YouTube Sound Classification

David Burian

October 11, 2024

1 My approach

I was fairly new to audio-processing tasks, and so the first thing that I did was to read up on topics regarding to sound-processing. This included the following topics:

- Fast Fourier Transform, Discrete Fourier Transform, and Discrete Cosine Transform,
- Spectograms, log-spectograms, Mel-spectograms,
- Cepstograms, Mel-cepstograms,
- various other features of sounds like Root Mean Square Energy or Zero Crossing rate,
- just few models such as Wav2vec, Hubert, Whisper, which are probably getting pretty old, but I wanted to have rough idea about the architectures used,
- I refreshed my knowledge of CTC loss.

Even though I didn't end up needing all of the knowledge I gathered, it was fun to learn and explore. After the study session I planed to approach the task as so:

1. Look at the data, and understand them and the type of task. (Section 2)
2. Try easy solutions to the problem. Apply off-the-shelf models from scikit-learn and see what you'll learn. (Section 3.2)
3. Train a neural network to get the best score possible. (Section 3.3)

2 Data Analysis

Relevant notebooks:

- data_analysis.nb.py – for Sections 2.1 and 2.2
- feature_engineering.nb.py – for Section 2.3

2.1 Getting to know the data

As with all other ML projects, I knew that proper data analysis is crucial and I spent good amount of time by looking at the data. First I listened to few examples per label combination. I identified the samples by index in the `tsv` files. These were my findings:

- not a cleanly annotated dataset
  - labels don't include everything that happens
    - e.g.
      - '133' missing instrument,
      - '681' missing wind,

        - '78' missing speech,
        - '124' missing vehicle, music,
        - '753' and '931' missing speech
        - '328' missing speech
  - labels sometimes stand for events that take up minority of the recording
    - e.g.
      - '643' 1s taken up by true label music,instrument, 9s unlabelled speech
      - '931' 8s of unlabelled talking, 2s of labelled tool
      - its not like events happen in the middle of the recording – sometimes they are at the beginning, sometimes at the end

- few mislabellings (extras and errors)
  - e.g.
    - '191' extra animal
    - '623' extra animal
    - '231' baby noise is sometimes interpreted as animal
    - '1523' extra vehicle
    - '1899' rap being recognized as signing
- speech includes multiple languages
- multi-labels are order-invariant
  - sometimes its impossible to say which comes first
  - sometimes they happen at the same time
  - if they are in order, the order doesn't need to correspond to the label order
  - for some combination more than one order exists in the dataset
    - e.g.
      - ''Music,Musical instrument'' and ''Music,Musical instrument''
      - ''Speech', 'Animal'' and ''Animal', 'Speech''
- singing sometimes means just concert noise
- there is a 'silence' label
- not all clips are 10s
  - e.g.
    - '706' 8s
    - '930' 9s
- volume of clips is not the same
  - e.g.
    - '954' a lot quieter than '1304', (same labels)
- duplicates
  - e.g.
    - '3290' ('Animal', 'Speech', 'Music') and '139' ('Speech', 'Music')
    - '7020' ('Music', 'Musical instrument', 'Speech') and (I know I heard it already, I just couldn't find the previous file)

    * ‘3‘ (’Water’,) and ‘111‘ (’Water’,)

- distorted audio
  - e.g.: ‘9753‘

To summarize:

- the dataset is not cleanly annotated: labels are missing or (rarely) are extra
- there are duplicates in the dataset
- labelling is order invariant, meaning the task is not to predict the *sequence* of labels, just their *presence*
- there are some technical gotchas to watch out for:
  - ’Silence’ label exists, meaning we cannot cut-out audio with low volume
  - volume differs, which suggests that I should consider using some kind of normalization
  - not all clips are 10s, but almost all of them are at max 10s long
- the labelling distribution is heavily biased, with some combinations appear only once per dataset

2.2 Creating train and dev split

After getting the ’feel’ for the data, I wanted to split them up to train and development split. This would allow me to analyse the train data more in detail, without worrying about overfitting. I decided against using separate ’test’ split, because it would decrease the amount of data for training and I wouldn’t be able to make further decisions after seeing an evaluation on the test split (since then it would be just another validation split).

