U-Net-Based Spectrogram Generation for Major Triad Chords: Mixed Losses and Frequency Isolation

Harry Leiter
harry.p.leiter.th@dartmouth.edu

Taka Khoo
matthew.t.khoo.25@dartmouth.edu

Doruk Ozel
doruk.ozel.26@dartmouth.edu

Abstract

We propose a system for converting a low-resolution digit image (e.g. 28×28 from MNIST/EMNIST) into a 1008×1008 spectrogram that isolates three frequency lines corresponding to a major triad chord (fundamental, major third, perfect fifth). We use a specialized data-conditioning pipeline (upscaling, Gaussian weighting, frequency row rounding, Perlin noise, etc.) to create target spectrograms and then train a U-Net to learn this mapping. We describe our multistage training procedure, including attempts with MSE vs. L1 vs. Huber loss, adding multiscale structural similarity (MS-SSIM) terms, and even partial frequency-domain losses. Despite extensive experimentation and some local minima, our final pipeline consistently yields chord-based spectrograms from arbitrary-digit inputs. This paper details every design choice, from code-level function definitions to advanced music rationale for major triad intervals.

1 Introduction

Mapping an image to an audio representation can be an artistic interface. Our goal is to create a system that takes a simple digit image 28×28 and produces a chord-based spectrogram (1008×1008) that contains frequencies for a major triad. The final audio is pleasant, since only fundamental (f_0), $1.25f_0$, and $1.5f_0$ rows are strong. This concept stems from the desire to let a user produce stable chord tones from any image, ensuring that no dissonant frequencies appear.

We build on previous approaches to chord generation (2) and image-based audio transformations. Our pipeline has multiple steps:

1. **Data conditioning:** Upscale the digit from (28×28) to (1008×1008), apply Gaussian weighting, isolate integer-rounded harmonic rows, add Perlin noise (3), etc.
2. **U-Net model:** We train a U-Net (1) to learn the mapping from the original image $(1, 28, 28)$ to the final chord spectrogram.
3. **Loss function evolution:** We explore MSE, L1, Huber, MS-SSIM (4), and partial losses in the frequency domain, describing how each affected training.
4. **Inference and audio synthesis:** We interpret the final spectrogram rows as frequencies, summing the sines in each row for a 1 second chord.

Though the final result is simple (major triad lines), our training journey encountered local minima, partial success with SSIM, large-scale dimension mismatch issues, etc. We detail these to show the complexity behind making a neural net produce stable chord spectrograms from images.

37th Conference on Neural Information Processing Systems (NeurIPS 2023).

2 Data Conditioning Pipeline

Our goal is to transform a low-resolution digit image, $I \in \mathbb{R}^{28\times 28}$, into a high-resolution spectrogram $Y \in [0,1]^{1008\times 1008}$ that encodes the full harmonic structure (within feasible auditory limits) of major triad chords. This process involves several steps: upscaling, Gaussian weighting, frequency selection, Gaussian smearing/smoothing, noise addition, and final normalization. In what follows, we describe each step mathematically and explain its necessity.

2.1 Upscaling

The input image I is upscaled from 28×28 to 1008×1008 using a resampling function. Formally, if we denote the upscaling operator as $\mathrm{Resample}(\cdot)$, then:

$$I_{\text{upscaled}}(r,c) = \mathrm{Resample}(I, 1008) \quad \text{for } r = 1,\ldots,1008,\ c = 1,\ldots,1008.$$

This operation is essential because the target spectrogram requires a high spatial resolution to clearly delineate the harmonic frequency lines.

2.2 Gaussian Weighting

After upscaling, we apply Gaussian weighting to emphasize intensity in a controlled manner. The one-dimensional Gaussian function is defined as:

$$G(x;\mu,\sigma) = \frac{1}{\sqrt{2\pi}\,\sigma}\exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right).$$

For each row index r, we compute a weight $G(r;\mu,\sigma)$ and multiply the corresponding pixel intensity:

$$I_{\text{weighted}}(r,c) = I_{\text{upscaled}}(r,c)\cdot G(r;\mu,\sigma).$$

This weighting simulates the natural decay in harmonic amplitude that one might observe in real instruments, and acts as a fade-in/fade-out for the audio sample.

