Singing Style Transfer

Andrew Li, Joseph Zhong, Ollin Boer Bohan

Abstract

Although neural style transfer for images has been highly successful, these algorithms have not yet been successfully applied to the audio domain. We propose to attempt this task in order to allow for style transfer of audio (focusing specifically on monophonic, singing audio) as a useful artistic tool for vocal processing in music production.

Project Scenario and Goals

A user (music producer / artist) submits a *style* audio file (assumed to be an acapella, a.k.a. a soloed sung vocal) and a *content* audio file to the app. The style from the *style* audio file is applied to the content from the *content* audio file, the result is converted back to a waveform representation, and the transferred audio is presented back to the user. If the result is low quality, it may only be used for inspiration or as a backing track, as with current-day vocoders; if the result is very good, it could be used for enhancing a lead vocal to match a professionally edited reference vocal.

This task (singing style transfer) is analogous to the task of image style transfer pioneered in Gatys et al (2015) and related to the task of image-to-image translation developed in Pix2Pix, MUNIT, and others, but it has proved much less tractable so far. The primary constraint is on the quality of the generated audio; the system does not need to be cheap or fast, but it must produce moderate-to-high-quality results in order to have value as a tool for production.

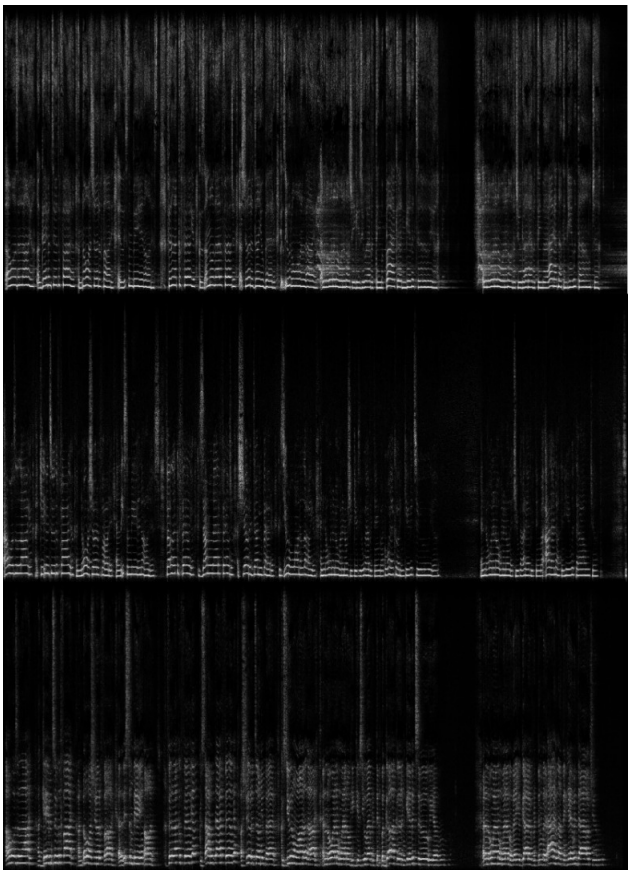
Design Strategy

An overall app would consist of a simple web frontend allowing submission of audio to a python webserver running the main style transfer module. The main style transfer module would be implemented using `librosa` for audio conversion and some combination of hand-engineered-processing (using `librosa` and other python libraries for audio/image processing) and neural-network based processing (using Keras, PyTorch or TensorFlow). Most processing will occur in the spectrogram domain.

Dataset: We will likely need to collect:

- A (small) parallel dataset of aligned audio files containing the same content with different styles. This allows us to develop an understanding for the components of style in order to refine our processing modules, and also provides a testing mechanism

(since if we have two content-equivalent files a, b, each cut in half to form a_1, a_2, b_1, b_2, style transfer with style a_1 and content b_2 should produce a_2). We can build this dataset using covers available on YouTube and SoundCloud. For example, here are three different spectrograms of Ariana Grande - One More Time (original, and two covers) collected during our early exploration:



- A larger non-parallel dataset of acapellas for training learned-processing modules. This is *mostly* already collected (we have 32 high-quality acapellas available from the AcapellaBot project), but we could add to it / clean it up a bit.