2.2.1 Trying to get rid of duplicates

However, before splitting I wanted to take some time to resolve the duplicate audios. Duplicates are bad since they cause information leakage in between splits and they might help with the annotation cleanliness. I started by looking at the duplicate audios I found and compared their signal as in Figure 1.

I tried several approaches to identify the duplicate signals:

1. Naively trying to find shift of one signal such that the difference between the two signals would be minimal.
2. Using off-the-shelf fingerprinting library
3. Applying K-NN to chunks of MFCCs with large windows.

The first approach surprisingly yield large differences even for almost completely overlapping signals. This meant that the decision boundary between “It’s a duplicate” and “It’s not a duplicate” was fairly small. So small I didn’t want to rely on in. For the second approach I tried to use DejaVu[1], but the PyPi package was outdated, installation cumbersome and the project seemed abandoned. So I settled for `ctoth`’s repo[2]. However it failed to recognize even the duplicates that I found, so I got rid of it. The last solution chunked-up MFCCs and fed them to a KNN model. During retrieval, I got audios whose chunks were on average closest to the queried chunks. This worked surprisingly well. I found that a distance below 100, is almost always a duplicate. On the other hand, the smallest distance to non-duplicates, from what I have seen, was around 150. After seeing around 50 query results, I got pretty confident that this technique might be able to successfully identify at least a portion of duplicates. However, I wasn’t totally sold and decided to postpone further de-duplication experiments.

[1] `https://github.com/tuxdna/dejavu`

[2] `https://github.com/ctoth/audioprint`

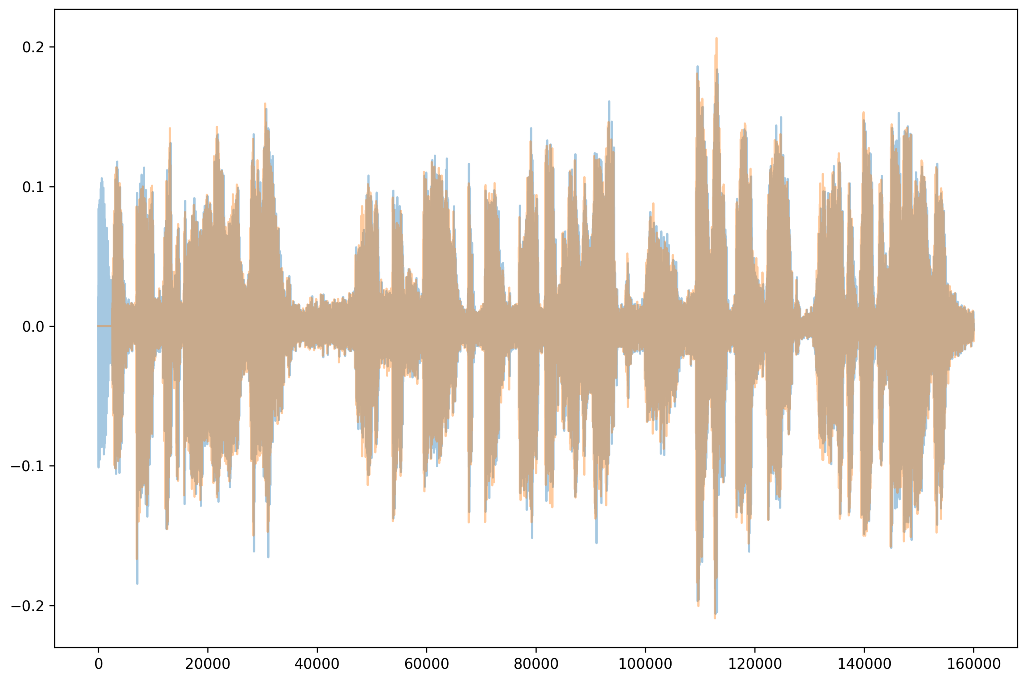


Figure 1: Duplicate signals with ids '3290' and '139'. '139' is shifted by 2550 samples forward.