2.3 Frequency Selection

The next step is to isolate only the rows corresponding to the desired frequencies—the fundamental frequency f_0, its major third (approximately $1.25f_0$), and its perfect fifth (approximately $1.5f_0$). We determine the fundamental frequency from the digit label using:

$$f_0 = 27.5 \times 2^{\frac{\text{label}}{12}},$$

where 27.5 Hz is the lowest note (A0) in our musical scale. Then, we define the frequency selection operation as:

$$I_{\text{freq}}(r,c) = \begin{cases} I_{\text{weighted}}(r,c), & \text{if } r \in \{\, f_0,\ \lfloor 1.25f_0 \rfloor,\ \lfloor 1.5f_0 \rfloor \,\}, \\ 0, & \text{otherwise.} \end{cases}$$

This selective masking guarantees that the target spectrogram exhibits nonzero values only at the harmonic frequencies, which is crucial for generating musically pleasing chords.

2.4 Gaussian Smearing and 2-D Smoothing

To mimic the natural spread of harmonics and reduce abrupt transitions, we apply a 1-D Gaussian smearing along each dimension. For the vertical direction:

$$I_{\text{smear}}(r,c) = \sum_k I_{\text{freq}}(k,c)\cdot G(r-k;\sigma_{\text{smear}}),$$

where σ_{smear} controls the extent of the smear. Subsequently, a 2-D Gaussian smoothing is applied:

$$I_{\text{smooth}}(r,c) = \sum_{i,j} I_{\text{smear}}(i,j)\, G(r-i,c-j;\sigma_{2D}),$$

with

$$G(x,y;\sigma_{2D}) = \frac{1}{2\pi\sigma_{2D}^2}\exp\left(-\frac{x^2+y^2}{2\sigma_{2D}^2}\right).$$

This operation further homogenizes the image, simulating the spread of sound energy in a real instrument.

2.5 Perlin Noise Addition

To avoid overly smooth outputs (which can lead to overfitting) and to simulate subtle natural variations, we introduce Perlin noise. Let $N(r, c)$ denote the Perlin noise field generated over the 1008×1008 domain. The noisy image is given by:

$$I_{\text{noise}}(r, c) = I_{\text{smooth}}(r, c) \times N(r, c).$$

An additional overlay step may be performed where a scaled noise image is added:

$$I_{\text{final}}(r, c) = I_{\text{noise}}(r, c) + \lambda \, N_{\text{overlay}}(r, c),$$

with λ controlling the overlay strength.

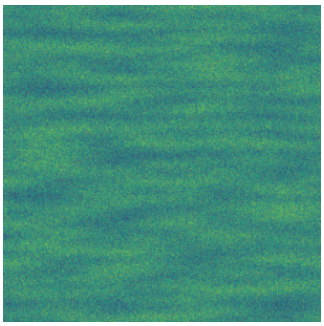

Figure 1: Example of perlin noise filter generated.

2.6 Final Normalization

Finally, to ensure the spectrogram values lie in the range $[0, 1]$, we perform normalization:

$$Y(r, c) = \frac{I_{\text{final}}(r, c) - \min(I_{\text{final}})}{\max(I_{\text{final}}) - \min(I_{\text{final}}) + \epsilon},$$

where ϵ is a small constant (e.g., 1×10^{-8}) to avoid division by zero.

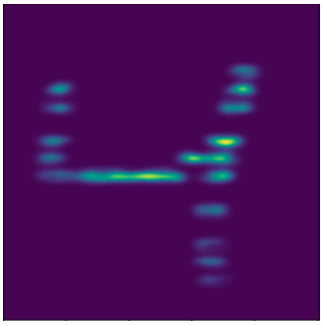

Figure 2: Final conditioned spectrogram.

Summary

In summary, the data conditioning pipeline transforms a low-resolution digit into a spectrogram by:

- Upscaling the image to provide sufficient resolution.
- Applying Gaussian weighting to simulate natural amplitude decay.
- Isolating frequencies corresponding to the fundamental and its harmonic multiples.
- Using Gaussian smearing and smoothing to avoid abrupt transitions.
- Introducing Perlin noise to mimic natural texture.
- Normalizing the final output for consistent network training.

These operations ensure that our training targets encapsulate the desired chord structure, making them an effective supervision signal for our U-Net model.

3 U-Net Architecture for Spectrogram Generation

Our objective is to learn a mapping

$$f_\theta : \mathbb{R}^{1\times 28\times 28} \to \mathbb{R}^{1\times 1008\times 1008}$$

from a low-resolution digit image to a high-resolution spectrogram that encodes the harmonic content corresponding to the digit (and, by extension, a musical note).

Our U-Net architecture is structured into three main stages: an encoder, a bottleneck, and a decoder with skip connections. Skip connections reintroduce spatial details lost during downsampling and have proven essential for learning the fine vertical frequency bands in our spectrogram outputs.

