An application of Compressive Transformer for music generation

Giovanni Sorice
g.sorice@studenti.unipi.it

Intelligent Systems for Pattern Recognition (651AA), A.Y. 2019-2020
Date: *18/01/2021*

Abstract

The following document aims to illustrate the work done for the Intelligent Systems for Pattern Recognition course project. I implemented a deep neural network for music generation integrating a variational auto-encoder approach with transformers. Starting from a first idea of the architecture and a first dataset, I tried to emulate the current state of the art models. I gave importance to the comparison of different models trained on different datasets with the aim to understand if one genre or one instrument could generalize in a good way also the other genres and instruments. At the end of the experiment, I noticed how regularization techniques and the amount of available data could impact the results. In the end, good results were achieved. However they are not comparable with the state of art results.

1 Introduction

Starting from the early 1960s with Iannis Xenakis [1], algorithmic compositions were first explored. The initial methods included stochastic composition through a set of probabilities designed by a composer and the use of grammar and rules to specify the style of a given corpus. The task of music generation was not fully satisfied due to methodological and computational shortcomings. Nerveless, they saw that this kind of algorithms could become an excellent tool for assisting musicians. The situation did not change until the neural networks came to the fore in 2012 with AlexNet [2] in the competition ImageNet. From that point on, the use of neural networks has changed and they have also been used in the music generation field. Currently, different models were used like Feedforward neural network, Autoencoder, Restricted Boltzmann Machine, Recurrent Neural Network, Generative Adversarial Network but also Genetic algorithms and Agents based systems.

My goal is to use the compressed transformer model to generate music. I decided to develop this project because I think that the creation and generation of new music is one of the most creative tasks that the human can do. Also, I could explore and better understand a really important and stimulating machine learning model that is the Transformer.

The remaining report is structured as follows. In §2 you can find an explanation of the model used and comments on the result obtained and in §3 there is a brief comment on other approaches, considerations on further improvements and a general comment on the experience.

2 Music Generation

2.1 Lakh MIDI Dataset

The dataset used in this project is part of the Lakh MIDI Dataset [3]. The full dataset is composed of 176 thousand MIDI files divided by artist. I selected two subsets of files from the Lakh MIDI Dataset:

1. 120 MIDI files are randomly and equally divided into three genres of different artist and all containing the piano instrument;
2. 120 MIDI files of only one genre of different artists and all containing the piano instrument;
3. 40 MIDI files are randomly and equally divided into three genres of different artist and all containing the piano instrument (this is a subset of the initial one);

The third dataset is a subset of the dataset n.1 and it was used for the model selection. The total number of MIDI files used in this project is 200. The 200 MIDI files are divided in three genres of different artist and all containing the piano instrument: Metal, Pop Rock and Rock. For the initial model choice, a subset of this 200 MIDI files has been used: 30 MIDI files for the training, 6 for the validation and 3 for the test, all chosen in a randomly and equally way between the 3 genres. From these dataset 3 models were trained, tested and compared:

- A model based on the full dataset n.1;
- A model based on the piano instrument MIDI part of the dataset n.1;
- A model based on the full dataset n.2;

Each model was trained on 85% of the dataset, validated and tested on the remaining 15%. The MIDI representation is a technical standard for controlling and synchronizing digital instrument. This is a really interesting representation because through the use of compatible libraries it is possible to extract the instruments used in the songs.

2.2 Transformer model: vanilla and XL

The architecture used in this project is the compressive transformers [6]. This architecture is an expansion of the Transformer XL [5] and also of the vanilla architecture [4]. The main component of this two architectures are the self-attention and the introduction of more memory in the Transformer XL. The vanilla architecture has an encoder-decoder design as you can see in figure 1. The general data flow is, given a sequence of tokens x they are mapped in a continuous representation z. The decoder takes as input one element of z at a time with the additional input of the previous output generated by itself (the model is auto-regressive).