Non-Learned Processing: Hand-engineered processing modules will likely include:

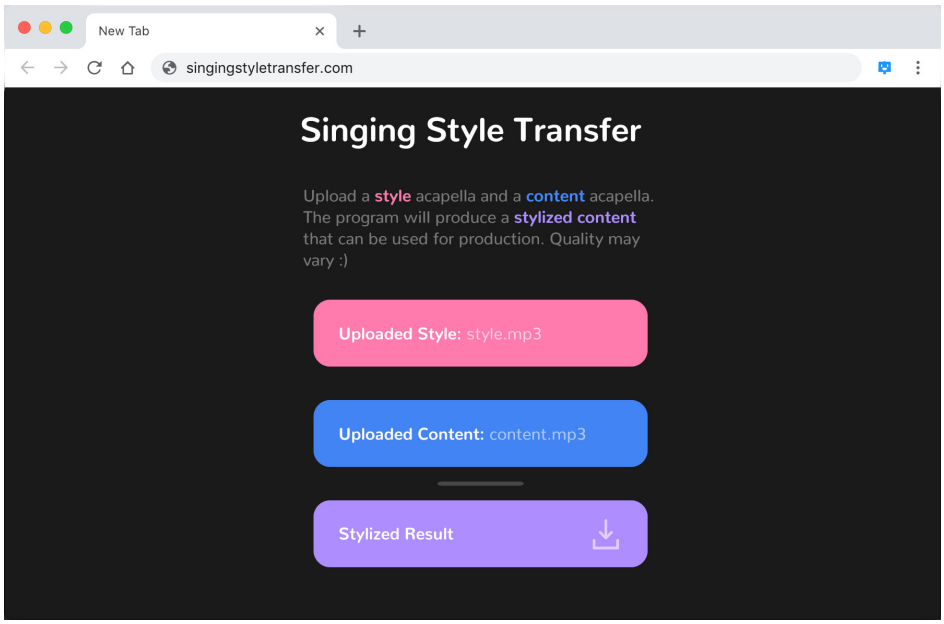
- Low-level style transfer (including transfer of vibrato and reverb tails) so that the shape of individual harmonics matches the style audio.
- Sound-conditioned spectral envelope matching and amplitude matching (including recovery of missing harmonics or high-frequency detail) to make the dynamics and mixing for vocal sounds (sibilants, vowels, in/exhalations) match those in the style audio.
- Global smoothed spectral envelope matching to achieve similar overall mixing.

Learned Processing: Neural-network processing modules will likely include:

- Spectrogram-domain post-processing to remove artifacts (particularly with respect to output phase)
- Audio-domain post-processing to remove artifacts and improve plausibility of the generated audio.

The networks may require several versions to settle on a successful architecture, but an initial implementation would be an image-to-image GAN operating on slices of the input audio, using a spectrogram feature representation. It is highly probable that adversarial networks will be required to generate plausible, sharp output. As the project progresses, we may be able to replace or subsume hand-engineered components by neural networks once we understand the subproblems better.

Frontend The frontend is a much lower priority than the backend (since even a command line tool would be useful), but if we are able to generate results an ideal frontend would be something like:



Design Unknowns / Risks

The primary challenge is the development of a qualitatively good style transfer mechanism (including developing a good conceptual description of what style *is* with respect to audio).

Although we've implemented neural nets for audio processing before, and have tested naive image-based style transfer, neither us (nor anyone else) have yet developed an audio style transfer architecture for singing audio that products *intelligible* and *plausible* results.

Example approaches:

- Autoencoder Based Architecture for Fast & Real Time Audio Style Transfer (2018): no samples given, but spectrogram sample looks unintelligible
- Refined Wavenet Vocoder For Variational Autoencoder Based Voice Conversion (2018): samples are intelligible but not plausible, on the simpler task of speech-to-speech conversion.
- TimbreTron: A WaveNet(CycleGAN(CQT(Audio))) Pipeline (2018): samples are intelligible but not plausible, on the simpler task of instrument-to-instrument conversion
- A Lightweight Music Texture Transfer System (2018): samples are intelligible but not plausible
- Multi-target Voice Conversion without parallel data by Adversarially Learning Disentangled Audio Representations (2018): samples are somewhat intelligible and not plausible
- A Universal Music Translation Network (2018): samples are intelligible and moderately plausible but not good enough for singer-to-singer translation
- Singing Style Transfer Using Cycle-Consistent Boundary Equilibrium Generative Adversarial Networks (2018): samples are intelligible and plausible for the highly limited case of female-to-male / male-to-female translation.
- Voice style transfer with random CNN (2018): samples are intelligible and moderately plausible on the simpler task of speech-to-speech conversion.

