ATIAM 2019 - ML Project
Multimodal embedding music for automatic piece recognition space

Lenny Renault, Hugo Fafin, Bruno Souchu, Arthur Terrasse

January 20, 2020

Abstract

In 2018, Matthias Dorfer and his team brought a method for audio pieces retrieval from a score, or score retrieval from an audio excerpt [1]. This method relies on the use of multimodal convolutional neural networks (m-CNN) and has shown encouraging results, exceeding the context of a synthetic database for training (where audio excerpt files are synthesized from a collection of scores). As part of our ATIAM program, we are proposing to reproduce this method with our own database.[2]

1 Introduction

The field of Music Information Retrieval (MIR) has been expanding widely along with the growing interest in machine learning within the scientific community for the last two decades. The tools provided by the advances in both domains allows for innovative ways to manipulate, analyse and synthesise sound. For instance, one can elegantly categorize music pieces depending on their genre, their structure or their harmonic content. Music transcription is an another fruitful track that research has explored. It consists in linking music (in an audio format) with its symbolic representation (in an score format). Such breakthroughs as valuable in a professional context (help for composition analysis) as in an amateur context (unreferenced song learning for beginners). Building such a link is the main focus of this paper. We will start by discussing the technology and theoretical concepts we employed, we will then present our implementation of the solution proposed by Dorfer et al. [1] and the possibilities it opens.

2 Embedding spaces and CNN

In order to be able to interchangeably go from a sheet representation to an audio point of view and vice versa, we need to find a common space in which we can embed both types of representation. To create such a space, Dorfer et al. [1] have used Convolutional Neural Networks (CNN).

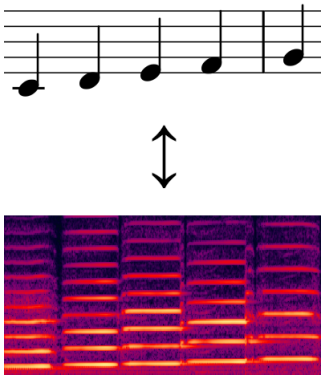

Figure 1: Score sheet and corresponding spectrogram

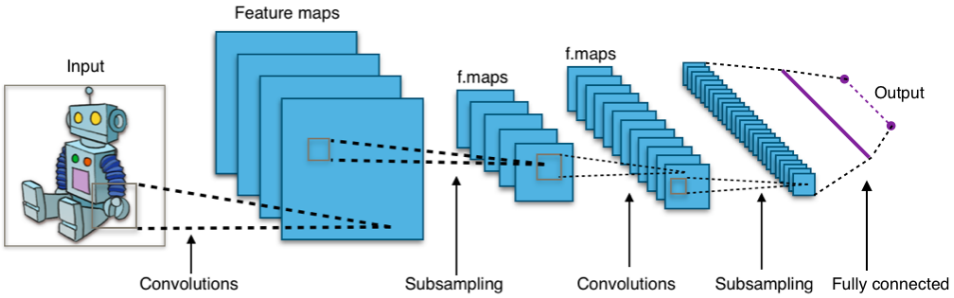

Figure 2: An illustration of a typical CNN

A CNN is a deep learning network specialised in the processing of data with an array-like structure by learning filters to apply to the input [3] in order to extract features from it.
These convolutions are used in conjunction with non-linear activation functions, which can be used to speed up the learning process by introducing non-linearities and pooling layers, which replace a section of an array with a single coefficient, thus reducing the number of variables the network has to deal with.

In our case, Dorfer et al. [1] treat audio spectrograms and MIDI piano rolls as images we can process with this type of network to turn these inputs into embedded vectors of a latent space.
A danger of the deep learning approach is that we have no guarantee that the embedded spaces built by the two networks are the same.
In order to remedy this problem, we will train both networks in parallel with a dataset of audio/MIDI snippet pairs. The intricacies of the training process and dataset building are described in more detail in the next part.

3 Dataset

3.1 Designing the training dataset

In order to make our training dataset, we chose to use the Nottingham dataset, which is comprised of 1035 British and American folk songs in MIDI format. The MIDI files are converted to .wav files using the BatchConverter plugin for the music notation software MuseScore 2.
We obtain stereo files with a 44100 Hz sampling rate, coded on 16 bits.
We then preprocess the files to make them usable with the multimodal network, the preprocessing procedure will be detailed in the following section.