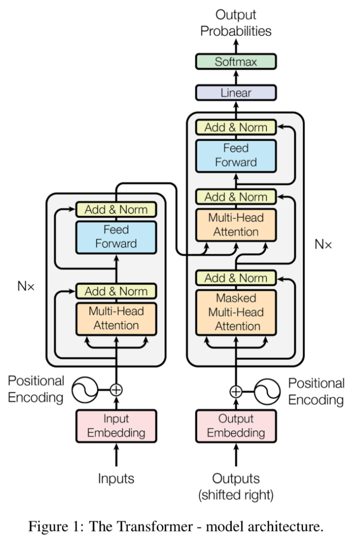


Figure 1: Model architecture of the vanilla transformer.

2.2.1 Vanilla Self-Attention

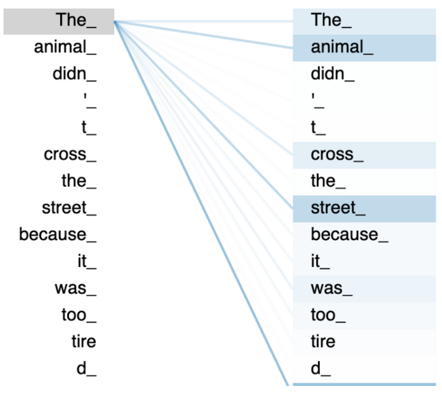


Figure 2: Encoding of the word "The" and measure of the attention paid to other token in the sequence.

The main idea of the self-attention [4] is to learn and describe the relationship between tokens in a sequence. This could be done in different ways (e.g. using the cosine similarity or the scaled dot-product attention equation 1). But, the main observation is to have an attention function that given a vector query Q, the matrices of keys K and values V it computes as output the relationship between them. The input Q and K have dimension d^k.

$$Attention(Q,K,V) = softmax(\frac{Q*K^T}{\sqrt{d^k}})*V \qquad (1)$$

The self-attention can be explained in a clear way using a natural language sequence. As you can see in figure 2, when we encode the first token of the sequence (this happens also for the other token) we can see clearly on which other words the token "The" is focusing. This happens also for the decoding part (in which we had to pay attention not to look at subsequent positions).

2.2.2 Transformer XL new techniques

Two important techniques were introduced by the Transformer XL to overcome the vanilla shortcomings:

- *Recurrence Mechanism*: as said in §2.3.1, this technique enables long-term dependencies by using information from previous segments. This is done conditioning the K and the V value of the current segment with the previously hidden state sequence produced by the previous segment concatenated with the earlier hidden state sequence produced by the current segment. To have a better understanding of the practical computation, in formulas 2, 3 and 4 you can find the mathematical explanation given by the authors of the paper [5].

$$\text{Let } s_\tau = [x_{\tau,1}, ..., x_{\tau,L}]$$
$$\text{be the } \tau\text{-th segment of length L } h_\tau^n \in \Re^{L \times d},$$
$$\text{where n = number of hidden layers and d = hidden dimension}$$
$$\text{Then for segment} s_{\tau+1}, \text{ hidden state of n-th hidden layer is given by:}$$
$$\widetilde{h}_{\tau+1}^{n-1} = [SG(h_\tau^{n-1}) \circ h_{\tau+1}^{n-1}] \tag{2}$$
$$q_{\tau+1}^n, k_{\tau+1}^n, v_{\tau+1}^n = h_{\tau+1}^{n-1} \cdot W_q^\intercal, \widetilde{h}_{\tau+1}^{n-1} \cdot W_k^\intercal, \widetilde{h}_{\tau+1}^{n-1} \cdot W_v^\intercal \tag{3}$$
$$h_{\tau+1}^n = \text{ Transformer-Layer } \left(q_{\tau+1}^n, k_{\tau+1}^n, v_{\tau-1}^n\right) \tag{4}$$
$$\text{where SG}(\cdot) \text{ is stop gradient,}$$
$$[h_u \circ h_v] \text{ is concatenation operation along length dimension}$$
$$\text{and W denotes learnable model parameters}$$

- *Relative Positional Encoding*: the recurrence mechanism introduce a new problem in the positional encoding of the transformer, now the current sequence and the previous one will have the same positional encoding. For this reason, a relative position encoding was introduced to compute correctly the self attention of the current segment. To have a better understanding of the mathematical process, in formulas 5, 6 and 7 you can find the explanation given by the authors of the paper [5].