2.2.2 Actually splitting data to train and dev split

I split the data in 80:20 ratio, according to the distribution of labels, so that both splits reflect the same data distribution. Note, that I didn't split the data according to the samples' label combinations, but rather according to a label's presence. In case one audio got to be in both splits, I removed it from the training split. My reasoning was that, we'd want the detection of different sounds to be independent. So we split the data as if they were independent (even though it might be harder to detect speech during a heavy metal concert).

In the end, I don't think this mattered as much since both distributions – of labels and of label combinations – ended up being very similar, as can be seen in Figure 2. 80:20 ratio gave me around 2k audio samples to validate on, which I think is plenty.

2.3 Closer look at training data

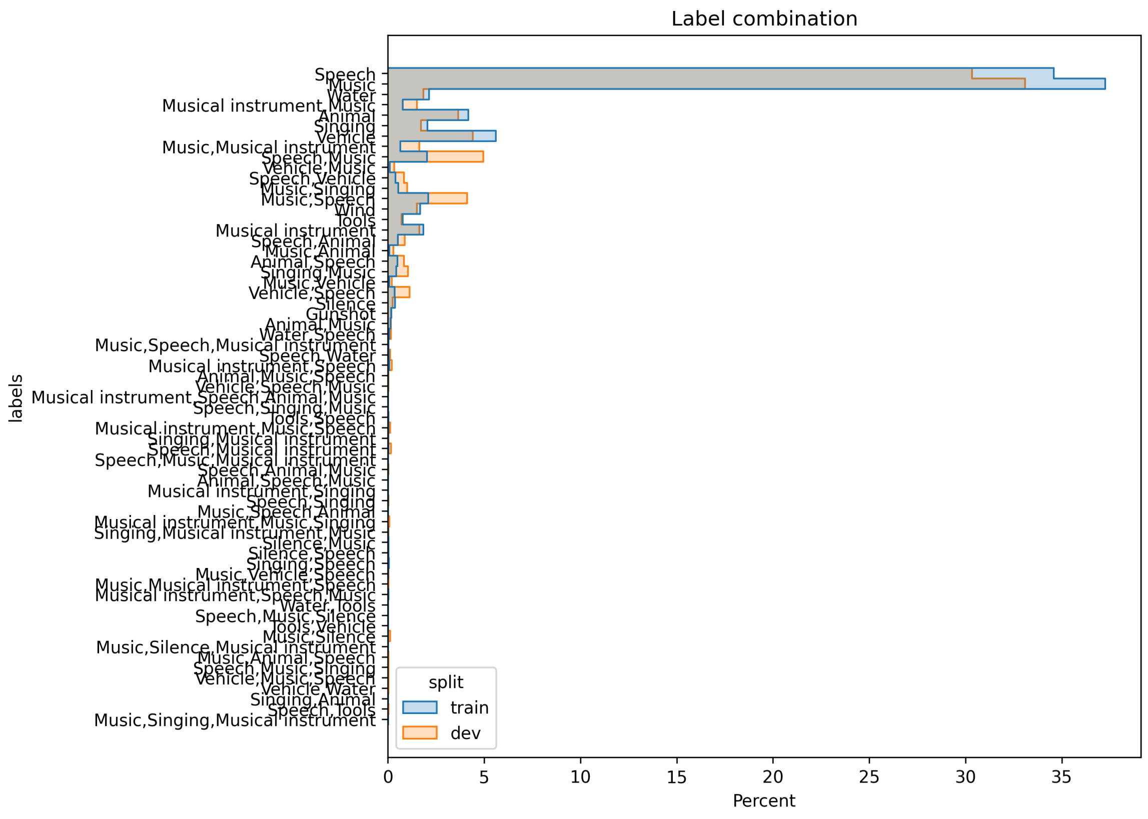
I took a second look at the train data to see the best features. I looked at loudness, RMS, ZCR. All of which are plotted in Figure 3. Due to the label combinations, a lot of the distributions (per label) are similar. However, there are some exceptions:

- 'Silence''s RMS is quite separable from the other labels
- 'Water''s and 'Tools''s ZCRs are also quite separable from the other labels

The two distinctions above could be used to construct an informed classifier. However, as the separation is not great, I decided to postpone this idea, but in the end I didn't have time to come back to it.

