A Casual Creator for Music Style Interpolation

Corey Ford

Queen Mary University of London, UK
c.j.ford@qmul.ac.uk

1 Introduction

Generative music has driven the design of many Casual Creators (CC) [3], where creating music is fun. For this project, a CC design was created where people can explore excerpts of music in two different styles: reggae and baroque. These excerpts are then broken into melodies and randomly selected to create a musical interpolation, allowing people to quickly create interesting musical pieces.

This report is organised as follows: i) background is explained on CCs and generative music, ii) the implementation of our system is detailed, iii) results of experiments are offered, and iv) the project is discussed with respect to CC.

2 Background

2.1 Casual Creators

Designing interfaces which support creativity is an active area of study. Indeed, Creativity Support Tools (CSTs) are a genre of software designed to facilitate people's motivation and development of expertise [17]. However, whereas CSTs focus on professional creativity, Casual Creators (CCs) purely focus on autotelic creativity, where emphasis is placed on people creating artefacts quickly and in a way that brings them feelings of pleasure and joy [3]. To guide the design of CCs, Compton and Mateas [3] present some design patterns. For this project, the following key design patterns are explored:

- **P1: No blank canvas**
  Provide direction for peoples first move.
- **P2: Limiting actions**
  Only provide appropriate choices with clear consequences.
- **P3: Modifying the meaningful**
  Allow people to change high-level parameters that make clear changes.

The following guiding principle is also drawn from Petrovskaya *et al.'s* [14] exploration of CCs"in-the-wild":

- **P4: One-touch**
  People interact with the application using one finger.

2.2 Time-Aware Neural Networks

As music generation projects require algorithms for time-dependent data, this section briefly introduces some relevant architectures. How these are combined to create useful music generation architectures is discussed in Section 2.3.

Standard feed-forward neural network (NN) architectures have two main limitations: i) they can be difficult to train when input vectors are large (e.g entire music or text sequences), and ii) have limited memory, forgetting data outside of windowed sequences. Recurrent NNs (RNNs) provide a solution as they're able to handle sequences of any length, retaining information via hidden states (which retain information from previous time steps) [10].

However, as RNN's derive gradients based on previous calculations during backpropagation, they can be suspect to the vanishing gradient problem — multiplying values repeatedly by a small number drags them to near zero. A solution is to use Long-Short Term Memory Networks (LSTMs), which extend RNNs to either forget, update or output information, via vector masks and normalising functions. Gated Recurrent Units (GRUs) are another solution, similar to LSTMs, but only reset or update information[1] [15].

Bi-directional LSTMs are often used in music generation [13,16,9,2]. They split data into a backward and forward layer and concatenate the output such that both past and future information are preserved (looking ahead and behind). Indeed, this is useful in domains like text or music generation where word/character/ note choice is based on surrounding contexts.

2.3 Music Generation

Music data can be formatted for NNs in many ways e.g. piano-roll or token format [2]. In this project, the latter approach is used, drawing inspiration from Fradet *et al.* [8], who presents a python package for encoding MIDI (a standardized representation for music data) into various tokens. The package also supports solutions to limitations of MIDI encoding, such as adding chord changes or tokens for rests [8,2].

One approach to generating tokenized music is to follow an iterative feed-forward approach [2]: i) select a seed (e.g. a note), ii) feed-forward this into a RNN to produce the next item, iii) repeat until a sequence of the required length is produced. Indeed, Sturm *et al.* [18] were successful in generating Celtic music melodies using a novel tokenizing encoding to train 3 stacked LSTMs (with 512 cells) via a feed-forward approach. Eck and Schmidhuber [6] also used a feed-forward approach successfully to generate blues music.

One problem with models trained using an iterative feed-forward approach is that they are often constrained to a specific genre [2]. One solution is a Variational Autoencoder (VAE). A VAE consists of an encoder, which learns a compressed representation of a

[1] Note that LSTMs and GRUs were eventually surpassed by transformers like GPT-2; discussion on this is beyond the scope of this project.

dataset (or *latent vector*), and decoder, which learns a mapping from the latent vector back to the original output [12]. The model is unsupervised, typically trained on a KL Divergence loss. Distinct from an autoencoder, a *variational* autoencoder learns a latent vector representing a probabilistic distribution and can be continuously explored (no gaps between data points) (ibid). Reccurent VAEs make use of RNN models for encoding and decoding [7]. Fabius and van Amersfoort [7] used a VAE to capture different styles of video game music. Roberts *et al.* [16] developed a large VAE model for music, named MusicVAE, incorporating a novel decoder to capture long-term structures. For this project, of most importance is that their model is open-sourced, high-quality, and capable of outputting novel interpolations in the latent space.