3.1 Encoder

The encoder extracts hierarchical features in two blocks. In each block a convolution is followed by a ReLU activation and downsampling via max pooling. We can summarize the operation in block i as

$$E_i = \text{MaxPool}\left(\text{ReLU}(W_i * X_i + b_i)\right),$$

where $X_1 = X$ (the input image) and $X_2 = E_1$. For example, after Block 1 the output has dimensions $64 \times 14 \times 14$ and after Block 2 it is $128 \times 7 \times 7$.

3.2 Bottleneck

At the bottleneck we increase the channel dimension to capture deeper features. LeakyReLU is utilized to prevent dead neurons:

$$B = \text{LeakyReLU}(W_3 * E_2 + b_3) \quad \in \mathbb{R}^{256\times 7\times 7}.$$

3.3 Decoder with Skip Connections

The decoder upsamples the latent representation back to the target resolution. Each upsampling block uses a transposed convolution followed by a convolution on the concatenation of the upsampled features with the corresponding encoder features. Mathematically, let the upsampled output be U_i and the corresponding encoder features be E_i'; then

$$D_i = \text{ReLU}\left(W_{d_i} * \text{Concat}(U_i, E_i') + b_{d_i}\right).$$

Here, the concatenation is defined as

$$\text{Concat}(A, B) \in \mathbb{R}^{(C_A+C_B)\times H\times W},$$

which allows the decoder to access both the coarse, global information and the fine, local details. After two such upsampling blocks, we bilinearly interpolate the output to 1008×1008 and apply a final convolution and activation to yield

$$Y = \text{LeakyReLU}\left(W_f * \text{Upsample}(D) + b_f\right),$$

with Y normalized in the range $[0, 1]$. Originally, a sigmoid activation function was used in the final layer, but LeakyReLU yielded better results in testing.

3.4 Rationale and Impact of Skip Connections

Skip connections directly add high-resolution features from the encoder into the decoder. This mechanism mitigates the loss of spatial detail due to pooling and enables the network to generate sharper spectrogram outputs with well-defined vertical lines at the harmonic frequencies. Our experiments confirmed that without these concatenations, the outputs became overly smooth and blurry.

4 Loss Functions and Training Strategy

4.1 Loss function evolution

We experimented with many losses:

- **MSE loss**:

$$\mathcal{L}_{\text{MSE}} = \frac{1}{N}\sum_{i=1}^{N} \|\hat{Y}_i - Y_i\|_2^2.$$

- **L1 (MAE) loss**: Encourages sharper solutions but can be slower to converge:

$$\mathcal{L}_{\text{L1}} = \frac{1}{N}\sum_{i=1}^{N} \|\hat{Y}_i - Y_i\|_1.$$

- **Huber loss**:

$$\mathcal{L}_{\text{Huber}}(\delta) = \begin{cases} \frac{1}{2}(\hat{Y} - Y)^2, & \text{if } |\hat{Y} - Y| < \delta \\ \delta\,|\hat{Y} - Y| - \frac{1}{2}\delta^2, & \text{otherwise.} \end{cases}$$

  We used $\delta = 0.5$ or 1.0 in some runs.

- **MS-SSIM** (4): We tried $1 - \text{MS-SSIM}(\hat{Y}, Y)$ to measure structural similarity in an attempt to preserve the vertical chord lines.
- **Frequency domain (STFT) loss**: We briefly attempted to compare short-time Fourier transforms of $\hat{Y}$ vs. Y, but found it complicated training, often leading to instability. This is probably related to the custom spectrogram implementation we used.
- **Custom Wav Function Loss**: $\mathcal{L} = \|\text{predicted\_wav} - \text{target\_wav}\|^2$ We implemented a loss function based on our spectrogram-to-wav conversion function, but similar to the STFT loss, it was not stable

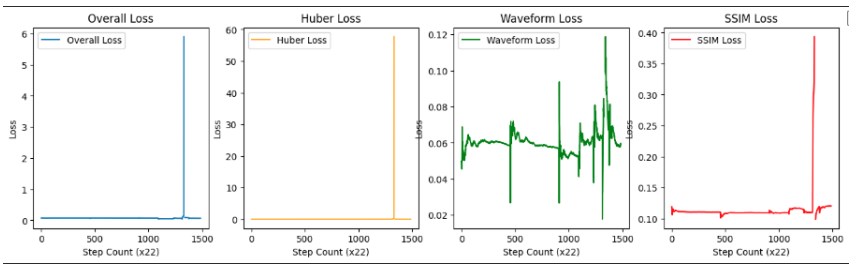

Figure 3: Increasing frequency loss weight during training led to exploding gradients and instability

We tested linear combinations, e.g.

$$\mathcal{L} = \alpha \cdot \mathcal{L}_{\text{Huber}} + \beta \cdot (1 - \text{MS-SSIM}).$$

We also tried adjusting the ratio over time to do a form of curriculum on SSIM or frequency losses.