2.3.1 Frequency filtering

I also wanted to see if there are some high or low frequencies, which I could filter out without risking filtering out any major features. However, after playing with amplitude thresholds, I discovered the signals span the entire spectrum from 0Hz to 8000 Hz. I could have filtered out some 30Hz on the bottom end, but I don't think that would have mattered.



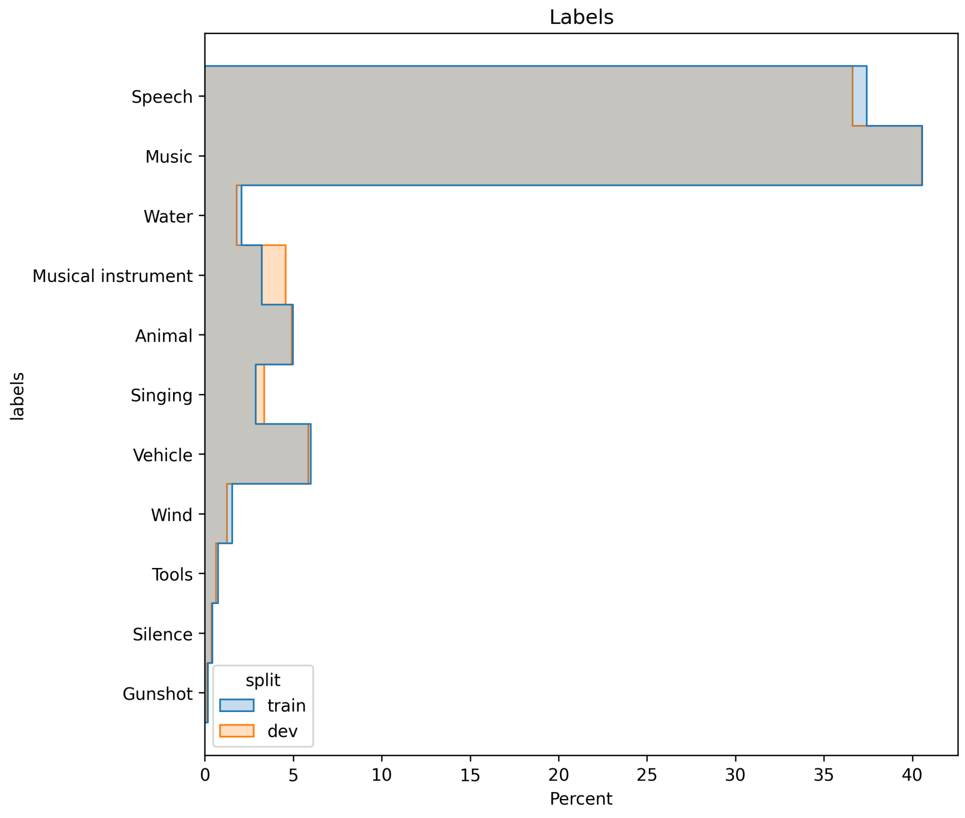


Figure 2: Comparison of label distributions for train and dev splits. Notice that dev split has more label combinations, a bit less of speech and a bit more of musical instruments.