Implementation plan and schedule

Data collection: Collect a small parallel dataset of aligned acapellas from YouTube / SoundCloud for testing (see **Dataset** under **Design Strategy** above). There is currently no existing good parallel corpus for singing audio, so this alone is potentially a meaningful contribution to the field.

Naive Models Implement components that:

- Extracts the underlying pitch envelope (we can use an existing trained network like `https://github.com/marl/crepe`); this information is useful for performing over tasks
- Use patchmatch or a similar naive algorithm to copy slices from the style spectrogram to the target spectrogram. This will (hopefully) match low-level characteristics like reverb and vibrato.
- Perform spectrum-conditioned amplitude matching (training a simple, shallow model that predicts the average amplitude across a slice based on the frequency distribution of that slice) and uses this to renormalize the input audio.
- Perform global spectral envelope matching (re-weighting harmonics in the content according to their average amplitude in the style data) to produce plausible output (will still sound mostly like the source audio).

Neural Models & Post-processing

- Test existing image-to-image translation models on the spectrogram representation (potentially as a post-processing step on the naive model). Good candidates:
  - `https://github.com/NVIDIA/FastPhotoStyle`
  - `https://github.com/lengstrom/fast-style-transfer`
- Improve our pipeline to preserve phase information throughout the process, and figure out the set of FFT parameters and post-conversion parameters that maximize output quality.
- Train a spectrogram post-processing network to reduce artifacts.
- Train an audio post-processing network to reduce artifacts.

Schedule:

- **Week 1:** Write proposal. Done!
- **Week 2:**
  - ☒ [1h] Collect one or two more parallel examples for reference
  - ☒ [1h] Set up skeleton code that just performs identity transform on the content and ignores the style
  - ☒ [1h] Write global EQ matching and test it
  - ☒ [1h] Fix the skeleton code to preserve the phase, or at least do it in mono
- **Week 3-5:**
  - ☒ **[all]** Take notes on stylistic differences between Rolling in the Deep acapellas (in `data/aligned/rolling_in_the_deep`) to see if we're missing any elements of style
  - ☐ ~~**[all]** Take notes on stylistic differences between Young and Beautiful acapellas (in `data/aligned/young_and_beautiful`) to see if we're missing any elements of style~~
  - ☒ **[ollin]** Implement super-resolution (recovery of high-frequency detail)
    - ☒ Implement fundamental frequency detection (rule-based or learned)
    - ☒ Implement harmonic super-resolution (duplicating the fundamental curve up to higher octaves… this may be doable with some fancy fourier transform magic - using a saw wave instead of a sin wave somehow - but we can also just do it graphically).
    - ☒ Add multiple-harmonic sampling to fundamental detection so it's more accurate (right now it's sorta glitchy)
    - ☐ ~~**[2h]** Add fundamental cloning to super-res (right now it's just line drawing so reverb is lost)~~ Not needed for patchmatch-based approach
    - ☐ ~~**[2h]** Build this into the global spectral envelope matching part of the pipeline.~~ Not needed for patchmatch-based approach
    - ☒ **[6h]** Implement sibilant detection / super-resolution
  - ☒ **[ollin]** Implement a 1d pitch- warping patchmatch using naive features (can switch to neural features later)
    - ☒ The output is interesting but not great. Using better features will help, but we may still also need to synthesize missing sounds somehow…

- ☒ **[2h, ollin]** Implement or figure out how to use a pitch warping + formant correction thingy so that we can pitch warp when patch matching without messing up the audio
  - ☒ We can test this by using the pitch normalization experiment (already works) and making sure that our formant correction on top of this generates plausible output.
- ☒ **[lia4 / josephz]** Test using DeepSpeech featurizer for matching (i.e. using basic clone-stamp method)
  - ☒ Install DS code and run it on sample acapella (tested; actual text output is not correct but features should still work)
  - ☒ **[4h]** Recover features from early-layer activations (so, we have a component that goes raw audio -> big vector thing that is hopefully semantically meaningful)
  - ☒ **[2h]** Test feature-based clone-stamping
  - ☒ **[2h]** Test feature-based clone-stamping with pitch warping
  - ☒ **[6h]** Test feature-based clone-stamping with pitch warping and formant correction
- ☒ ~~Test out neural-network image-to-image approaches (e.g. the fast photo style transfer thing) so that we have some baseline of what results modern end-to-end systems yield~~ nvm
  - ☒ ~~`https://github.com/NVIDIA/FastPhotoStyle`~~ this probably won't work
  - ☒ ~~`https://github.com/msracver/Deep-Image-Analogy`~~ we're using this as a reference!
  - ☐ Try `https://github.com/pkmital/time-domain-neural-audio-style-transfer`