4.2 Training details and commentary

Mixed precision. We used `torch.cuda.amp` and `torch.GradScaler` to speed up training. This improved throughput significantly, but introduced some hyper-sensitivity to learning rates at times.

Hyperparameters. We tried LR $= 3 \times 10^{-5}$ up to 1.5×10^{-2} in different phases, occasionally "bumping" the LR to escape local minima. Batch sizes of 2–32 were tested, with 2 used for memory constraints in certain runs.

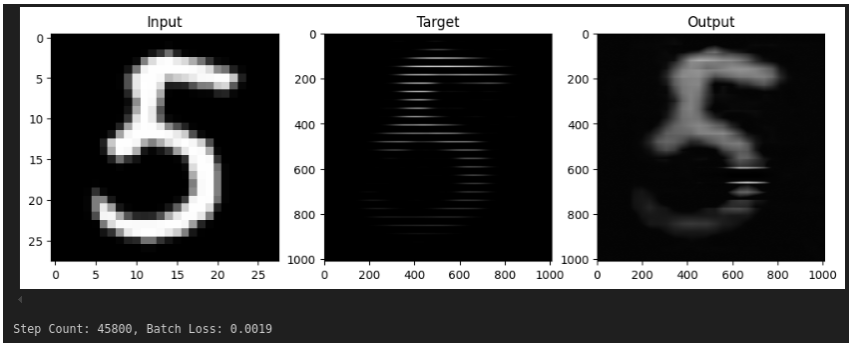

Figure 4: Consistent blurring/ bright streak pattern along with a very low loss function suggested we were stuck in a local minima

Observations from logs. - *At 10k–50k steps:* The chord lines partially formed, but some digits were producing consistently incorrect labels, and improvement was minimal and loss was low, suggesting the model was stuck at a local minima.

- *At 50k–80k steps:* Adding MS-SSIM scaled by certain factors improved structural line continuity but introduced blur or random artifacts. - *At 80k–100k steps:* We found partial local minima; some digits became correct, others had large black or cloudy areas. - *After 100k steps:* Further improvements were minor. We tried "curriculum" focusing on certain digits, changing LR, reintroducing the full dataset, etc.

Eventually, we decided to freeze the model around 150k steps, with a modest final performance: *most* digits produce correct chord lines, though some remain imperfect.

Figure 5: Sample of final checkpoint outputs (inputs not present in dataset)

4.3 Loss Monitoring

Loss was manually reviewed throughout the training process using intermittent sampling and loss functions of overall loss, and relevant loss functions.

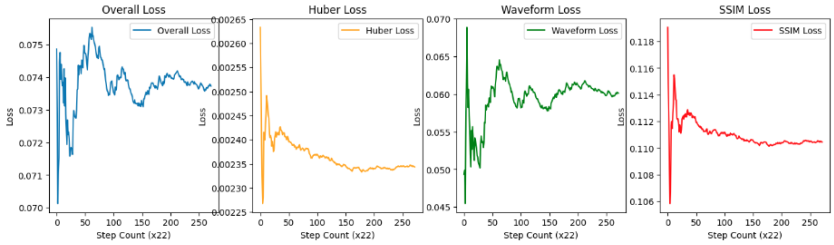

Figure 6: Sample taken mid-training. Huber and SSIM loss decreased steadily throughout training, but phase related loss functions failed to converge.

4.4 Backprop equations

In each layer, if $z^\ell = W^\ell * a^{\ell-1} + b^\ell$, $a^\ell = \sigma(z^\ell)$, the chain rule yields:

$$\frac{\partial \mathcal{L}}{\partial W^\ell} = \sum_i \frac{\partial \mathcal{L}}{\partial a_i^\ell} \frac{\partial a_i^\ell}{\partial z_i^\ell} \frac{\partial z_i^\ell}{\partial W^\ell}.$$

We rely on PyTorch for actual gradient updates:

$$\theta \leftarrow \theta - \eta \nabla_\theta \mathcal{L}(\theta).$$

Skip connections pass additional partial derivatives from deeper layers to shallower features, preserving spatial detail (1).