3.2 Preprocessing

The full length MIDI and audio files in the dataset are too complex to be processed by non-recurrent networks, such as the multimodal network described in the next section. We need to cut those file into inputtable short-tensors that are more suitable for the local-analyzing capabilities of convolutionnal networks.

We describe the conversion chain process below, with parameters inspired by the second and third sections of M. Dorfer's paper [1].

Audio preprocessing
- Load with torchaudio
- resample to 22.05kHz
- convert to mono
- spectrogram computation
(92 frequency bins [30 Hz, 6kHz])
- normalization

MIDI preprocessing
- Load with pretty-midi
- stack all instruments into 1 channel
- piano roll computation
(128 note bins)
- conversion to tensor

Note that stacking all instruments into 1 channel strengthens the harmonic information contained in the except, whereas keeping each instrument into different channels would strengthen the melodic fea-

tures. Here, we choose to stack the instruments both for having a constant number of channels over the snippets, and the harmonic information should be more useful for music transcription and orchestration.

The computed spectrograms/pianorolls are then split and stored as short snippets : the paper recommends keeping 42 time bins per audio snippet. We thus wants to also keep 42 time bins per midi snippet, hence we need to use an suitable sampling period when computing the pianoroll.

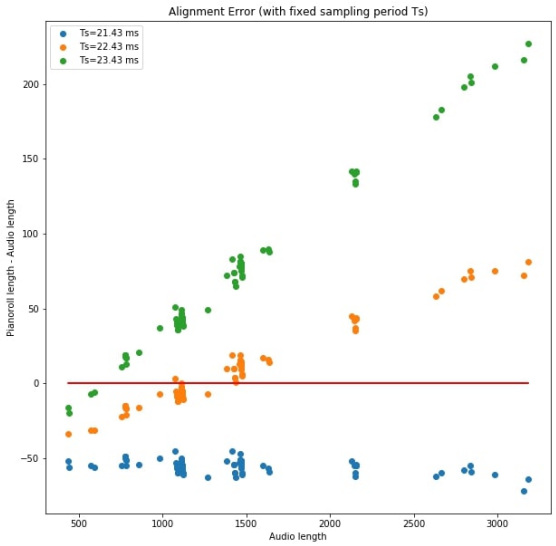

Figure 3: Length difference between audio and matching midi songs, for different pianoroll sampling periods. For 100 songs.

For different sampling periods, we observed the length (in bins) of the rendered piano roll in order to tune it as close as possible to the target audio length bins. With fixed sampling rate, it seems that the length difference follows is proportional with the total length of the music (figure 3). Longer musics tends to have less errors with a small sampling period (around $22ms$) whereas shorter pieces have less errors with longer sampling periods (around $23ms$).

We want the alignment error to be as close to 0 as possible, thus we propose an empirical log-affine function in order to have an adaptive sampling period according to the music length:

$$\begin{aligned} SamplingPeriod &= a \times \log(EndTime) + b \\ &= f_{a,b}(EndTime) \end{aligned}$$

with a and b such as $21.9s = f_{a,b}(150)$ and $23.3s = f_{a,b}(20)$.

The alignment error is shown in figure 4: we managed to keep it between -21 and $+21$ time bins, which means that during the splitting in 42-bins-long snippets, we will only have at worst ± 1 more midi snippets than audio snippets, for a given song.

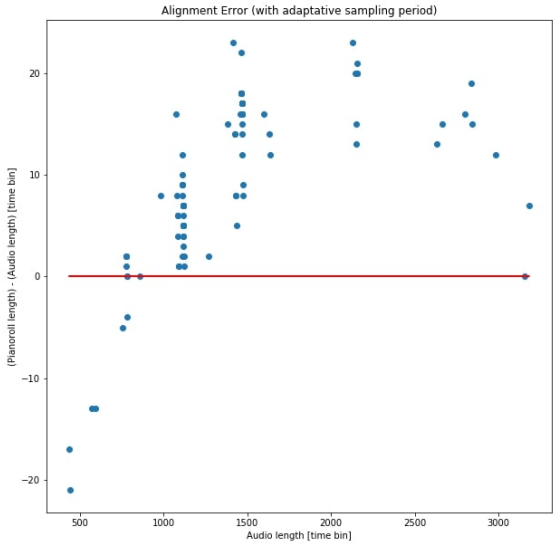

Figure 4: Length difference between audio and matching midi songs, for adaptive pianoroll sampling periods. For 100 songs.

