ConvNeXt small + Fluffy, batch size 16, IRMAS + AudioSet, sum 2 + time shift	0.8977
ConvNeXt small, batch size 6, IRMAS + OpenMic, sum 2	0.8974
ConvNeXt small, IRMAS	0.8949
ConvNeXt small, cosine annealing, IRMAS	0.894

Inference analysis

One thing all machine learning engineers have in common is that they get caught up in creating models that are accurate, but not necessarily fast. We wanted to have a section to compare the time performance of every model. In order to accomplish this, we measure the time needed for the forward pass of each convolutional model 10 times and calculate the mean of each run. The reason we only consider the convolutional models is that they use chunking which seriously affects the time required for the forward pass. We consider audio sequences starting from a length of 10

seconds, and gradually increasing by 10 seconds up to 600 seconds. Each of the convolutional models in the comparison use the mel spectrogram feature.



The figure shows the mean time needed for the forward pass with chunking where the batch size is 1. We can clearly see that the relationship between the length of the audio and the mean time required for the forward pass is linear. The slopes of the curves shown are proportional to the number of parameters for each of the models. As expected, the length of the audio seriously affects the time required for prediction, which would mean that the MobileNet V3 large is a clear winner in this category. All of the measurements were done using a Gigabyte GeForce RTX 3060 Vision OC 12Gb graphics card.

Conclusion and future work

Our approach mainly focused on using spectral features to obtain various spectrograms, and employing various deep learning methods mostly taken from computer vision. This goes hand in hand with the translational invariance of sound signals, however not all aspects of images do translate to sound, most notably sounds are *transparent* and objects in images are not. Nevertheless, this did not hinder the performance of our models.

We've found that smaller models perform better than overly large ones, but this is most likely due to the small amount of training data. Also, we concluded that splitting tasks among the different parts of the network, i.e. Fluffy architecture does indeed help with these sorts of problems. The performance bottleneck is most likely due to the fact that the feature vectors are not general enough, i.e. more subtler parts of the sound, such as the presence of background instruments get overshadowed by leading instruments.

The most powerful and cheap augmentation is certainly waveform adding, it indeed made the model more robust to noise and it did manage to disentangle the spurious correlations among instruments, also it is the only way to force the model to work well in regimes with enormous amounts of instruments.

Furthermore, models can vary but their importance is overshadowed but the data quality. This is visible by the size of the model graveyard, the architecture becomes less important, when leveraged with poor data. The models that work best are the ones that most efficiently work with the data they are given. The most significant improvements to the performance come from adding more data, by simply adding new audio tracks or by more sophisticated means of using pre-trained models. This fact is confirmed by the excellent performance of the AST model in this task.

Some vague notion of perceptual similarity between instruments can be obtained by leveraging the fact that they are grouped in instrument families. Our scheme of penalizing or rewarding models for missing the family is a very simplistic one, and more refinements can be done perhaps in the embedding space of a well trained model. We could, for instance, force the model to group instruments of the same families together more than the ones in different families via a form of metric learning; this is however, difficult to formulate in the multi-label setting.

As for the possible future work, several alterations can be done to improve this project. For instance the pre-training procedure can be improved on, by leveraging masked auto encoding. This is done by masking patches of the spectrogram, and training a encoder-decoder structure to fill the missing patches, once these models are

appropriately trained, the decoder is disposed of and the encoder is used as the pre-trained backbone for a classification head.

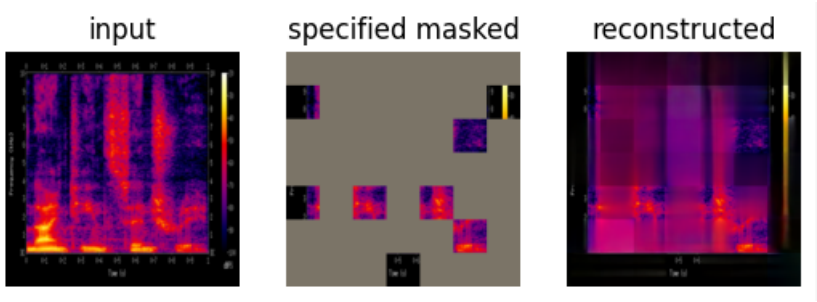


Figure: Masked auto-encoding task generated with Sparse and Hierarchical Masked Modeling (SparK). The network was pre-trained on ImageNet-1K.

Also, different preprocessing procedures can be attempted, for instance we believe that the power normalization is sensitive to sudden maximums, a better normalization that can be explored is loudness normalization.

Lastly, we could further attempt to use an ensemble of learned models for optimal inference accuracy. To be more exact we could leverage the fact that many models were decently trained and using their combined judgment to extract the instruments present. However, this might be a slight overkill for the task in hand.