3 Implementation

Influenced by CCs [3], a mock GUI interface was created where people can search for pre-generated outputs in two genres — reggae and Bach chorales — based on a set of musical metrics, and create a musical interpolation. Here implementation details are given for the following pipeline: gathering and preparing data >> training models to generate extracts >> calculating musical metrics for extracts >> creating the GUI and music interpolations.

3.1 Gathering & Preparing Data

Two datasets were gathered: i) reggae midi music[2] *(n=108)* and ii) Bach chorales[3] *(n=31)*. As our NN has no notion of instruments, the reggae midi data was manipulated using the music21 python package [4] to remove all percussion tracks (resultant generations where thus harmonic but not constrained only to melody lines, i.e., still contained signature reggae guitar offbeat rhythms). For both styles, each MIDI file was converted to tokens and concatenated. The REMI encoding was used, provided by the Miditok package [8].

3.2 Training Model to Generate Extracts

The data was split into target and training data, using a length of 32, where the target data predicted the next sequence of tokens, resulting in 7354 and 4366 instances for the reggae model and Bach model respectively. Batches were made from the data, with batch size = 64, shuffling the data with buffer size = 10000.

Four different model architectures were tested:

- M1: 1 GRU with 512 recurrent units.
- M2: 1 LSTM with 512 recurrent units.
- M3: 1 Bi-Directional LSTM with 512 recurrent units (inspired by [13,16,9])

[2] `http://midi.dubroom.org/`
[3] `http://www.jsbach.net/midi/`

– M4: 3 stacked LSTM's each with 512 recurrent units (inspired by [18])

The NNs above were preceded by an embedding layer with 256 dimensions (to inflate the vectors) and followed by a dense layer (of the vocab size). Categorical cross entropy loss was used. The Adam [11] optimizer was used with learning rate = 0.005 and $\beta 1$ = 0.5, training the model for 150 epochs. To generate musical output, 250 tokens were fed-forward, starting with the input seed 128.

3.3 Musical Metrics

The procedure in Section 3.2 was followed to output 100 extracts, for both the Bach and reggae datasets, using architecture M4 (see Section 4). Music21 [4] was then used to calculate the following musical metrics for each extract, inspired by Banar and Colton [1]:

– Pitch Count = the number of unique pitches used atleast once

– Pitch Range = the difference between the highest and lowest pitches

– Average Note Duration = the mean duration of notes

– Stepwise Motion = the fraction of melodic intervals corresponding to a minor or major second

The tokens and metrics were stored in a CSV file, and loaded as a database for the graphical user interface (GUI). A hard-coded example is given in the colab notebook submission, so that examiners don't need to run the entire generative process.

3.4 Graphical User Interface with Interpolation

The GUI uses colab's form tools, exposing sliders for both genres of musical extracts, for each of the musical metrics (see Figure 1 and Section 3.3). The sliders are used to search our database, finding the closest matching values by i) subtracting the slider value from the actual values ii) passing the values through an $abs()$ function, and iii) finding the smallest n values, narrowing the search down each time to a single database item until each metric has been accounted for and only a single result remains.

Once the extracts have been narrowed down (one for each genre), the tokens were converted to MIDI and reloaded, as a note sequence object, to match Google Magenta's VAE library. Time signatures and instrument information were then removed from the note sequences, such that Magenta is able to processes the notes, extracting melodies that can be input into their MusicVAE model. Indeed, one of the extracted melodies from both note sequences is randomly parsed into a MusicVAE model, which produces the interpolation.