  In the vanilla Transformer, the attention score between query q_i and key k_j within the same segment is:

$$\begin{aligned}\mathbf{A}_{i,j}^{\text{abs}} &= \text{query}\,(x_i) \cdot \text{key}\,(x_j) \\ &= [\mathbf{W}_q\,(\mathbf{E}_{x_i} + \mathbf{U}_i)]^T\,[\mathbf{W}_k\,(\mathbf{E}_{x_j} + \mathbf{U}_j)] \\ &= (\mathbf{E}_{x_i} + \mathbf{U}_i)^T\,\mathbf{W}_q^T\mathbf{W}_k\,(\mathbf{E}_{x_j} + \mathbf{U}_j)\end{aligned} \tag{5}$$

  After simplification:

$$\begin{aligned}\mathbf{A}_{i,j}^{\text{abs}} &= \underbrace{\mathbf{E}_{x_i}^\top\mathbf{W}_q^\top\mathbf{W}_k\mathbf{E}_{x_j}}_{(\boldsymbol{a})} + \underbrace{\mathbf{E}_{x_i}^\top\mathbf{W}_q^\top\mathbf{W}_k\mathbf{U}_j}_{(b)} \\ &+ \underbrace{\mathbf{U}_i^\top\mathbf{W}_q^\top\mathbf{W}_k\mathbf{E}_{x_j}}_{(c)} + \underbrace{\mathbf{U}_i^\top\mathbf{W}_q^\top\mathbf{W}_k\mathbf{U}_j}_{(d)}\end{aligned} \tag{6}$$

  Authors re-parameterised the 4 terms as:

$$\begin{aligned}\mathbf{A}_{i,j}^{\text{rel}} &= \underbrace{\mathbf{E}_{x_i}^\text{T}\mathbf{W}_q^\top\mathbf{W}_{k,E}\mathbf{E}_{x_j}}_{(a)} + \underbrace{\mathbf{E}_{x_i}^\top\mathbf{W}_q^\top\mathbf{W}_{k,R}\mathbf{R}_{i-j}}_{(b)} \\ &+ \underbrace{u^\top\mathbf{W}_{k,E}\mathbf{E}_{x_j}}_{(c)} + \underbrace{v^\top\mathbf{W}_{k,R}\mathbf{R}_{i-j}}_{(d)}\end{aligned} \tag{7}$$

– Replace all U_j with R_{i-j} as the relative distance matters for where to attend;

– Replace the query $\mathbf{U}_i^\top \mathbf{W}_q^\top$ with $u \in \mathbb{R}^d$ in term (c). $\mathbf{U}_i^\top \mathbf{W}_q^\top$ is the absolute positional encoding for the query. As it is the same for all query positions (we only care about relative key position) we use u. Similarly, we replace term (d) with v;

– Separate the two-weight matrices $W_{k,E}$ and $W_{k,R}$ for producing content-based key vectors and location-based key vectors respectively.

2.3 Compressive Transformers architecture

The compressive transformer architecture is based on the Transformer XL architecture [5] with the addition of compressed memory. As stated in §2.2 the core structure is the same for all the three models (vanilla, XL and compressed). Now, we can understand why and how the memory compression was added and which compression functions and loss were used in the paper [6].

