

Hearing Paintings

Understanding abstract art through sonification

CAS AICP Module 3 Exercise
Samuel Flückiger

Hearing Paintings: Sonifying Abstract Art with Neural Networks

Research questions

- (1) *How do different AI-based tools translate images to sound help me identify patterns and characteristics within a series of abstract paintings?*
- (2) *Can pre-trained neural networks and image-sonification tools be used to extract recurring visual patterns from one painter's work and translate them into distinguishable sonic structures?*

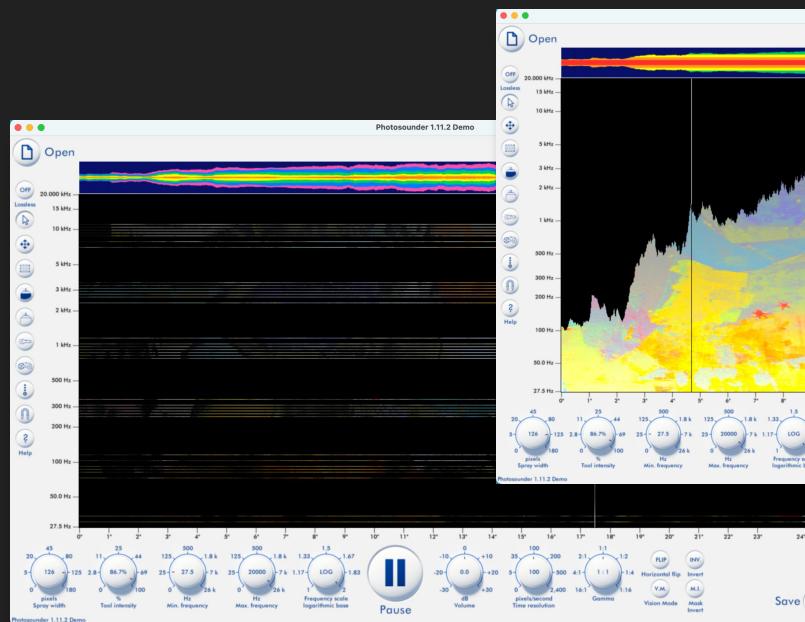
Key milestones

- Create a dataset: ~45 paintings
- extract visual structure using neural networks / deep learning
- design a mapping to sound
- compare at least two approaches.