The pre-processed and split excepts are then stored in temporary folders, in wait to be put in a network model.

3.3 Data Augmentation

When loading the snippets, several augmentation processes can be made to reinforce the generalization capacity of the models:

- add noise to the audio excepts.
- render the midi files with different soundfonts.

- temporally stretch the excepts by a factor of ±5%. The except is then cropped or padded in order to keep the 42-bins length.

We cannot apply the vertical translation like Dorfer did for augmenting his sheet images, since this transformation does not change the important information contained in the sheet images (pitch and rhythm) whereas note shifting our pianorolls would completely mess up the pitch information.

The above mentioned transformations have yet to be tested with the training.

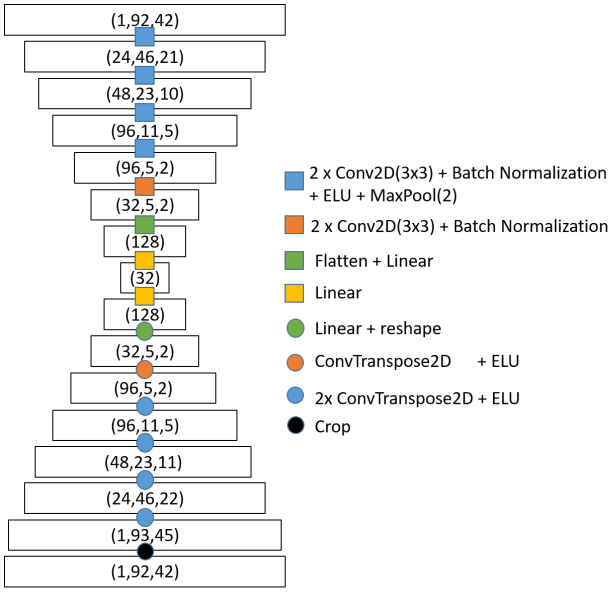

Figure 5: Audio auto-encoder, with tensors sizes around each layers.

4 Models

4.1 Auto-encoders

We first implemented traditional auto-encoding networks for compressing the information contained in the audio and midi snippets into 32-dimensional embedded vectors. The encoders are the two Convolution Neural Network pathways of the multimodal network described in [1].

The number of Multi-Layer-Perceptron layers used to transform the features maps computed by the CNN is chosen according to the output dimension in order to avoid a tight bottle neck. This is the case for the MIDI encoder's Linear layers that has to transform 512-dimension tensors into 32-dimension ones.

We chose to define the decoders symmetrical to the encoders, with 2D transposed convolution layers to revert the transformation of the convolutional and pooling layers. Note that since the dimensions are not all multiple of 2, more than twice the dimensions are lost during some 2-pooling layers. We tune the parameters of the decoder in order to have slightly larger audio and midi reconstruction. We then crop the end to obtain the target dimensions. The network should learn that the cropped values aren't relevant since they are not taken into account in the chosen loss function (Mean-Square Error).

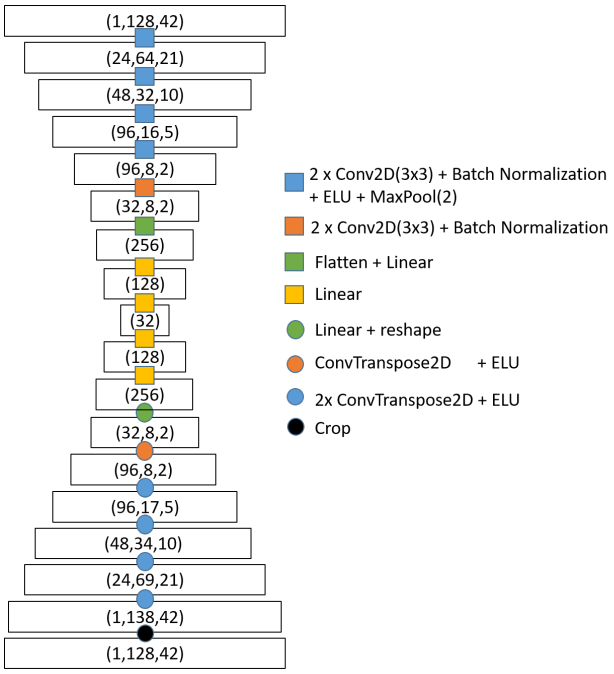

Figure 6: MIDI auto-encoder, with tensors sizes around each layers.