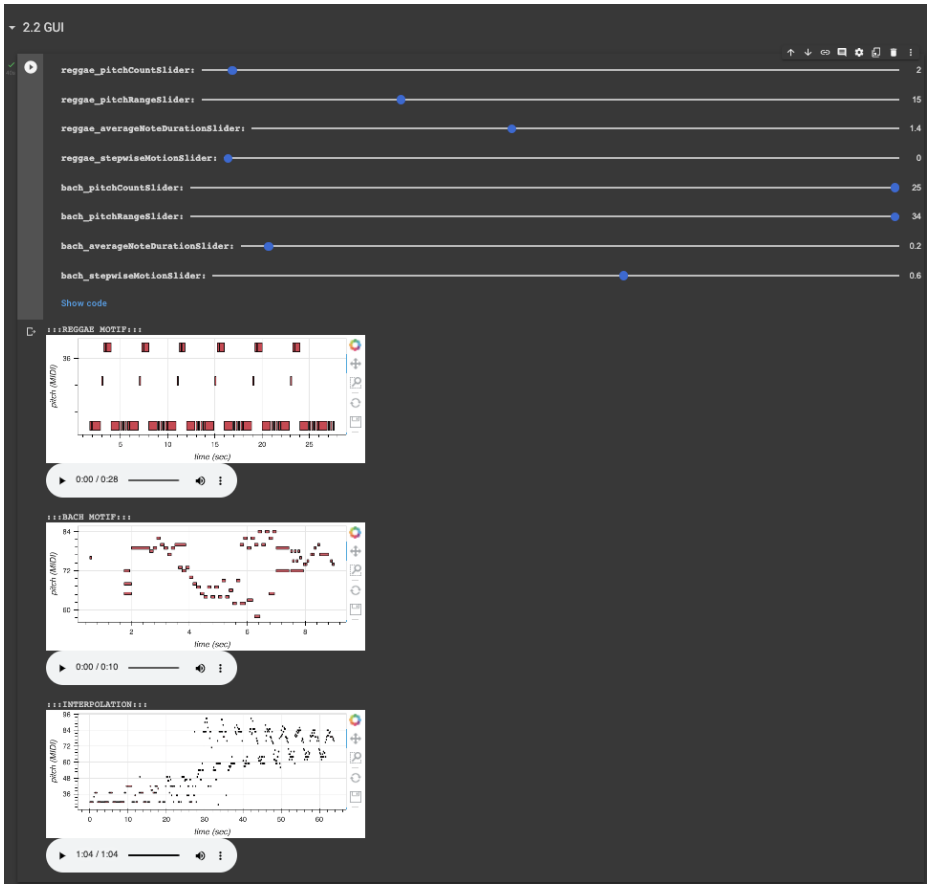

Fig. 1: Screenshot of the Casual Creators's GUI.

4 Experiments & Results

Here the loss values from the different NN configurations, referred to as M1-4 throughout (see Section 3.2), are reported. We also present cherry-picked examples of interpolated outputs.

4.1 Experiments with Generative Models

Figure 2 and 3 shows the training loss for the Bach and reggae models respectively. Both exhibit similar trends: M3 (Bi-directional) achieves the lowest loss, M4 (Stacked) is the second most accurate, M2 (LSTM) gets worse over time and M1 (GRU) is the worst model.

Although M3 (Bi-directional) achieves the lowest loss, the outputs were often minimal or sporadic (see Figures 4 and 5). Perhaps, the model overfit to the data, frequently

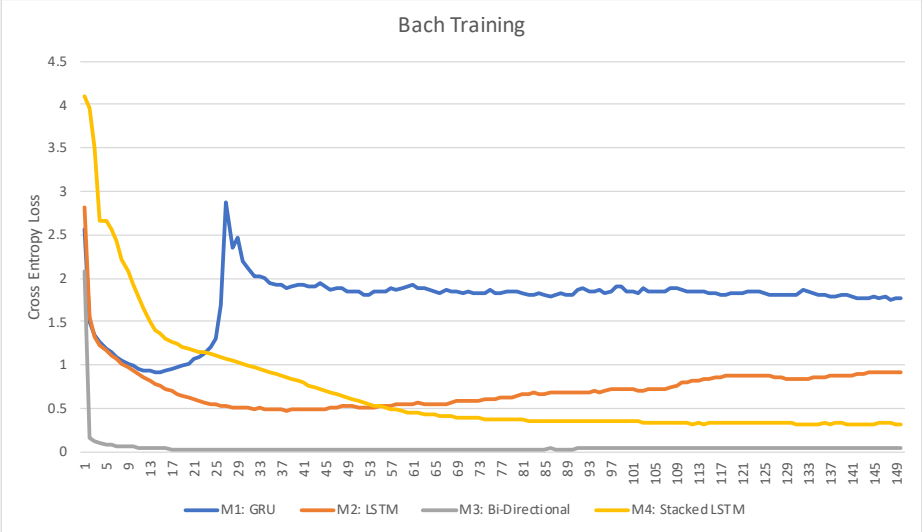

Fig. 2: Plot showing the loss over epochs for each neural model configuration, for the Bach dataset.