2.3.1 Memory compression

In the Transformer XL [5] was introduced the recurrence mechanism. The goal of the recurrence mechanism is to overcome the vanilla shortcomings enabling long-term dependencies by using information from previous segments. This requires the addition of memory to store the information of the past segments. For obvious reasons, the memory cannot be of infinite size and in this way, we could lose some important information deriving from previous segments. At this point, we could introduce the memory compression to help us store as much information as we can. In particular, through the compressed memory as you can see in figure 3, we can achieve a longer temporal range. Suppose that our Transformer XL has l layers, m as memory size and n as sequence length, the maximum temporal range is $l * m$ with an attention cost of $(n^2 + n * m)$. Suppose having the transformer XL but with memory m divided in m_s as standard memory and m_{cm} as compressed memory, and c as the rate of compression, the maximum temporal range is of $l * (m_s + c * m_{cm})$ with an attention cost of $(n^2 + n * (mm_s + m_{cm}))$.

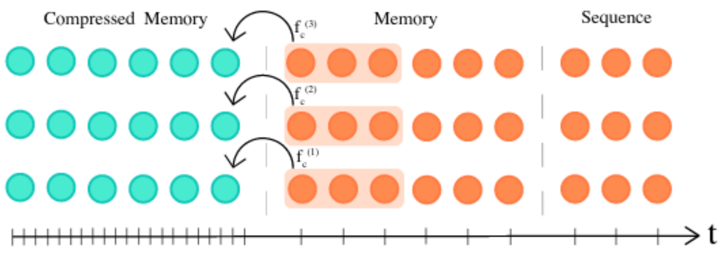


Figure 3: Example of compressed memory in a transformer with 3 layer, a sequence length of 3, a memory size of 6, a compressed memory size of 6 and rate of compression of 3.

2.3.2 Compression functions and loss

The compression function could be chosen between a wide range of function, in the original paper of the compressive transformers [6] four different functions were considered:

- max/mean pooling (kernel and stride set to c);
- 1D convolution (kernel and stride set to c, take into consideration only its layer information);
- dilated convolutions (reducing the space taking also the spatial correlation between information);

- most-used (where only most used memory was preserved).

In my opinion, the authors of the paper [6] decided to explore this compression function because they are at the same time simple to use and fast to compute.

The loss function used for the Transformer task objective is the cross-entropy loss. The most performing loss for the compression network was the attention-reconstruction loss (algorithm 1) because our goal is to reconstruct the original memory. Also, all the information that are no longer attended can be discarded.

Algorithm 1 Attention-Reconstruction Loss.

```
1: $L^{attn} \leftarrow 0$
2: for layer i = 1, 2,..., l do
3:     $h^{(i)} \leftarrow$ stop_gradient($h^{(i)}$)
4:     $old\_mem^{(i)} \leftarrow$ stop_gradient($old\_mem^{(i)}$)
5:     Q, K, V ← stop_gradient(attenstion params at layer i)
6:     def attn(h,m) ← $\sigma$((h·Q)·(m·K))·(m·V)
7:     $new\_cm^{(i)} \leftarrow f_c^i(old\_mem^{(i)})$
8:     $L^{attn} \leftarrow L^{attn} + \|$attn($h^{(i)}, old\_mem^{(i)}$) - attn($h^{(i)}, new\_cm^{(i)}$)$\|_2$
9: end for
```