We give a condensed representation of the networks in figures 5 and 6, but complete representations with all layer parameters can be found in the Appendix section (figures 11 and 12).

The auto-encoders have been optimized with an Adam update rule (with initial learning rate of 0.002, like in [1]), over the *Mean-Square Error* loss here, which is better suited for evaluating the re-

construction of an image (pianoroll or spectrogram) with its target.

4.2 Multimodal network training

We define our multimodal network using the audio encoder and MIDI encoder as parallel models with 2 inputs (the midi snippet and a matching audio snippet) and 2 outputs (their respective embedded representations).

For using the acceleration provided with pytorch's *DataLoader*, we define a custom *PairSnippets* dataset class that combines the midi snippets dataset with the audio snippets dataset (grouped by matching labels), which is then split into train-validation-test sets. We use 75% of the given dataset for training, 5% for validation and 20% for testing.

When creating the contrastive audios batch, we draw the excepts from the set being called (train, validation or test set) by randomly picking an audio whose label is not contained in the label batch, until the constrastive batch attains it batch size. Since an audio snippet is drawn until it doesn't belong to the matching batch, we impose the condition that the set size should be at least 3 times the contrastive batch size, in order to avoid loosing too much time generating the contrastive batch with this random picking method. If this condition is not met, we simply run through the set and add an except if it doesn't belong to the matching batch, until the contrastive batch is full, or we have examined all snippets in the set.

The contrastive audio batch for each batch of midi-audio snippet pairs are added for computing the pairwise ranking objective function [4]:

$$\mathcal{L}_{\text{rank}} = \sum_{\mathbf{x}} \sum_{k} \max\left\{0, \alpha - s(\mathbf{x}, \mathbf{y}) + s\left(\mathbf{x}, \mathbf{y}_k\right)\right\}$$

with x the midi snippets, y their matching audios and y_k the contrastive audios snippets. s is the cosine similarity and $\alpha = 0.7$ the loss margin.

The model is then optimized using the Adam update rule, with initial learning rate of 2.10^{-3} and weigth decay of 10^{-5}.

Several training runs were made from scratch and can be monitored with *Tensorboard* using the "valid-runs" folder in our github repository for the log-directory. We display in appendix the results of training the model from scratch, with batch size of 64, 200 epochs, on a Nvidia Titan V GPU. The subset used was all the 'ashover' and 'xmas' songs from the nottingham dataset (59 songs for 2 Go of audio and midi snippets).

It can be seen (figure 10) that the networks starts overfitting beyond epoch 8, we thus keep the model state at the 8th epoch for future network benchmarking.

With *Tensorboard* Projector visualizer , we can observe a 3D-PCA projection of the embedded space. The network still struggles combining the audio snippets with the midi snippets, as a cluster of midi snippets and a cluster of audio snippets can be seen very distinctively (figure 7).

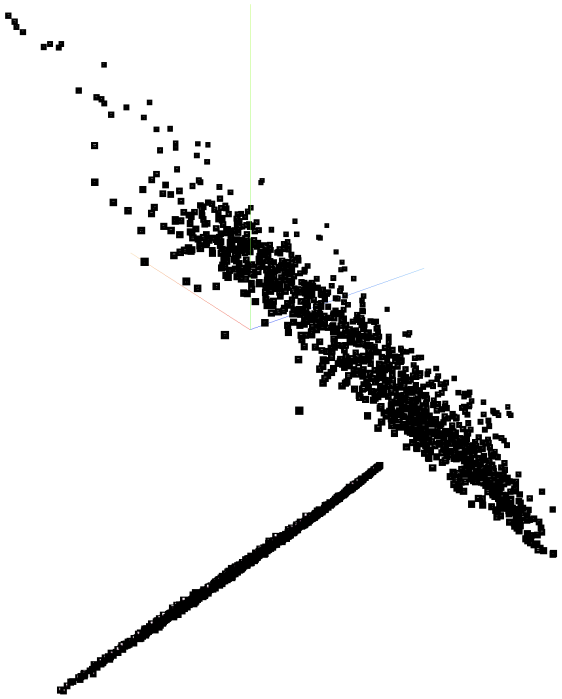

Figure 7: 3D-PCA computed over the embedded representation of the train set, on epoch 8. The MIDI cluster is above whereas the audio cluster is below.