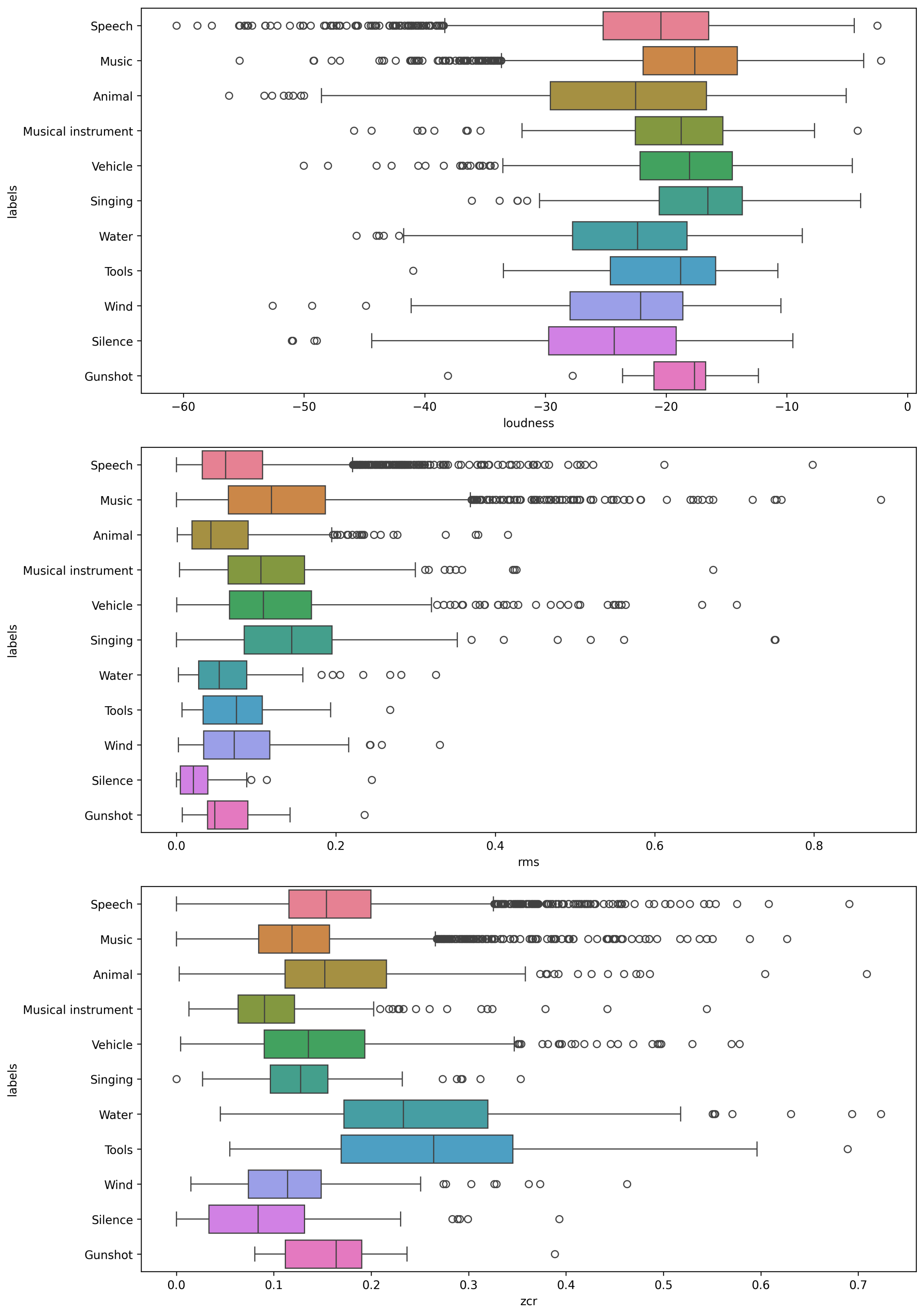


Figure 3: Distribution of Loudness, Root-Mean-Square energy, and Zero Crossing rate of signals in training split.