2.4 Experiments

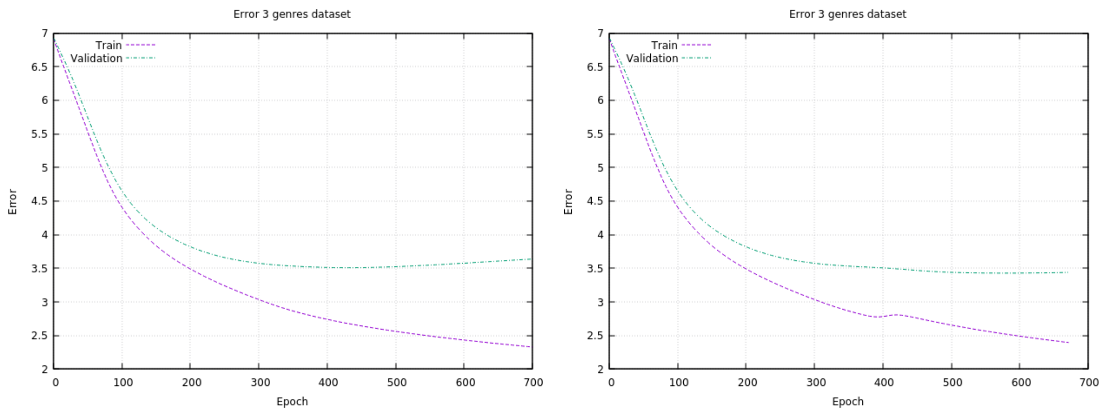
The architecture is based on the implementation of Phil Wang [7] of the compressive transformers. Now I am going to describe the exact architecture used for all three datasets. The input consists of a vector of 64 notes and chords. Then, there are the encoder and the decoder, for a total of 6 residual layers. All the models were trained with Adam PyTorch Optimizer. Different configurations of the architecture were validated, the first important parameter searched was the sequence length. Three sizes were taken into consideration: 32, 64 and 128. After some considerations about the pros and cons of each configuration, like information understanding and correlation, costs and time for training and probability of overfitting or underfitting, I chose to use 64 as sequence length. Also, the regularization is important in this kind of tasks with models with a large number of parameter. For this reason, I tried two types of models, one without regularization and one with 0.1 of dropout parameter.

2.5 Training

The training was computed on Google Colab (with Nvidia k80, T4 and P100 gpu). Each model was trained at least for 100 epochs. Each epoch had a time to compute in the range of 200 to 500 seconds for dataset 1 and 2 and of in the range of 60 to 80 seconds for dataset 3. A summary of the execution results is available in the table 1. Interesting behaviors can also be observed in figures 4, 5, 6 and 7. First of all, in all the plots are visible the positive effects of the regularization (only the dropout regularization was tested because it was the only type available in the library). As expected from the theory, the models trained with dropout regularizer decreases the variance with respect to models without. Also, the models trained with dataset 1 and 2 shows that a large dataset decreases the overfitting risk. Following that, it is possible to observe that the models trained with piano 1 and 3 show overfitting in less than 50 epoch. In conclusion, in figure 4 and 6 it is possible to see that the adam optimizer changed the approached local minima and I think that this is due to the regularization.

Dataset	Epoch	Best epoch	Dropout	Error TR	Error VL
1	700	435	0	2.6694	3.5057
1	671	570	0.1	2.5349	3.4274
Piano 1	104	40	0	2.5028	4.1092
Piano 1	148	105	0.1	2.6281	4.0401
2	228	135	0	2.7016	3.2504
2	525	400	0.1	2.7504	3.0382
3	255	30	0	2.3218	4.3809
3	187	25	0.1	2.5188	4.3612

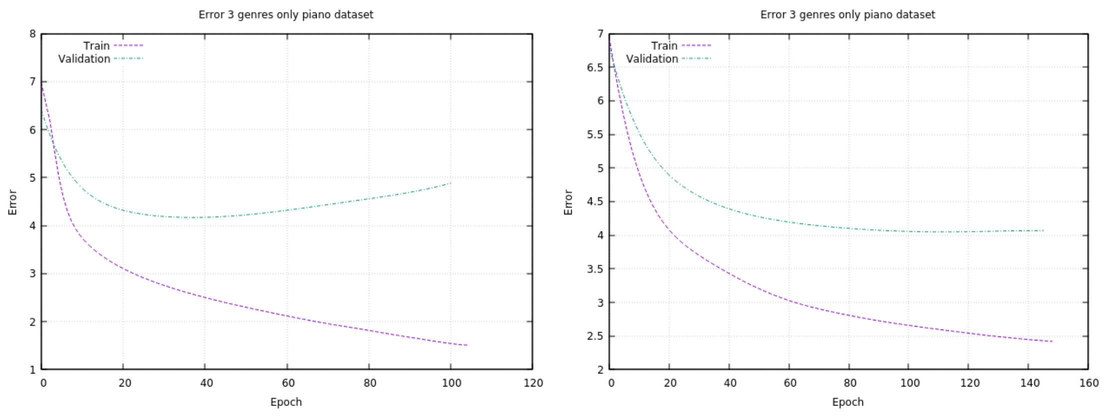
Table 1: Best epoch training and validation error value.



