# End-to-end learning for music audio tagging at scale

**Jordi Pons**[1,2], **Oriol Nieto**[2], **Matthew Prockup**[2],
**Erik M. Schmidt**[2], **Andreas F. Ehmann**[2] **and Xavier Serra**[1]
[1]Music Technology Group, Universitat Pompeu Fabra, Barcelona.
[2]Pandora Media Inc., Oakland.
jordi.pons@upf.edu

## Abstract

The lack of data tends to limit the outcomes of deep learning research – specially, when dealing with end-to-end learning stacks processing raw data such as waveforms. In this study we make use of musical labels annotated for 1.2 million tracks. This large amount of data allows us to unrestrictedly explore different front-end paradigms: from assumption-free models – using waveforms as input with very small convolutional filters; to models that rely on domain knowledge – log-mel spectrograms with a convolutional neural network designed to learn temporal and timbral features. Results suggest that while spectrogram-based models surpass their waveform-based counterparts, the difference in performance shrinks as more data are employed.

## 1 Introduction

The music audio tagging task consists in automatically estimating the musical attributes of a song. These attributes may include: moods, language of the lyrics, year of composition, genre(s), instruments, harmony traits, or rhythmic traits. Many approaches have been considered for this task (mostly based on *feature extraction* + *model* [1, 12, 10]), with recent publications showing promising results using deep learning (which enables end-to-end learning pipelines) [4, 2, 9, 6]. In this work we confirm this trend, and we study how different deep learning architectures scale with a large music collection. To this end: we compare our model with a traditional method based on *feature extraction* + *model* [10], and we train our models with one million tracks annotated by musicologists.

We divide deep learning models into two parts: front-end and back-end (see Figure 1). The front-end is the part of the model that processes the input signal in order to map it into a latent-space, and the back-end predicts the output given the representation obtained by the front-end. In the following, we discuss several front- and back-ends.

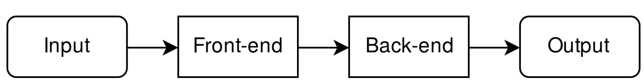


Figure 1: Deep learning pipeline.

**Front-ends.** These are generally conformed by convolutional neural networks (CNNs) [4, 2, 13, 8, 9], since these can learn efficient representations by sharing weights (feature representations) along the signal. Generally, a single filter shape is used in the first CNN layer [4, 2, 5], but some recent work reported performance gains when using several filter shapes in the first layer [13, 8, 9]. Using many filters facilitates leveraging domain knowledge for designing the filters' shape, and also promotes a more rich feature extraction in the first layer – where the signal is available. Further, the design of the filters can be either based on domain knowledge or not. For example, one leverages domain knowledge when a waveform front-end is designed so that the length of the filter is set to be the same as the window length in a STFT [4]. Or for a spectrogram front-end, one can use vertical filters to enforce learning timbral representations [9] or horizontal filters to enforce learning temporal features [8]. On the other hand, when domain knowledge is not used, it is common to use a stack of small filters:

31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.

3x1 (for waveforms [6]) or 3x3 (for pre-processed waveforms formatted in 2D [2]) – similar to a VGG architecture in computer vision. These VGG-like models make minimal assumptions over the local stationarities of the signal, so that any structure can be learnt via hierarchically combining small-context representations. Finally, front-ends can also be divided into two groups depending on the used input signal: waveforms (for end-to-end learning in the strictest sense) [4, 13, 6] or pre-processed waveforms (such as spectrograms) [2, 8, 9].

**Back-ends.** Many back-ends could be used for auto-tagging, and among the different options we identified two main groups: *(i)* fixed-length input back-end, and *(ii)* variable-length input back-end. Interestingly, the generally convolutional nature of the front-end allows to naturally process different input lengths. Therefore, the back-end unit adapts the variable-length feature map to a fixed output-size. Former group of models (*i*) assume that the input of the model will be kept constant – examples of those are fully-convolutional models or DNNs front-ends [4, 2]. The second group (*ii*) can use different input-lengths[1], since the model is flexible in at least one of its input dimensions – examples of those are back-ends using temporal-aggregation strategies such as max-pooling, average-pooling, attention models or recurrent neural networks [11]. Given the variation in song lengths, this last type of back-ends are ideal candidates for music processing.

## 2 Experimental setup

After an initial exploration of the different front-ends discussed above, we selected the two that were performing the best: one for processing waveforms, and another for spectrograms. We aim to study how superior/inferior waveform-based architectures are when compared to spectrogram-based ones given the unprecedented amount of data used for training – 1M tracks for training, 100k for validation, and 100k for test. Experiments below share the same back-end, which enables a fair comparison among different front-ends. Implementation details for the following models and further information about those is available online[2].