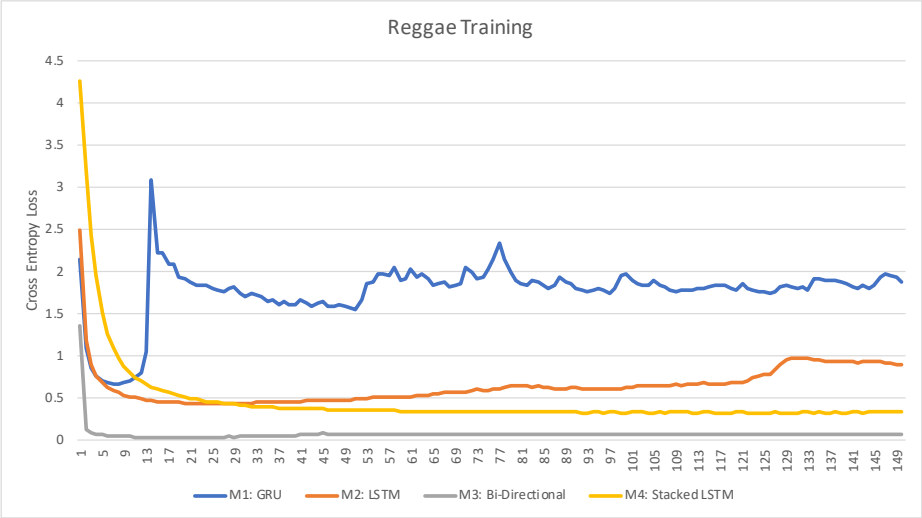

Fig. 3: Plot showing the loss over epochs for each neural model configuration, for the reggae dataset.

generating rests or empty bars. In the authors opinion, M4 (Stacked) returned much more stable and musically coherent outputs.

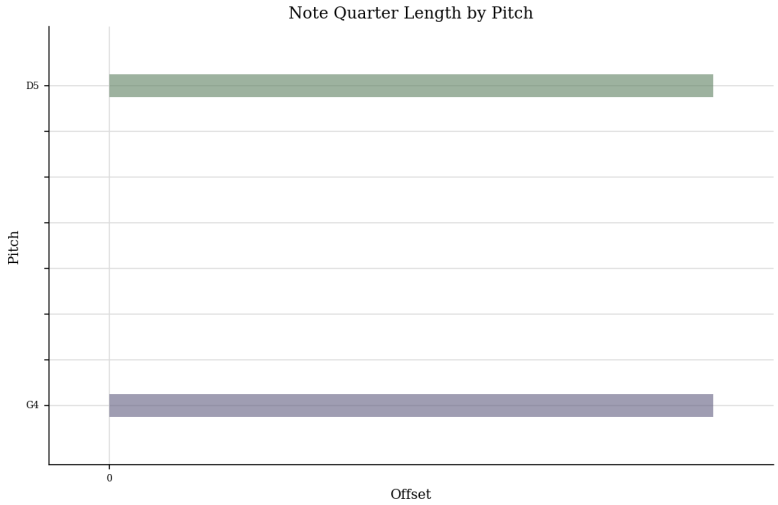

Fig. 4: Example of a poor, possibly overfit, output from M3 (Bi-directional) on the Bach data

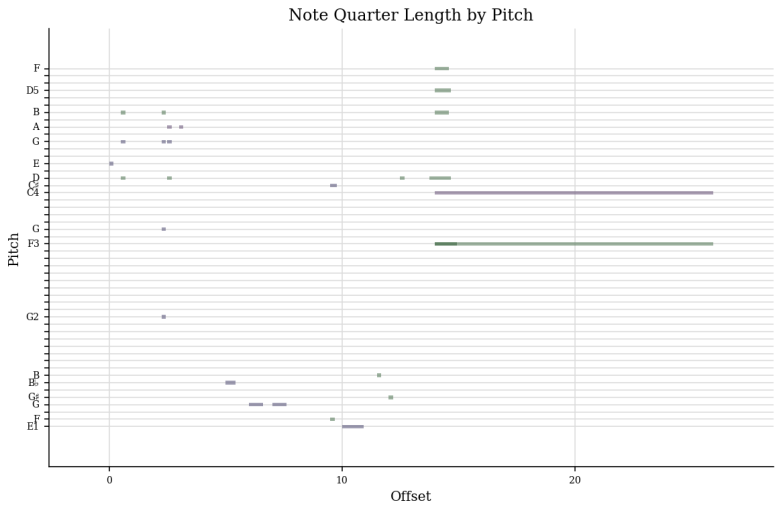

Fig. 5: Example of a poor, possibly overfit, output from M3 (Bi-directional) on the reggae data.