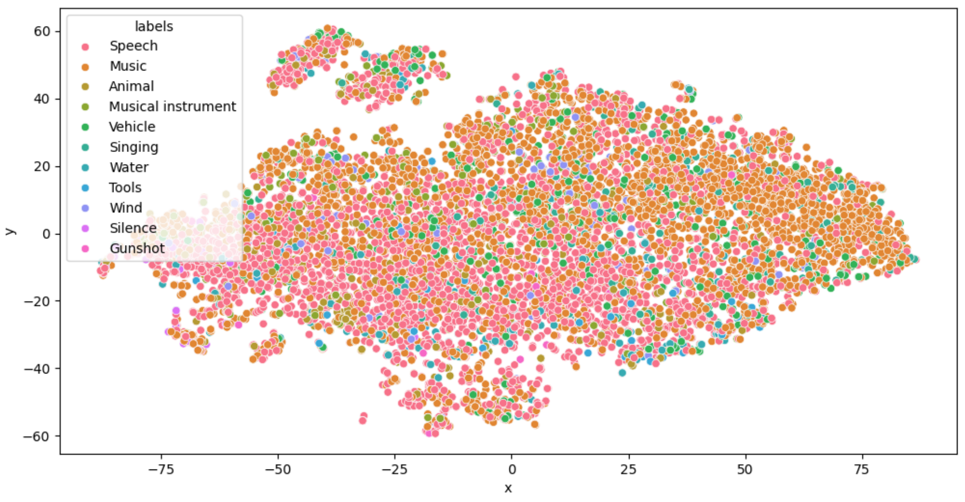


Figure 4: Projection of MFCCs to 2d using PCA to 51 components, than TSNE to 2.

2.3.2 Projecting MFCCs to 2d

This experiment was a bit arbitrary, but I wanted to see if PCA + TSNE could uncover some structure in the space of MFCCs. This was largely inspired by my de-duplication k-NN model, that worked better than expected. I applied PCA to get cover 0.9 variance, and projected these components to 2d using TSNE. However, the projection is a 2d 'blob', which doesn't really tell us anything.

2.4 Looking at MFCCs

Since I wasn't very familiar with MFCCs I wanted to get the 'feeling' for them, similarly as I did with the audio clips. I wasn't sure if MFCCs are better features than just Mel spectrograms, but in the end I experimented almost only with MFCCs. I think, here some extra familiarity with sound processing would pay off. I wasn't sure if the information encoded in the MFCCs is visible enough, and if it isn't, how to highlight it.

2.5 Visualizing transformations

Later, during training of neural networks I experimented with several augmentations. To debug and test the augmentations, I used the rest of the notebook feature_engineering.nb.py.

3 Experiments

3.1 Metrics

Throughout my experiments I kept an eye on precision, recall, f1 and support. I chose the main metric to be macro-averaged f1-score. This simulates a scenario where we want to achieve good performance on all labels and we value TP and TN equally.

3.2 Scikit-learn solutions

Relevant notebooks:

- sklearn_baseline.nb.py – experimenting with sklearn models (not runnable)

I tried 3 sklearn models to fit on MFCCs: K-NN, Random Forest (RF) and SVM. The motivation for these experiments was the successful de-duplication model which was based on similarity of MFCCs of different signals. For all 3 models I repeated the following 3 steps:

1. Overfit on train split
2. CrossValidate on train split to get the best hyperparameters
3. Evaluate the best variant on dev split

All 3 models easily overfitted, however, showed very poor performance on validation split. The performance was so poor, I stopped experimenting with these models and moved on. Later the codebase changed a bit, and so, the notebook currently *cannot be run*, as it relies on interface that no-longer exists.

3.3 Neural network models

Relevant notebooks:

- training.nb.py – running trainings
- cnn_train_testing.nb.py – testing various aspects of training

Relevant modules:

- youtube_asr.train – creating *DataLoader*s and *Trainer*
- youtube_asr.preprocess – data preprocessing functions and augmentation
- youtube_asr.dataset – loading dataset

For a neural network model I followed the standard procedure: overfit on training split with smallest model possible, then regularize to improve validation metrics.

3.3.1 Overfitting on train split

I started by generating fairly limited MFCC:

- 13 MFC coefficients
- 1024 window length
- 256 hop length
- 64 Mel filters

I experimented with several small network architectures before arriving to the following:

- 8d, (3, 7) kernel, (1, 5) stride, ReLU
- 16d, (3, 5) kernel, (1, 3) stride, ReLU
- 4 of ResNet-like small blocks, where each block has:
  - 16d, 3 kernel, 1 stride, 0 padding, ReLU
  - 16d, 3 kernel, 1 stride, 0 padding
  - sum with residual connection
- 4096d, ReLU
- 128d, ReLU
- 11d

I trained with common settings:

- Adam optimizer, learning rate 10^{-4}
- Cross-Entropy Loss

- batch size 8

The above settings were able to overfit on 1 batch in 2.7k steps as seen in log `whimsical_lemming/version_24`.

To describe the following experiments in reasonable amount of space, I will describe only the experiment's differences compared to the previous experiment.