**Shared back-end.** It consists of three CNN layers (with 512 filters each and two residual connections), two pooling layers and a dense layer – see Figure 2 (*Bottom-left*). We introduced residual connections in our model to explore very deep architectures, such that we can take advantage of the large data available. Although adding more residual layers did not improve our results, we observed that adding these residual connections stabilized learning while slightly improving performance. The used 1D-CNN filters [4] are computationally efficient and shaped such that all extracted features are considered across a reasonable amount of temporal context (note the $7 \times M'$ filter shapes, representing *time* $\times$ *all features*). We also make a drastic use of temporal pooling: firstly, down-sampling *x2* the temporal dimensionality of the feature maps; and secondly, by making use of a global pooling layer with mean and max statistics. The global pooling strategy allows for variable length inputs to the network and therefore, the proposed model can be classified as a *(ii)* variable-length input back-end. Finally, a dense layer with 500 units connects the pooled features to the output. The study of alternative temporal aggregation strategies (such as recurrent neural networks or attention) is left for future work.

**Waveform front-end.** We make use of the *sample-level* front-end proposed by Lee *et al.* [6] which is composed of a stack of seven CNNs (3x1 filters), batch norm, and max pool layers – see Figure 2 (*Top-left*). Each layer has 64, 64, 64, 128, 128, 128, and 256 filters, respectively. By hierarchically combining small-context representations and making use of max pooling, the *sample-level* front-end yields a feature map for an audio segment of 15 seconds (down-sampled to 16kHz) which is further processed by the previously described back-end.

**Spectrogram front-end.** Firstly, audio segments are converted to log-mel magnitude spectrograms (15 seconds and 90 mel bins) and normalized to have zero mean and unit variance. Secondly, we propose using vertical and horizontal filters explicitly designed to facilitate learning the timbral and temporal patterns present in spectrograms [7, 8, 9]. Note in Figure 2 (*Right*) that the proposed front-end is a single-layer CNN with many filter shapes that are grouped into two branches [7]: *(i)* top branch – timbral features [9]; and *(ii)* lower branch – temporal features [8]. The top branch is designed to capture pitch-invariant timbral features that are occurring at different time-frequency scales in the spectrogram. Pitch invariance is enforced via enabling CNN filters to convolve through the frequency

[1]With a practical constraint: if one wishes to parallelize the computations via batch processing, this requires having signals of the same length within the batch.

[2]`https://github.com/jordipons/music-audio-tagging-at-scale-models`

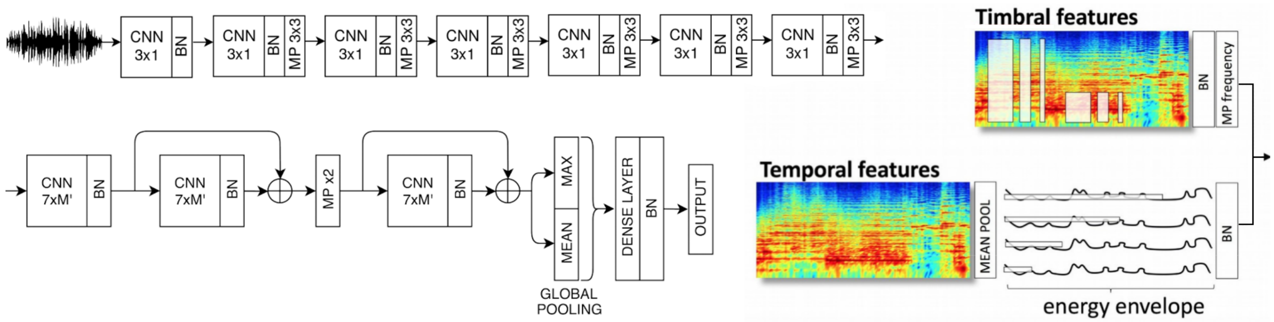


Figure 2: ***Bottom-left*** – back-end. ***Top-left*** – waveform front-end. ***Right*** – spectrogram front-end. ***Definitions*** – *M'* stands for the feature map's vertical axis, *BN* for batch norm and *MP* for max-pool.

domain, and via max-pooling the feature map across its vertical axis [9]. Note that several filter shapes are used to efficiently capture different time-frequency patterns, *e.g.*: kick-drums (with small rectangular filters capturing sub-band information for a short period of time), or string ensemble instruments (with long vertical filters capturing timbral patterns spread in the frequency axis). The lower branch is meant to learn temporal features, designed to efficiently capture different time-scale representations by using several filter shapes [8]. These CNN filters operate over an energy envelope (not directly over the spectrogram) obtained via mean-pooling the frequency-axis of the spectrogram. By computing the energy envelope in that way, we are considering high and low frequencies together while minimizing the computations of the model – note that no frequency/vertical convolutions are performed, only 1D (temporal) convolutions are computed. Therefore, domain knowledge is also providing guidance for minimizing the computational cost of the model. The output of these two branches is merged, and the previously described back-end is used for *going deep*.