4.2 Cherry-Picked Outputs

By using the interface, the author found that some of the most fun interpolations occurred when extracts were at polar extremes, e.g., a reggae bassline with limited pitch range interpolating to a busy Bach chorale in a higher register (see Figure 6).

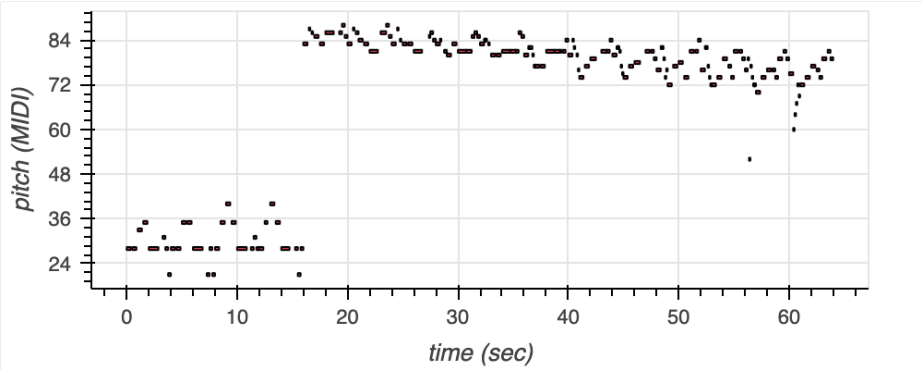

Fig. 6: An interpolation where the reggae and Bach extracts were at contrasting extremes.

However, this led to quite extreme jumps in the interpolation's pitches. Taming the extracts to have similar average durations and similar stepwise motion somewhat helped this (see Figure 7). Where the Bach gradually entered with the reggae bass lines occurred when the Bach note range was wide enough to overlap with the reggae note range (see Figure 8 and 9.

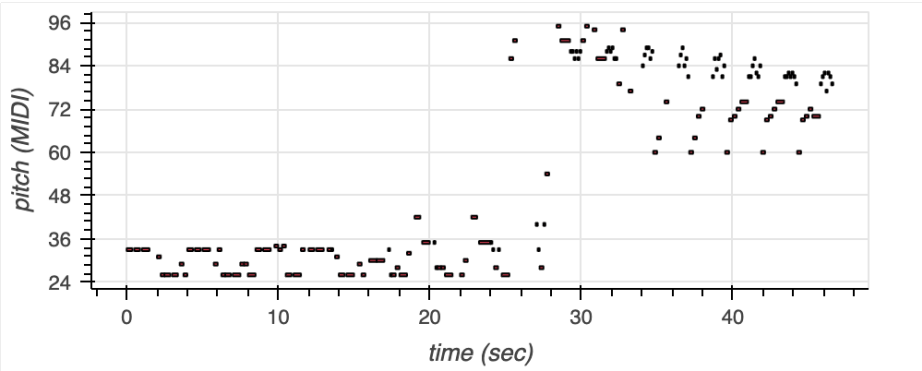

Fig. 7: An interpolation where the avereage duration and stepwise motion for both input genres is similar.

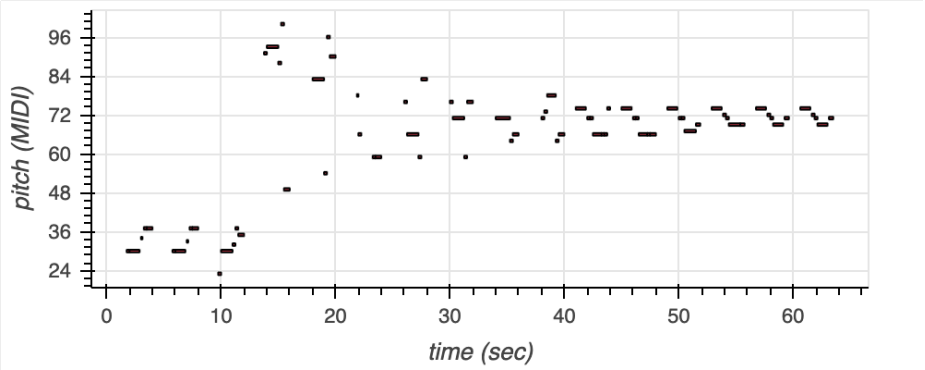

Fig. 8: An interpolation where the Bach and reggae note ranges overlapped, causes a more gradual piecing together of the two extracts.