1. `flat_wrasse/version_0`: batch size 16, overfitting on 20% of training split
2. `grumpy_squirrel/version_0`: batch size 32, overfitting on 50% of training split
3. `gainful_vulture/version_0`: overfitting on 80% of training split in 50k steps
    - batch size 64
    - 32 MFCCs
    - Initial convolution has (5, 7) kernel, (3, 5) stride
    - 6 ResNet-like blocks
4. `versatile_bonobo/version_0`: overfitting on 100% of training split in 23k steps
    - transitioned to ResNet-like pre-activation blocks where ReLUs are applied before each convolution

3.3.2 Regularization

To improve validation performance I started to apply regularization. Following the previous setup, these are the experiments I've ran and the changes they introduced:

5. `vague_angelfish/version_0`: adding Batch Norm between every ReLU and convolution
    - didn't dramatically increase `val/f1/macro`
    - it was harder for the model to overfit on train split
6. `large_caracara/version_0`: adding 10^{-4} weight decay for non-bias parameters
    - similar effects as BN
7. `successful_ammonite/version_0`: adding *time_wrap* augmentation
    - 100% probability of applying *time_wrap*
    - Time-wrap got generated once for entire training, which probably made overfitting a bit easier
8. `friendly_mastiff/version_0`: adding *upsample* augmentation
    - 60% probability of applying *time_wrap*
    - Upsampling to even-out label distributions
    - validation f1 score of label's with large upsampling factor went up
    - validation f1 score of label's with smaller upsampling factor went down (as expected)
    - slower overfitting on train split
9. `pumpkin_heron/version_0`: switching to binary cross-entropy
    - I decided to give BCE a try since it much better follows the idea of independent predictions per each label
    - with BCE, the model overfits much more easily
    - but validation metrics didn't plummet, which I would've expected

experiment name (log dir)	val micro f1-score	val macro f1-score
`versatile_bonobo`	0.59	0.338
`successful_ammonite`	0.60	0.32

Table 1: Final results on dev split.

10. `fluffy_poodle/version_0`: adding *rand_frequency_mask* augmentation
    - 50% probability of masking out between 0 and 16 MFCCs for all frames
    - the model had still no problem overfitting
    - no major improvement on validation metrics
11. `affable_caiman/version_0` – more data augmentation
    - masking out up to 24 MFCCs with 0.7 probability
    - masking out up to 80 time frames with 0.7
    - still overfitting very quickly
    - validation scores seem to suffer more
12. `ruby_pronghron/version_0` – dramatically decrease model size
    - only 2 pre-activation ResNet-like blocks
    - only one 256d fully connected layer after convolutions
    - with decreasing model size, validation metrics decreased noticeably
13. `fabulous_bear` – try normalizing MFCCs before applying augmentations
    - dramatically worsened validation scores

3.4 Results

I disclose the best results I was able to achieve in Table 1.

4 Issues I faced & Unresolved problems

Though the dataset isn't large, the task is not straightforward. From my experience, these are the major challenges:

1. Lack of clean validation split. – the missing/extra annotations cause validation split to be less reliable. I can deal with poor quality of annotations in the training split, however, I'd like an evaluation that I can trust.
2. Poor generalization across several models. – I've tested 4 types of models and done over 15 experiments with CNN models. Unfortunately, all models exhibit poor generalization. I don't think that I approached the task wrongly, but I suspect that I miss a piece of the puzzle.
3. Unbalanced label distributions. – With some combinations appearing only once in a dataset, it's clear that problems will appear. I solved this issue by upsampling examples according to their label percentage in the dataset. Along with heavy augmentation, this meant that the model saw approximately the same number of each label with slightly different input.

4.1 Learnings

Along with the problems above I face several technical issues. I was used to machines with TBs of memory, hundreds of CPUs and (almost) unlimited storage. Transitioning to a machine with 16GBs of memory, 100GB of free space and 12 CPUs was a bit of wake-up call to pay more attention to efficient data processing. HuggingFace datasets just don't cut it anymore.

I also acknowledge that I tried maybe too many ideas, spreading my time across several branches of work. This resulted in lack of time and a list of ideas that I haven't had time to go through.