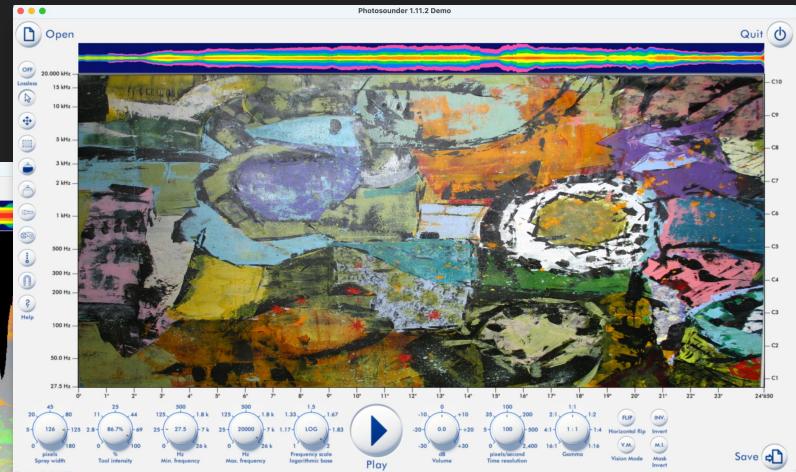
The paintings Katharina Bucher



Image Sonification Tools (1): Photo Sounder

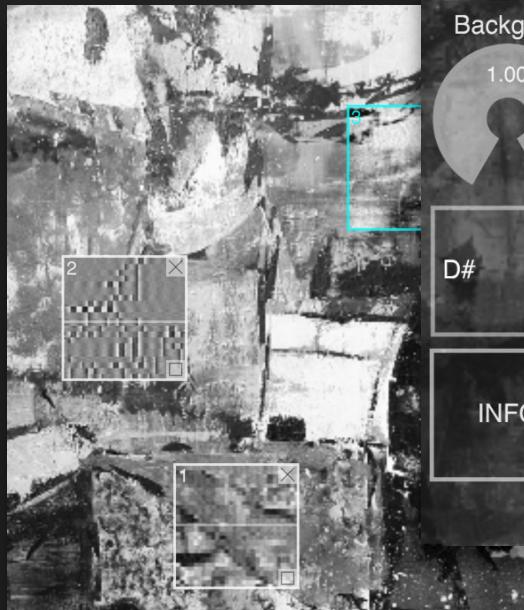


Group to nearest semitone -> Pipe Organ



Vertical Curves -> Wind

Image Sonification Tools (2): Vosis



The screenshot displays the Vosis software interface, which overlays various sonification parameters and controls onto an aerial photograph of a landscape. The interface includes:

- Top row: Circular sliders for Backgrnd (1.00), RevMix (0.15), RevDec (0.38), BPM (70), and Gain (-3).
- Middle row: Buttons for D# (with a dropdown arrow), MJR (with a dropdown arrow), and three other unlabeled buttons.
- Bottom row: Buttons for INFO, LIMITR, EQLZ, STEREO, and two arrows (<< and >>).
- Bottom right corner: A smaller inset showing a zoomed-in view of the software's controls.
- Bottom center: The text "Electronic, musical sounds".