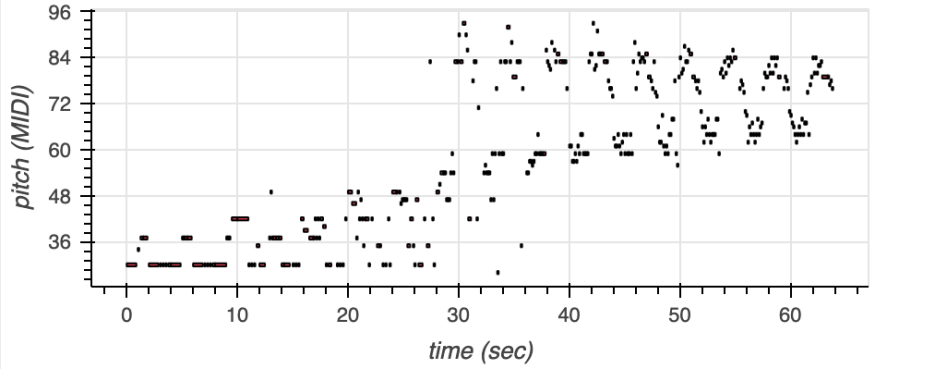

Fig. 9: Another interpolation where the Bach and reggae note ranges overlapped, causes a more gradual piecing together of the two extracts. This is the author's favourite example.

Matching the slider values exactly for both genres was also an intriguing experiment, as the musical qualities shift more gradually towards the Bach style whilst retaining a similar rhythmic quality (see Figure 10).

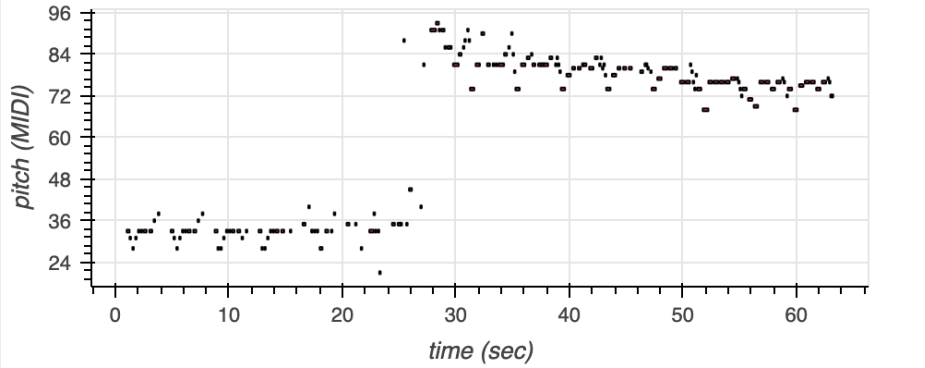

Fig. 10: A piece where both the Bach and reggae extracts have identical musical metric values.

5 Discussion

Here the high-level computational creativity issue of CC [3] is considered. Specifically, each of the CC design patterns, P1-4, introduced in Section 2.1, are explored.

The design satisfies P1 by always generating output, no matter what slider values are used — no initial action needs to be taken other than running the cell. The user also only needs to use one-touch interaction (P4), making it simple and pleasurable to explore.

Actions are also limited (P2) by providing high-level musical metrics to guide the selection of extracts. The metrics were mostly good choices, with note duration and pitch count having clear effects on the music. As the interpolations often have a large jump between sections, perhaps it would've been useful to have an average pitch control also, to tame the difference between the extracts, helping to better modify the meaningful (P3). Indeed, perhaps applying pitch changes post-hoc, and training the generative models on augments musical tracks (in the same key), might lead to better user control and stronger results.

5.1 Future Work

As a colab demo, the app requires compilation each time sliders are changed. For the best CC experience, a more rapid-feedback loop is needed. As their is a relatively small possibility space, it might be possible to pre-compute interpolations, and load responses when needed.