5 Inference and Audio Generation

Inference. After training, we load the best checkpoint. For a new 28×28 digit image x, we feed it into the U-Net to get $\hat{y} \in \mathbb{R}^{(1008\times1008)}$. We then normalize to $[0, 255]$.

Audio Synthesis. We interpret row r as frequency r + offset, or use a direct mapping if r matches a chord row. Then for each column c, we have amplitude $A_{r,c} = \hat{y}[r, c]/255$. The final wave is

$$\text{combined}(t) = \sum_{r=0}^{1007*samplerate} A_{r,c} \sin(2\pi\, r\, t + \phi_r),$$

for t in the appropriate time window. Because only chord lines are bright, the user hears a stable major triad chord (2). Our code implements `convert_to_wav`, resampling and phase randomization.

6 Experimental Results and Discussion

Visual results. Figure 5 We aimed to produce an output with a similar likeness to the input, with frequency characteristics related to the fundamental note assigned to that input label. Outputs produced consistently reproduces some banding behavior. For certain notes e.g. 0/A, the banding was accurate, but others produce consistent artifacts and noise.

Audio quality. Listening to `predicted_audio.wav` reveals a clear major chord with some ambient noise. Some amplitude mismatch occurs, but overall the chords are audible. Using only fundamental rows ensures no dissonant partials.

Loss curve observations. We saw the combined $\text{Huber} + \text{MS-SSIM}$ drop quickly but then plateau. The model seemed to "lock in" certain chord lines. Tweaking the ratio gave partial improvements but introduced artifacts. Phase related loss functions were very volatile or plateaued, likely due to the custom spectrogram implementation.

Local minima and advanced attempts. We introduced frequency-domain losses to ensure label d truly mapped to frequency f_0. However, that caused training instability or overshadowed structural learning. We tried "curriculum learning" for a single digit, then reintroducing the others, but some digits still produced incomplete lines. We found a final compromise: we rely mostly on Huber + a small SSIM portion, which yields decent chord lines for most digits.

We attempted to increase add an additional convolutional layer to our neural network, but failed to see significant convergence after 100,000 steps.

In summary. Despite the complexities, the pipeline consistently yields a major triad chord from any digit image. The chord lines in the spectrogram are easily identified, and the audio is stable. While we have not reached perfect digit-frequency alignment for all digits, the system demonstrates the feasibility of image-to-chord transformations at scale.

7 Conclusion and Future Directions

We have built a pipeline that turns 28×28 digits into large spectrograms containing a major triad chord. We overcame numerous training hurdles:

- Replacing ESRGAN with simpler `resample` for upscaling.
- Integer rounding for $1.25f_0$ and $1.5f_0$ so chord lines do not vanish.
- Attempting MSE, L1, Huber, MS-SSIM, partial frequency losses, plus dynamic weighting of these losses.
- Curriculum learning for certain digits, LR "bumps," and reintroducing the full dataset to escape local minima.

Ultimately, the final model often produces correct triad lines for each digit label. The audio from `convert_to_wav` is pleasing. In the future, we plan to:

1. Extend from major triads to more chord types (minor, dominant 7th, etc.).
2. Implement automatic rather than manual learning rate scheduling, and dynamic variable updates which can be changed during training
3. Investigate advanced frequency alignment methods that do not destabilize training such as Sobel filtering.
4. Integrate real-time updates: user sketches an image, instantly hears chord changes.
5. Implement more typical spectrogram reconstruction, e.g. Mel Spectrogram, to increase efficacy and compatibility with existing tools

Overall, this project demonstrates a creative synergy of image processing, deep learning (U-Net), and chord-based audio generation, offering an accessible path to convert any digit into a stable major triad chord.

Acknowledgments and Disclosure of Funding

Acknowledgments. We thank the ENGS 106 staff for their guidance. We also appreciate the HPC resources that allowed extensive experimentation with large-scale spectrogram outputs.

References

[1] O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention*, 2015.

[2] R. Parncutt. *Harmony: A psychoacoustical approach*. Springer, 1989.

[3] K. Perlin. An image synthesizer. *ACM SIGGRAPH Computer Graphics*, 19(3):287–296, 1985.

[4] Z. Wang, E. P. Simoncelli, and A. C. Bovik. Multiscale structural similarity for image quality assessment. In *The 36th Asilomar Conference on Signals, Systems & Computers*, 2003.