**Additional settings.** 50% dropout before every dense layer, ReLUs as non-linearities, and our model is trained with stochastic gradient descent – initial learning rate of 0.001, optimizing the MSE[3] with ADAM and a batch size of 16. During training our data are converted to audio patches of 15 seconds, but during prediction one aims to consider the whole song. To this end, several predictions are computed for a song (by a moving window of 15 sec) and then averaged. Although our model is capable of predicting tags for variable-length inputs, we use fixed length patches since predicting the whole song at once yielded worse results than averaging several 15 second patch predictions. In future work we aim to further study this behavior, to better exploit the whole song during prediction.

## 3 Experimental results

As our baseline we set a system consisting of a music feature extractor (in essence: timbre, rhythm and harmony descriptors), and a model based on gradient boosted trees (GBT) for predicting each of the tags [10]. By predicting each tag individually, one aims to turn a hard problem into multiple (hopefully *simpler*) problems. A careful inspection of our dataset reveals that, among tags, two different data distributions dominate the annotations: *(i)* tags with classifiable bi-modal distributions, where most of the annotations are zero; and *(ii)* tags with pseudo-uniform distributions that can be regressed. An example of a classification tag is any genre; and an example of a regression tag is *'acoustic'* – which indicates how acoustic a song is (from zero to one, *e.g.:* zero being an electronic music song, and one a string quartet). We use two sets of performance measurements: ROC-AUC and PR-AUC for the classification tags, and error ($\sqrt{MSE}$) for the regression tags[4]. Moreover, note that ROC-AUC can lead to over-optimistic scores in cases where data are unbalanced [3]. Given that classification tags are highly unbalanced, we also consider the PR-AUC metric since it is more indicative than ROC-AUC in these cases [3]. We also depict the performance difference ($\Delta$) between spectrogram and waveform models for every measurement. Results are presented in Table 1.

The spectrogram-based model trained with 1M tracks achieves better results than the baseline in every measurement. However, the deep learning models trained with 100k tracks were performing worse than the baseline. This result confirms that deep learning models require large datasets for

[3]We optimize MSE instead of cross-entropy because part of our target tags (annotations) are not bi-modal.

[4]Remember that the model learns to jointly predict classification and regression tags via optimizing the MSE.

clearly outperforming strong (traditional) methods based on feature-design. However, it is important to remark that large datasets are generally not available for most audio tasks. Moreover, the biggest performance improvement *w.r.t.* the baseline is seen for the PR-AUC, which provides a more informative picture of the model's performance when the dataset is unbalanced [3]. And finally, note that the proposed models could be further improved – *e.g.*: one could explicitly address the data imbalance problem during training, or improve the back-end via exploring alternative temporal aggregation strategies.

Furthermore, note that the waveform model trained with 1M tracks also achieves remarkable results. For most measurements it yields better results than the baseline, yet worse than those of the spectrogram model. However, a closer inspection of the results reveals that the differences between spectrogram and waveform models shrink as more data are used for training. Note that such differences ($\Delta$) are halved when models are trained with more data.

Table 1: ***Center*** – performance measurements of the baseline (GBTs+features [10]) and the studied models (waveform and spectrogram front-ends) when considering different training set sizes. ***Right*** – performance difference ($\Delta$) between spectrogram and waveform models for every measurement. For ROC-AUC and PR-AUC, the higher the score the better – but for $\sqrt{MSE}$, the lower the better. Models are of ≈5.5M parameters. The best performing model is highlighted in bold.

| *Models* | *#training examples* | *ROC AUC* | *PR AUC* | $\sqrt{MSE}$ | $\Delta$*ROC AUC* | $\Delta$*PR AUC* | $\Delta\sqrt{MSE}$ |
|---|---|---|---|---|---|---|---|
| GBT+features [10] | 1.2M | 91.61% | 54.27% | 0.1569 | - | - | - |
| Waveform | 1M | 91.54% | 57.86% | 0.1501 | 0.6% | 1.49% | 0.0021 |
| **Spectrogram** | **1M** | **92.14%** | **59.35%** | **0.1480** | | | |
| Waveform | 500k | 91.23% | 56.15% | 0.1537 | 0.54% | 1.75% | 0.0044 |
| Spectrogram | 500k | 91.76% | 57.90% | 0.1493 | | | |
| Waveform | 100k | 89.16% | 49.25% | 0.1591 | 0.97% | 2.83% | 0.0049 |
| Spectrogram | 100k | 90.13% | 52.08% | 0.1542 | | | |