(a) Without dropout.

(b) With 0.1 of dropout.

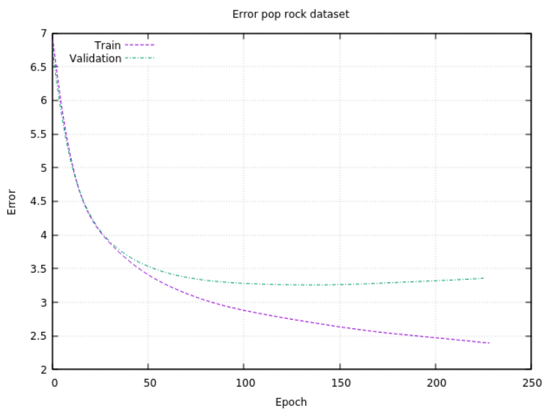
Figure 4: Learning curves with dataset 1.



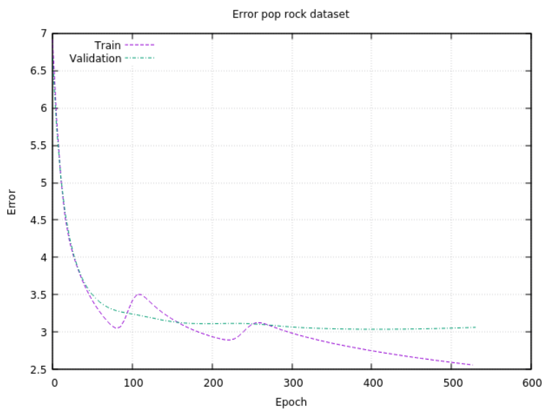
(a) Without dropout.

(b) With 0.1 of dropout.

Figure 5: Learning curves with only notes and chords containing piano instrument of dataset 1.

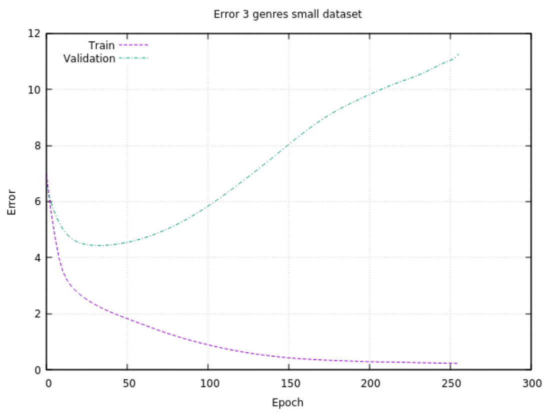


(a) Without dropout.

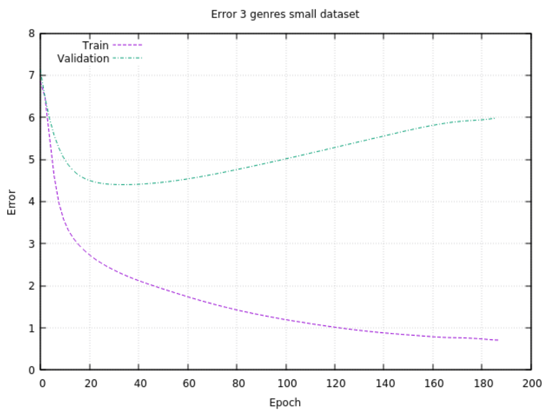


(b) With 0.1 of dropout.

Figure 6: Learning curves with dataset 2.



(a) Without dropout.



(b) With 0.1 of dropout.

Figure 7: Learning curves with dataset 3.