The app is also limited to the styles of music the NN were trained on. It might be possible to use processes such as MidiMe (see [5]) to quickly generate musical extracts based on data given by the user, possibly increasing their feelings of ownership cf. [3]. The sliders also require some knowledge of the music domain (e.g. what is pitch), which won't support most casual users.

Moreover, sharing options (e.g. being able to tweet your song) or slider configurations would help to build a community around our CC [3].

Finally, using the generated extracts to create a web app – perhaps using p5js – would complete the CC experience.

References

1. B. Banar and S. Colton, "Evaluation of GPT-2-based Symbolic Music Generation," in *Proceedings of the DMRN+16: Digital Music Research Network One-Day Workshop*, 2021.
2. J.-P. Briot, G. Hadjeres, and F.-D. Pachet, "Deep learning techniques for music generation – a survey," 2017. [Online]. Available: https://arxiv.org/abs/1709.01620
3. K. Compton and M. Mateas, "Casual Creators," in *Proceedings of the 6th International Conference on Computational Creativity*, 2015.
4. M. S. Cuthbert and C. Ariza, "music21: A toolkit for computer-aided musicology and symbolic music data," 2010.
5. M. Dinculescu, J. Engel, and A. Roberts, Eds., *MidiMe: Personalizing a MusicVAE model with user data*, 2019.
6. D. Eck and J. Schmidhuber, "A first look at music composition using lstm recurrent neural networks," Tech. Rep., 2002.
7. O. Fabius and J. R. Van Amersfoort, "Variational recurrent auto-encoders," *arXiv preprint arXiv:1412.6581*, 2014.
8. N. Fradet, J.-P. Briot, F. Chhel, A. El Fallah-Seghrouchni, and N. Gutowski, "Miditok: A python package for midi file tokenization," in *22nd International Society for Music Information Retrieval Conference*, 2021.
9. G. Hadjeres, F. Pachet, and F. Nielsen, "Deepbach: a steerable model for bach chorales generation," 2016. [Online]. Available: https://arxiv.org/abs/1612.01010
10. A. Karpathy, "The unreasonable effectiveness of recurrent neural networks." [Online]. Available: http://karpathy.github.io/2015/05/21/rnn-effectiveness/
11. D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.
12. D. P. Kingma and M. Welling, "Auto-encoding variational bayes," 2013. [Online]. Available: https://arxiv.org/abs/1312.6114
13. A. Pati, A. Lerch, and G. Hadjeres, "Learning To Traverse Latent Spaces For Musical Score Inpainting," in *Proceedings of the 20th International Society for Music Information Retrieval Conference*, 2019. [Online]. Available: https://archives.ismir.net/ismir2019/paper/000040.pdf
14. E. Petrovskaya, C. S. Deterding, and S. Colton, "Casual creators in the wild : A typology of commercial generative creativity support tools," *Eleventh International Conference on Computational Creativity*, May 2020.
15. M. Phi, "Illustrated guide to lstm's and gru's: A step by step explanation." [Online]. Available: https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21

16. A. Roberts, J. Engel, C. Raffel, C. Hawthorne, and D. Eck, "A Hierarchical Latent Vector Model for Learning Long-Term Structure in Music," in *Proceedings of the 35th International Conference on Machine Learning*, 2018. [Online]. Available: https://goo.gl/magenta/musicvae-code
17. B. Shneiderman, G. Fischer, M. Czerwinski, M. Resnick, B. Myers, L. Candy, E. Edmonds, M. Eisenberg, E. Giaccardi, T. T. Hewett, P. Jennings, B. Kules, K. Nakakoji, J. Nunamaker, R. Pausch, T. Selker, E. Sylvan, and M. Terry, "Creativity Support Tools: Report From a U.S. National Science Foundation Sponsored Workshop," *International Journal of Human-Computer Interaction*, vol. 20, no. 2, pp. 61–77, 2006. [Online]. Available: https://doi.org/10.1207/s15327590ijhc2002_1
18. B. L. Sturm, J. F. Santos, O. Ben-Tal, and I. Korshunova, "Music transcription modelling and composition using deep learning," *arXiv preprint arXiv:1604.08723*, 2016.

Appendix

The colab notebook for this project can be found here: `https://colab.research.google.com/drive/1CQ6TqUeD3o9xNN8DWwA3rZhCInbmZ0PX?usp=sharing`