Image Sonification Tools (3): MaxMSP

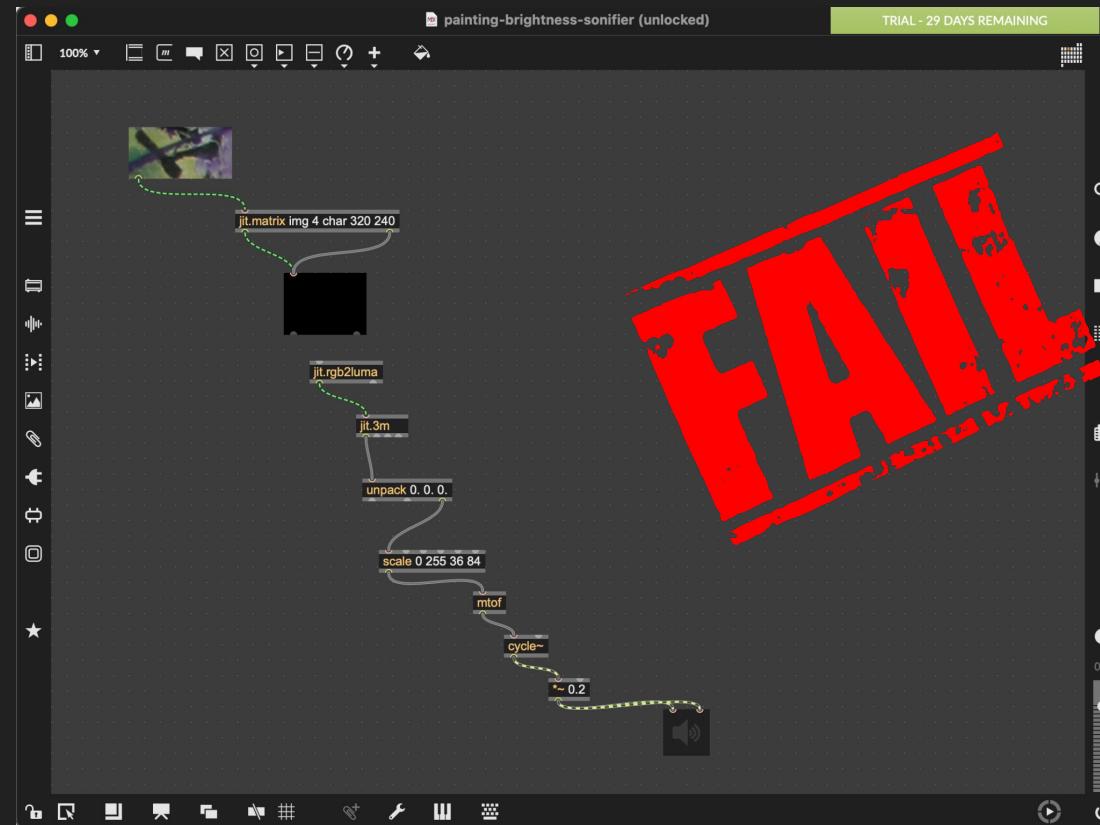


Image Sonification Tools (4): Working with GPT5 & Python

Jupyter Notebook interface showing code for image sonification:

```
# Welcome Exercise_Module_3_SF.ipynb Exercise_Module_3_Refined_SF.ipynb
Exercise_Module_3_SF.ipynb > # Updated Cell - create_midi_from_pca with motif depending on PCA
Generate + Code + Markdown | Run All Restart Clear All Outputs JupyterVariables Outline ...
old_min4, old_max4 = PC_MINS[3], PC_MAXS[3]
pc4_norm = normalize_value(pc4, old_min4, old_max4, 0.0, 1.0)

# If pc4_norm is high, we sometimes add +12 (one octave up)
use_octave_jumps = pc4_norm > 0.6

return base_pattern, use_octave_jumps

# Cell 7 - Helper function to map values between
def normalize_value(x, old_min, old_max, new_min, new_max):
    """
    Map x from [old_min, old_max] to [new_min, new_max]
    If old_min == old_max, return the midpoint of [new_min, new_max]
    """
    if old_min == old_max:
        return (new_min + new_max) / 2.0
    return new_min + (x - old_min) * (new_max - new_min) / (old_max - old_min)

# Updated Cell - create_midi_from_pca with motif
def create_midi_from_pca(pca_vec, out_path: Path):
    """
    Map the 4 PCA values to musical parameters and write a short 4-bar MIDI file to out_path.
    Uses actual min/max of each PCA component and
    ...
    pc1, pc2, pc3, pc4 = pca_vec

    # Use real min/max for each PCA component
    old_min1, old_max1 = PC_MINS[0], PC_MAXS[0]
    old_min2, old_max2 = PC_MINS[1], PC_MAXS[1]
    old_min3, old_max3 = PC_MINS[2], PC_MAXS[2]
    old_min4, old_max4 = PC_MINS[3], PC_MAXS[3]

    # Map PCA - musical parameters
    base_pitch = int(normalize_value(pc1, old_min1, old_max1, 48, 72)) # C3-C5
    bpm = int(normalize_value(pc2, old_min2, old_max2, 60, 140)) # tempo
    note_density = int(normalize_value(pc3, old_min3, old_max3, 2, 10)) # notes/bar
    velocity = int(normalize_value(pc4, old_min4, old_max4, 60, 120)) # loudness
```

Output of the code:

```
Saved MIDI: 01.mid
Saved MIDI: 02.mid
Saved MIDI: 03.mid
Saved MIDI: 04.mid
Saved MIDI: 05.mid
Saved MIDI: 06.mid
Saved MIDI: 07.mid
Saved MIDI: 08.mid
Saved MIDI: 09.mid
Saved MIDI: 10.mid
```

Output of the code (continued):

```
error Traceback (most recent call last)
Cell In[10], line 6
      4     midi_name = img_path.stem + ".mid" # same name as image, but
      5     midi_path = MIDI_DIR / midi_name
--> 6     create_midi_from_pcapca_vec(midi_path)
      8 print("Done! Check the midi/ folder next to your notebook.")

Cell In[9], line 37
      34         midi.addNote(track, 0, pitch, note_start, duration, velocity)
      35     with open(out_path, "wb") as f:
--> 37     midi.writeFile(f)
      38 print("Saved MIDI!", out_path.name)

File ~/anaconda3/envs/CASIAISound/lib/python3.10/site-packages/midiutil/MIDIFile.py:1633, in MIDIFile.writeFile(self, fileHandle)
    1632 self.tracks[-1].close()
--> 1633 Close the tracks and have them create the MIDI event data structure.
    1634 self.close()
    1635 # Write the MIDI Events to file.
    1640 for i in range(0, self.numTracks():

File ~/anaconda3/envs/CASIAISound/lib/python3.10/site-packages/midiutil/MIDIFile.py:1697, in for i in range(0, self.numTracks):
    1696     self.tracks[i].adjustTimeAndOriginInOrigin, self.adjust_origin)
--> 1699     self.tracks[i].writeMIDIStream()
...
--> 2157 fourbyte = struct.pack('>I', self.tempo) # big-endian uint32
2158 threebyte = fourbyte[1:4] # Just discard the MSB
220 varTime = writeVarLength(self.tick - previous_event_tick)

error: argument out of range
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

Thumbnail preview of generated MIDI files:

