Jay Sebastian

CS 6501

Neural Style Music Visualizer

Overview: This project uses an implementation of the Neural Style paper in conjunction with Python signal processing libraries and Pixi.js (WebGL library) to create a music visualizer. When a song's bass tones are dominating, the background will show one style of some ambient background, and when the treble tones are dominating, the background will show a different style for the same background. "In-between" tones will present a blend of the two styles. Additionally, particles will appear for strong bass (blue) and treble (red) noises. These particles are initialized with a random speed and direction but they follow a vector field formed by the gradient of the background image.

Problem: Creating a visually interesting “movie” that adapts to the music. There is no objectively correct solution to this problem, as the design space is quite large. For this toy project, I decided to have our visualization be some background image that morphs between styles depending on the dominant frequencies in the input song, as well as some particle systems that accent the louder noises.

Approach: I wanted to leverage one of the photo collection works we discussed in class – the Gatys et al. paper (Neural Style) felt like the best choice since we can use it to transform the same image into a host of other images that look similar but have a different mood (based on color pallet, brush strokes, etc.). Another good candidate paper was Laffont et al. (Transient Attributes – particularly the season transfer).

With the paper chosen, I found an implementation online which took care of the heavy lifting in terms of setting up the neural network and producing images. We used the VGG-19 network, which was created for ILSVRC-2014 for localization and classification challenges. The 16 convolutional layers of this network were used (3 fully connected layers at the end were unused). This implementation had an option to use multiple style images with different weighting factors, which was convenient for my task. I decided to generate five “keyframes” of some ambient content image using two different styles – one representing bass and the other representing treble – with different weighting factors for each frame. This process took about 2 hours on a CS server using its CPU (I was having issues with Torch running CUDA because apparently something was not installed correctly).

With that done, I created a simple signal processing function that would read a WAV file, split it into many small samples, and compute the periodogram (power spectral density) of each sample. The strongest frequency overall as well as the strongest frequencies for certain passbands were written to a file.

Another auxiliary task which was part of the visualization was creating a vector field for the background based on the gradient of the image. I used the image that was transformed with 50% of each of the two styles as input. Using Python’s image processing libraries, I converted the image to grayscale, blurred it, computed its gradient, and wrote that vector field to a file.

For rendering, I used a library called Pixi.js to draw images using WebGL. As the webpage started, I loaded all 5 keyframes of background and stacked them on top of each other. Manipulating their alpha values, I could control which blend of styles was the strongest. I also retrieved the song frequency data and background vector field data from the server.

As the song plays, the strongest frequency for the current audio sample is queried and a Gaussian distribution of weights for the images is computed, where the mean is at the style blend representing that frequency. For example, if the cut frequency of our song is 1000 Hz (i.e. we define bass as below 1000 Hz and treble as above), and the current sample of our song is 300 Hz, we would want to show a background style that is primarily “bass.” Therefore, the opacity for the 100% bass style keyframe would be highest, while the opacity for the 75% bass, 25% treble style keyframe would be second highest – the relative amounts are determined by the stddev of the Gaussian (some user-determined parameter). Alternately, if the sample is right at 1000 Hz, we would want to show a style that has a 50-50 blend of bass and treble, so the keyframe with 50% bass, 50% treble style would have the highest opacity. We use a logarithmic scale to determine distance from the cut frequency in this computation, as that makes the most sense for audio. With the weights computed for our current position in the song, we adjust the alpha values accordingly, “smoothing” with linear interpolation so that the transition from frame to frame in our animation is not too abrupt (this would lead to a flickering visual for most types of music).

After finishing the background, I added some simple particle systems to go on top. I designated the color blue as “bass” and the color red as “treble.” As the song is playing, the strongest frequency components in the bass and treble passbands (as computed earlier) are queried. If these components are above a certain threshold, a burst of particles of the appropriate color (with some small random variation in tint) are created at a random location (the number of particles is determined by the strength of the signal). These particles are initialized with a random direction and speed, but afterward their speed is affected by the vector field obtained from the server. The purpose of this part of the system is to more clearly visualize “punchy” parts of music like a bass drum.

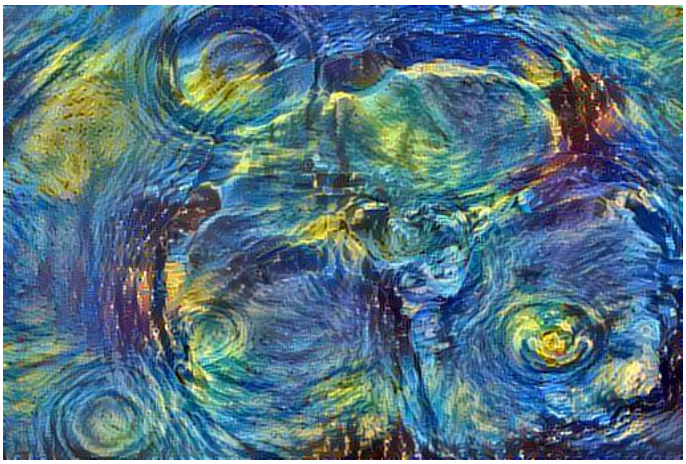
Results: Please run the attached code (index.html) from a local server running PHP (I used XAMPP to do this). The graphics part is of course done client side, but the server needs to send the frequency data and vector field data to the client so that the visualization can work. Pressing “Play” without changing anything will visualize the track “Ready err Not” by the artist Flying Lotus using the Whirlpool Galaxy background.

Overall I was satisfied with my product, although there are many different directions I can go with it now. I noted these on the bottom of index.html – in terms of graphics, creating even more dynamic visuals, possibly by interpolating between keyframes rather than varying their opacities, would be neat. I could also try transforming a music video rather than some ambient background, though that would take much more computation (since there are many more frames) as well as some additional coding for temporal coherence from frame to frame.

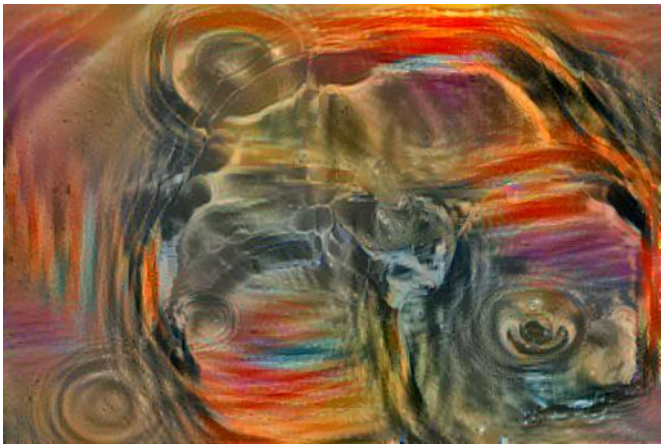
In case the WebGL app does not work, I have provided some pictures of keyframes I obtained from Neural Style

Original Content Image – Water (taken from Wikipedia article on capillary wave)

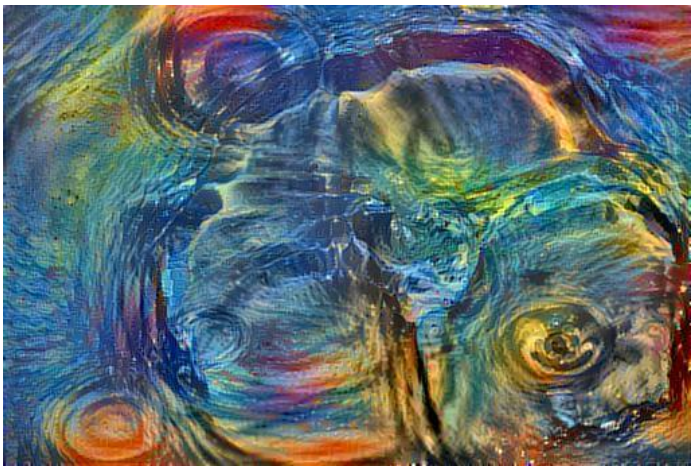
Water with Starry Night by Van Gogh



Water with The Scream by Munch



Water with 50% Starry Night, 50% The Scream



Original Content Image – Galaxy (taken from Wikipedia article Whirlpool Galaxy)

Galaxy with Seated Nude by Picasso

Galaxy with Composition VII by Kandinsky

Galaxy with 50% Seated Nude, 50% Composition VII

Screenshot of the system