**Qualitative results.** We compared the predictions of our best performing model to the predictions of the baseline, and to the human-annotated ground-truth tags. Two examples are available online for discussion[5]. First, we observed that the proposed deep learning model is biased towards predicting popular tags (such as *'lead vocals'*, *'English'* or *'male vocals'*). Note that this is expected since we are not addressing the data unbalancing issue during training. And second, we observe that the baseline model (which predicts the probability of each tag with an independent GBT model) predicts mutually exclusive tags with high confidence – for example it predicted with high scores: *'East Coast'* and *'West Coast'* for an East Cost rap song, or *'baroque period'* and *'classic period'* for a Bach aria. However, the deep learning model (predicting the probability of all tags at once) was able to better differentiate these similar but mutually exclusive tags: *'East Coast'* score is twice the *'West Coast'* one (0.23 and 0.12); and *'classic period'* is removed from the top 10 while keeping *'baroque period'*. This suggests that deep learning models have an advantage when compared to traditional approaches, because relations between similar but mutually exclusive tags could be encoded within the model.

## 4 Conclusions

The two presented models are based on two conceptually different design principles. The first is based on a waveform front-end, and no domain knowledge inspired its design. The assumptions of this model are reduced to its minimum expression: raw audio is set as input, and the used CNN does minimal assumptions over the structure of the data due to its set of very small filters. For the second model, with a spectrogram front-end, we make use of domain knowledge to guide the design of our model – and our best performing model was designed following this design strategy. The proposed models are capable of outperforming the proposed baseline, and our results confirm that spectrogram-based architectures are still superior to waveform-based models. However, the gap between waveform-based and spectrogram-based models is reduced when training with more data.


## Acknowledgments

This work is partially supported by the Maria de Maeztu Programme (MDM-2015-0502).


[5] `http://www.jordipons.me/apps/music-audio-tagging-at-scale-demo/`

## References


[1] Y. Bayle, P. Hanna, and M. Robine. Revisiting autotagging toward faultless instrumental playlists generation. *arXiv:1706.07613*, 2017.

[2] K. Choi, G. Fazekas, and M. Sandler. Automatic tagging using deep convolutional neural networks. *arXiv:1606.00298*, 2016.

[3] J. Davis and M. Goadrich. The relationship between precision-recall and roc curves. In *ICML*, 2006.

[4] S. Dieleman and B. Schrauwen. End-to-end learning for music audio. In *ICASSP*, 2014.

[5] H. Lee, P. Pham, Y. Largman, and A. Y. Ng. Unsupervised feature learning for audio classification using convolutional deep belief networks. In *NIPS*, 2009.

[6] J. Lee, J. Park, K. L. Kim, and J. Nam. Sample-level deep convolutional neural networks for music auto-tagging using raw waveforms. *arXiv:1703.01789*, 2017.

[7] J. Pons, T. Lidy, and X. Serra. Experimenting with musically motivated convolutional neural networks. In *14th International Workshop on Content-Based Multimedia Indexing (CBMI)*, pages 1–6. IEEE, 2016.

[8] J. Pons and X. Serra. Designing efficient architectures for modeling temporal features with convolutional neural networks. In *ICASSP*, 2017.

[9] J. Pons, O. Slizovskaia, R. Gong, E. Gómez, and X. Serra. Timbre analysis of music audio signals with convolutional neural networks. *arXiv:1703.06697*, 2017.

[10] M. Prockup, A. J. Asman, F. Gouyon, E. M. Schmidt, O. Celma, and Y. E. Kim. Modeling rhythm using tree ensembles and the music genome project. *Machine Learning for Music Discovery Workshop at the 32th International Conference on Machine Learning*, 2015.

[11] C. Raffel. *Learning-Based Methods for Comparing Sequences, with Applications to Audio-to-MIDI Alignment and Matching*. PhD thesis, 2016.

[12] M. Sordo, C. Laurier, and O. Celma. Annotating music collections: How content-based similarity helps to propagate labels. In *ISMIR*, 2007.

[13] Z. Zhu, J. H. Engel, and A. Hannun. Learning multiscale features directly from waveforms. *arXiv:1603.09509*, 2016.