This can be explained by the definition of the loss function that does not specially discourage midi snippets to be far from each other in the latent space, only pushing out the non-matching audio snippets. Also, it has been observed (figure 8) that the snippets contains a lot of information over time (more than 20ms worth of information), the pre-

processing and splitting steps may have unresolved issues.

Figure 8: A MIDI snippet (time bins over the horizontal axis, pitch bins on the vertical axis). The temporal length is a bit too long to be easily matched with its audio counterpart.

5 Evaluation and improvements

5.1 Two-way snippet retrieval

Two-way snippet retrieval consists in pairing an audio snippet to the corresponding MIDI snippet and conversely.[1]
In order to do this, we first need a trained multimodal network, in order to be sure that the latent space is shared between the audio and MIDI part of the network.
To pair the snippet x with its counterpart, we first feed x to its corresponding encoding network, thus associating a point in the latent space to x. We will call it $\hat{x}$ form now on. Knowing $\hat{x}$, we will now encode the dataset in the latent space, again using the corresponding network.
With the latent space set up, we can now identify the snippet corresponding to x by looking for the nearest neighbour of $\hat{x}$ in the latent space, which we call $\hat{y}$. Once $\hat{y}$ is found, we can find the corresponding snippet y by labeling $\hat{y}$ during its projection in the latent space.
In practice, we find the nearest neighbour according to the cosine distance $d(x||y) = 1 - \frac{\underline{x}.\underline{y}}{||x||||y||}$ by concatenating the vectors obtained with the dataset transposed in the latent space and computing said distance between the input snippet and the columns of the resulting matrix. As a result, we end up with a vector of distances labeled by their position in this distance vector. This allows up to sort this vector in ascending order of distance while still keeping track of which distance corresponds to which vector. This allows us to easily retrieve the right vector, as it corresponds to the column of the matrix which index is the label applied to the first element of the vector once it is sorted in ascending order.
Possible improvements and optimization of this process are proposed and discussed in the "Improvements" paragraph of this paper.

5.2 Piece identification and Performance retrieval

A similar process can be used to identify entire pieces. To be able to complete such a task, we first need to label the audio and MIDI snippets with the piece that the previously mentioned snippet comes from. Doing this will allow us to execute the two-way retrieval process for each snippet. We can then use the results of the retrieval for each snippet to "elect" the corresponding piece by finding out which label is attached to the majority of retrieved snippets [1].
We have unfortunately not been able to implement this procedure.

5.3 Improvements

We think that the embedded space construction can be improved using a combination of the following ideas :

- data augmentation: as mention in section 3.3
- pretrain the multimodal network by taking the encoders from the trained audio and midi auto-encoders.
- dissect the files into even smaller snippets in order to capture the instantaneous information and make the retrieval easier.
- enhance the loss function to also draw contrastive midi snippets and widen the distance between them and a matching audio snippets batch.

The two-way snippet retrieval process can be optimized, more specifically concerning searches in the latent space. Fortunately, there are faster alternatives to the simple algorithm we used such as the

EFANNA (Extremely Fast Approximate Nearest Neighbour Algorithm) algorithm [5] which can be up to 300 times faster than a naive implementation of the nearest neighbour search. Despite being faster this algorithm returns an approximation for the nearest neighbour. It is thus prone to error. Fu and Deng found a 5% error rate for EFANNA [5].
The retrieval process, when used in conjunction with dataset augmentation, can also be optimized by labeling the data with the name of the snippet before augmenting it. This would create categories of augmented snippets that can be reduced to a single point in the latent space using an algorithm such as k-Means [6] or clusters of snippets with a border defined by a support vector machine (SVM) for instance.
This would allow extensive dataset augmentation while avoiding an increase in the number of operations needed to carry the retrieval process out.
However, should one consider these options one needs to be aware of the shortcomings of these potential improvements.The k-Means alogrithm might, for example, classify an outlier of one category as a member of an another category should the centroid of the latter group be closer.
SVMs might avoid this pitfall at the cost of runtime and memory storage. In our case however, we found that, given the high number of categories at hand, using a tool as complex as an SVM would be ill-advised.

6 Conclusion

Using MIDI files instead of sheet images for the symbolic part gives more flexibility to our choice of dataset, since we do not need to manually align events with their audio representation. We were able to build a framework capable of constructing and visualizing a multimodal embedded space from a MIDI and audio dataset. However, it lacks some fine tunings in the pre-processing steps in order to allow the multimodal network to be fully capable of joining the two representations. The current state of the project is still unfinished but we still have ideas to test that may improve the performances.