2.6 Model comparison

All the models presented in §2.5 were tested on a dataset of 8 songs of 3 different genres and all containing the instrument piano. The results are presented in table 2:

Dataset	Dropout	Error TR	Error VL	Error TS
1	0	2.6694	3.5057	4.3238
1	0.1	2.5349	3.4274	4.2746
Piano 1	0	2.5028	4.1092	4.8609
Piano 1	0.1	2.6281	4.0401	4.7762
2	0	2.7016	3.2504	4.4305
2	0.1	2.7504	3.0382	4.1874
3	0	2.3218	4.3809	5.8341
3	0.1	2.5188	4.3612	5.4717

Table 2: Best epoch training and validation error value.

Using as baseline the models trained on full dataset 1, It is possible to do some comparisons. First, from table 2, it is possible to observe that the model trained only on Piano instrument, does not perform as the others. From this, I can conclude that it is better to consider possibly all the instruments used at each time. Surprisingly, the model trained only on one genre performed better than the baseline, I think that it is due to the similarity of the three chosen genres. Finally, the model trained on the small dataset obtained the worst value but it was presumably due to the higher probability of overfitting. In conclusion, the overall error on the training set is higher with respect to the state of the art result and I will discuss some improvement in the §3.

3 Conclusion and further approaches

Music generation is an old and discussed argument and for this reason, different architectures can be used to solve this task. Different examples can be found in the paper [8], like LSTM, BI-LSTM and others. Currently, the most performing and promising architectures are the transformer-based due to the attention mechanism.

The current state of the art architectures for the transformer model introduce memory efficiency usage, reversible transformers and try to take in consideration the melody of the songs ([9], [10]). In [9], the memory is exploited using LSH techniques combined with attention function. From the idea of this paper, I decided to exploit the memory usage with compressive transformers. A possible improvement that could be done is the encoding of more than one instrument for each note with also the temporal parameter set. In this way, during the generation of the music, more than one instrument could be associated with each note or chord generated making music more listenable and harmonious.

References

[1] Iannis Xenakis
Formalized Music: Thought and Mathematics in Composition, Indiana University Press, 1963.

[2] A. Krizhevsky, I. Sutskever, and G. E. Hinton.
ImageNet classification with deep convolutional neural networks, Advances in Neural Information Processing Systems, NIPS, 2012.
`http://www.cs.toronto.edu/ fritz/absps/imagenet.pdf`

[3] Colin Raffel *The Lakh MIDI Dataset v0.1.*
"Learning-Based Methods for Comparing Sequences, with Applications to Audio-to-MIDI Alignment and Matching". PhD Thesis, 2016.
`https://colinraffel.com/projects/lmd/`

[4] A. Vaswan et al.
Attention Is All You Need, NIPS 2017.
`https://arxiv.org/pdf/1706.03762.pdf`

[5] Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc V. Le, Ruslan Salakhutdinov
Transformer-XL: Attentive Language Models Beyond a Fixed-Length Context, arXiv 2019.
`https://arxiv.org/pdf/1901.02860.pdf`

[6] Rae, Jack W and Potapenko, Anna and Jayakumar, Siddhant M andHillier, Chloe and Lillicrap, Timothy P
Compressive Transformers for Long-Range Sequence Modelling, arXiv 2019.
`https://arxiv.org/abs/1911.05507`

[7] Phil Wang
Compressive Transformer in Pytorch.
`https://github.com/lucidrains/compressive-transformer-pytorch`

[8] Jean-Pierre Briot,1, Gaëtan Hadjeres†and François-David Pachet
Deep Learning Techniques for Music Generation– A Survey.
`https://arxiv.org/pdf/1709.01620.pdf`

[9] Nikita Kitaev, Łukasz Kaiser, Anselm Levskaya.
The Efficient Transformer.
`https://arxiv.org/pdf/1709.01620.pdf`

[10] Kristy Choi, Curtis Hawthorne, Ian Simon, Monica Dinculescu, Jesse Engel.
Encoding Musical Style with Transformer Autoencoders.
`https://arxiv.org/pdf/1912.05537.pdf`