- **Week 5-7:**
  - ☐ Fixes to current pipeline
    - ☒ **[ollin]** There's probably a bug or two in the harmonic reweighting version of sst.py right now, the output sounds worse than patch matching even though my testing code shows that harmonic reweighting should sound great. **EDIT:** Fixed now, I was 1) not pitch normalizing correctly in the harmonic feature computation and 2) reweighting the original content instead of the super-resolution content
    - ☒ **[andrew / joseph]** Figure out why DeepSpeech features are incorrectly sized and how to properly correct for this, rather than just blindly rescaling the features bilinearly :P - **EDIT:** Turns out we're doing this correctly! The only difference is our FFT window sizes–we're using 1536 samples, but DS is using 0.02s ~= 1765 samples. The DS network produces features of the same size as the input, so we just need to stretch the time axis.
    - ☐ **[andrew / joseph]** Clean up DeepSpeech feature-retrieval code to be faster and less hacky (shouldn't need to write to temp files, shouldn't need to have python calling a shell script calling python, should ideally be able to batch multiple inputs)
    - ☐ **[ollin]** Make PatchMatch faster if possible, right now it takes too long with high iteration counts (~20).

    - ☐ [??] Test if using pitch-normalized inputs improves DS feature matching results, and, if so, switch to doing that.
  - ☐ [**All**] Implement initial draft of post-processing networks (see `notes/post_processing_net.md` in the repo)
  - ☐ [??] Test the primary source of PatchMatch error by comparing the MSE in feature space between our PatchMatch reconstruction and an n^2 nearest-neighbor search.
    - ☐ **If n^2 search is better (low prior):** If the conclusion is that PatchMatch can't *find* good patches, because of lack of continuity in the feature vectors, we may need to switch to a lookup-based approach (e.g. kd trees), hybrid approach (splitting the style image into consonant and vowel sub-components and only searching in the correct one), cheat by blurring the feature vectors along the time axis, or just run PatchMatch with higher iteration counts.
    - ☐ **If both are equal (high prior):** then we need to determine if the issue is that our features are bad, or that good matches don't exist. We can do this by comparing the results of PatchMatch using the actual style (which doesn't necessarily have good matches) and the "reference_stylized" as the style (which necessarily has good matches for everything).
      - ☐ **If PatchMatch with reference_stylized is way better (low prior):** then the conclusion is that *good patches don't exist* (because the style audio does not necessarily contain all of the necessary consonant / vowel sounds), and we need to figure out how to resynthesize them and add this to the search.
      - ☐ **If both are equal (high prior):** then the conclusion is that both the harmonic features and the DeepSpeech features are garbage for matching, and PatchMatch on them is doomed, test alternate featurizers `https://github.com/fordDeepDSP/DeepSpeech/issues/13` e.g. `https://drive.google.com/file/d/1E65g4HlQU666RhgY712Sn6FuU2wvZTnQ/view` or just pray that post-processing can save us.
- **Week 7-9:**
  - **Best-case:** work on the demo, make a pretty web interface, speedup
  - **Worst-case:** focus on dataset, and do a more rigorous comparison of existing methods, including notes for how they can be improved.

Evaluation

The primary evaluation metric will be qualitative–does the result sound **a)** intelligible (the content is preserved) and **b)** plausible (the style is transferred)? For songs within our parallel dataset we can compare against the gold-standard stylized output (see **dataset** above), but the most important test for our program will be qualitative evaluation on unseen data.

If we develop a moderately successful method, we can potentially conduct broader testing comparing mean opinion scores of our method to baselines.

Related work

https://arxiv.org/abs/1807.02254v1

https://nips2017creativity.github.io/doc/Neural_Style_Spectograms.pdf

http://madebyoll.in/posts/singing_style_transfer/

https://github.com/msracver/Deep-Image-Analogy

https://arxiv.org/pdf/1705.01088.pdf

http://openaccess.thecvf.com/content_cvpr_2018/papers/Gu_Arbitrary_Style_Transfer_CVPR_2018_paper.pdf

https://github.com/madebyollin/acapellabot

https://arxiv.org/pdf/1711.11585.pdf