References

[1] Matthias Dorfer, Andreas Arzt, and Gerhard Widmer. Learning audio-sheet music correspondences for score identification and offline alignment, 2017.

[2] Anssi Klapouri and Manuel Davy. *Signal Processing Methods for Music Transcription*. Springer, 2006.

[3] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[4] Lorenzo Rosasco, Ernesto De Vito, Andrea Caponnetto, Michele Piana, and Alessandro Verri. Are loss functions all the same? *Neural computation*, 16:1063–76, 06 2004. doi: 10.1162/089976604773135104.

[5] Cong Fu and Deng Cai. Efanna : An extremely fast approximate nearest neighbor search algorithm based on knn graph, 2016.

[6] J. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Statistics*, pages 281–297, Berkeley, Calif., 1967. University of California Press.

7 Appendix

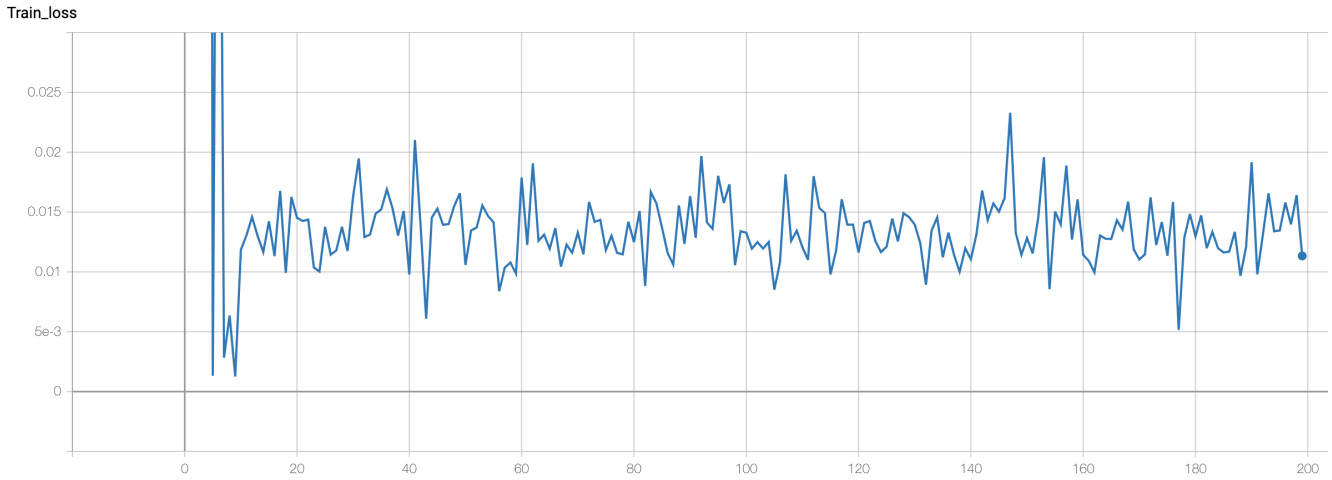

Figure 9: Training loss over the 200 epochs

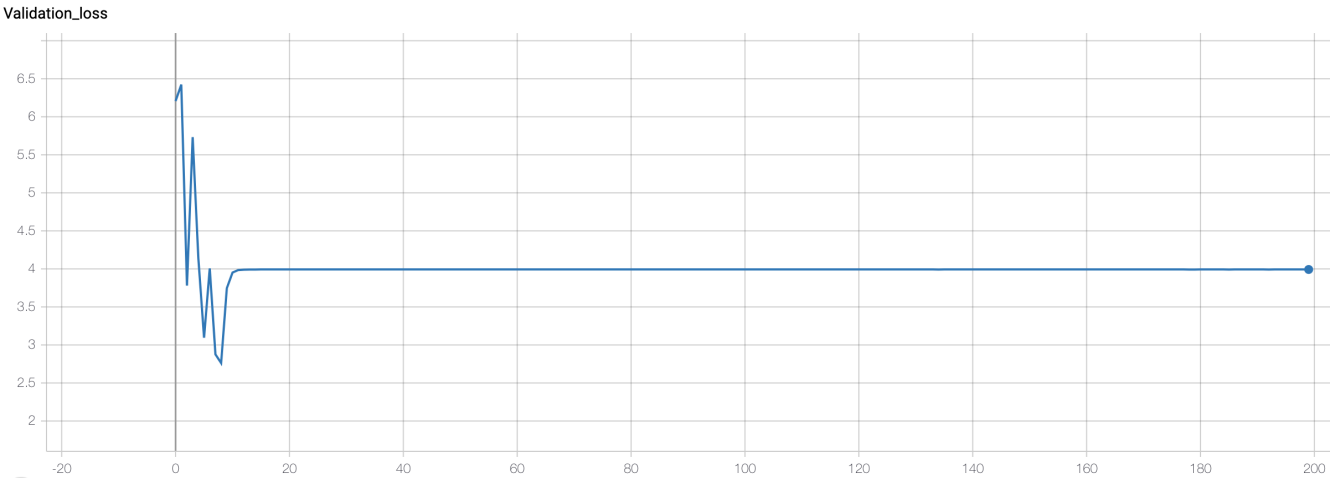

Figure 10: Validation loss over the 200 epochs. The model overfits starting from epoch 9 and onward.

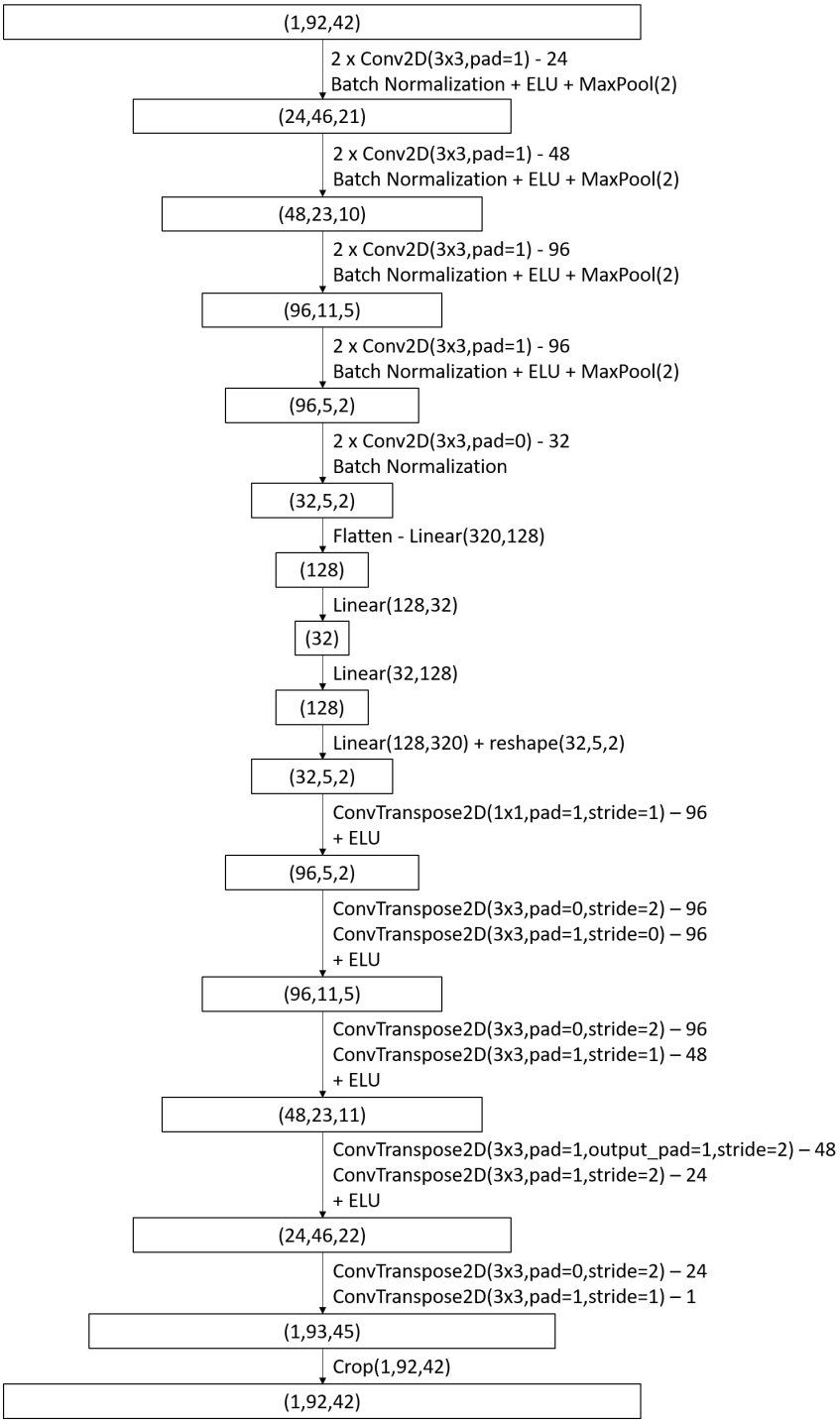

Figure 11: Full Audio Auto-Encoder network, with layer parameters.

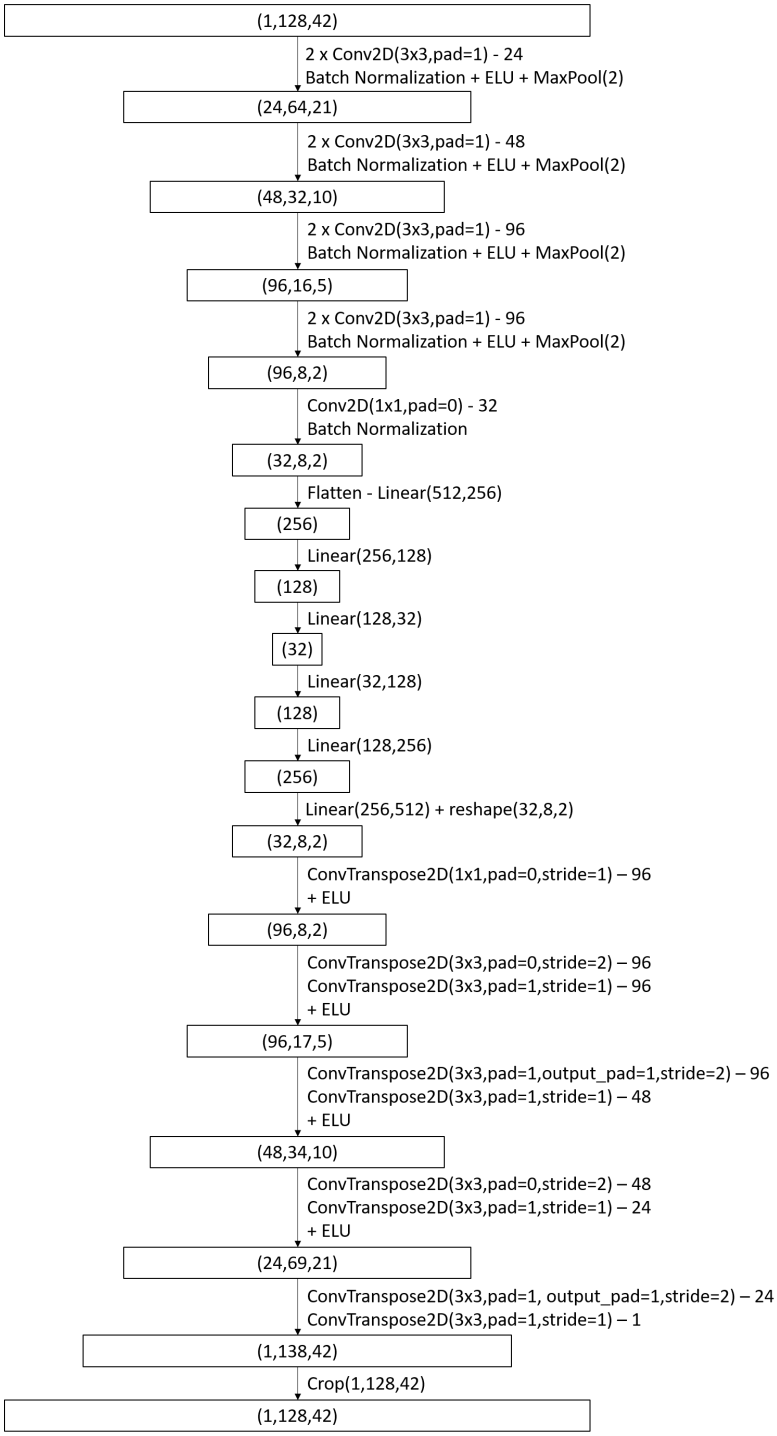

Figure 12: Full MIDI Auto-Encoder network, with layer parameters.