Speech time-scale modification using GANs

By Jayin Khanna
Supervisor: Dr Prasanta Kumar Ghosh
Mentors: Haritha Neelapuja & Jesuraj

1. Defining and the Problem
2. ScalerGAN
3. Prerequisite: GANs
4. Architecture

Problem statement

Alter speech rate (slow down/speed up) while preserving:

- Speaker identity i.e.
  - Timbre
  - Pitch
- Intelligibility and,
- Naturalness.

Upshot: **scaling the time axis** of the signal while attempting to keep the **spectral content** (pitch, timbre) intact.

Challenges:

1. Traditional signal processing methods (e.g. PSOLA, WSOLA, Phase Vocoder) cause artifacts at extreme rates.

2. No supervised data: Parallel utterances at different rates are unavailable.

Goal: Develop a deep learning solution for high-quality TSM without paired training data.

Proposed solution: ScalerGAN

Unsupervised GAN framework inspired by CycleGAN and InGAN, adapted for speech spectrograms:

A new deep learning algorithm for TSM that can speed up or slow down speech signals at a given rate.

Key Components:

1. **Generator (G)**:
   - Input: Speech spectrogram + desired rate r.
   - Output: Time-scaled spectrogram.
2. **Multi-Scale Discriminator (D)**:
   - 5 sub-discriminators at different spectrogram scales.
   - Classifies "real vs. synthetic" spectrograms.
3. **Decoder**:
   - Reconstructs original spectrogram from time-scaled output using G with rate r^{-1} (cycle consistency).

Generative Adversarial Networks (GANs): Core Idea

The Min-Max Game
A GAN consists of two neural networks locked in a competitive game:

Real: 1
Fake: 0

- **Generator (G):** The forger. Tries to create fake data that looks real.
- **Discriminator (D):** The detective. Tries to distinguish real data from fakes.

Their goals are directly opposed, formalized by this Min-Max Objective:

The value function is:

$$V(D,G) = \mathbb{E}_{x \sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))]$$

D(x): D's estimate that real data *x* is real (goal: ~1).
D(G(z)): D's estimate that fake data G(z) is real (goal: ~0 for D, ~1 for G).

The Ultimate Goal: Distribution Matching
The generator doesn't just memorize data; it learns to model the underlying probability distribution (*p_data*) of the real data.

- **Initial State:** Generator outputs random noise from a simple distribution (e.g., Gaussian).
- **Final State:** Generator transforms noise into data that lies on the same space as the real data.
- At convergence: Generator's distribution *p_g* ≈ *p_data*, and the discriminator is forced to guess (D(x) = 0.5 everywhere).

Training of GANs

Step-by-Step Training Loop

For each training iteration:

1. **Sample Data:**
   - Get a batch of real data *x* and a batch of random noise vectors *z*.
2. **Generate Fakes:**
   - Forward pass through Generator: G(z) = x_fake.
3. **Update the Discriminator (D):**
   - Zero D's gradients.
   - **Forward Pass:** Compute D(x) (real) and D(x_fake) (fake).

Calculate D's Loss:

```
L_D = -[log(D(x)) + log(1 - D(x_fake))]
```

   - • This loss is high if D is wrong (e.g., if D(x_fake) is high).
   - Backpropagate L_D to calculate gradients for D's parameters.
   - Update D's weights to become a better detective.
4. **Update the Generator (G):**
   - Zero G's gradients.
   - Generate new fakes: G(z) = x_fake (Forward pass again).
   - Re-run D on new fakes: D(x_fake) (D is now frozen).

Calculate G's Loss:

```
L_G = -log(D(x_fake))
```

   - • This loss is high if D correctly identifies the fake (D(x_fake) is low).
   - Backpropagate L_G to calculate gradients for G's parameters.
   - Update G's weights to become a better forger.

Generative adversarial Networks

Framework: Two neural networks contesting:

1. **Generator (G)**: Creates synthetic data (e.g., spectrograms).
2. **Discriminator (D)**: Distinguishes real vs. synthetic data.

Training Dynamics:

1. **Generator (G)**: Creates synthetic data from random noise.
2. **Discriminator (D)**: Distinguishes real data from synthetic data.

GAN Objective Function:

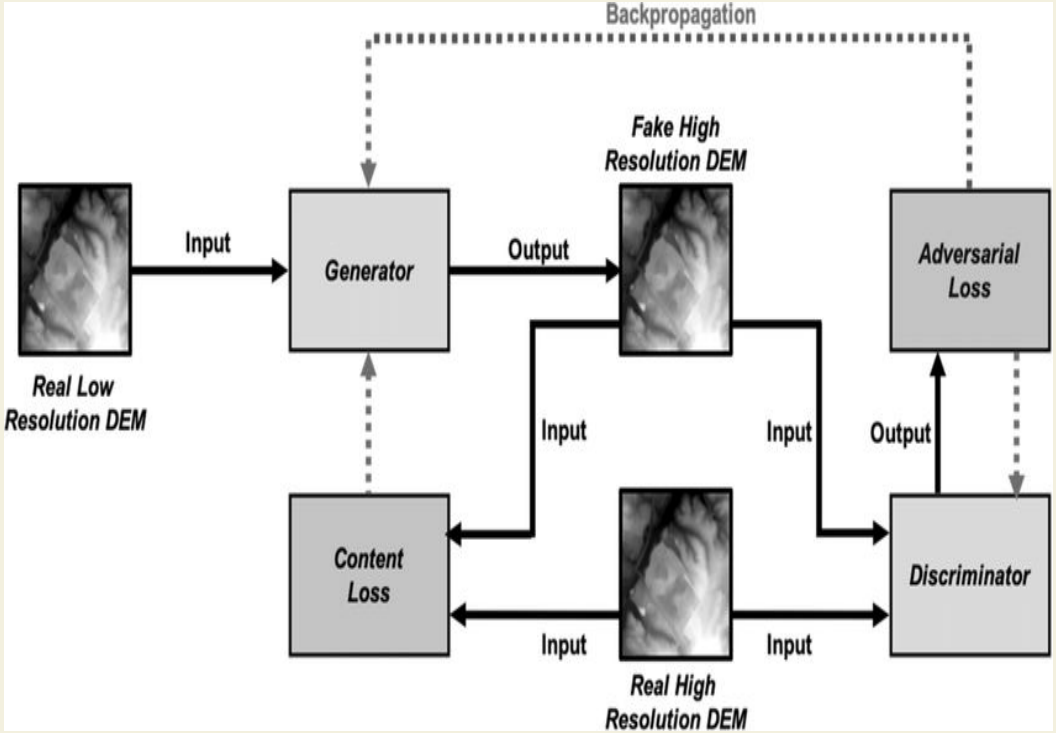
$$\min_G \max_D V(D,G) = \mathbb{E}_{x \sim p_{\text{data}}}[\log D(x)] + \mathbb{E}_{z \sim p_z}[\log(1 - D(G(z)))]$$

1. Discriminator maximises the probability of assigning the correct labels
2. Generator minimizes the log probability that the discriminator labels generated image as fake

GAN training

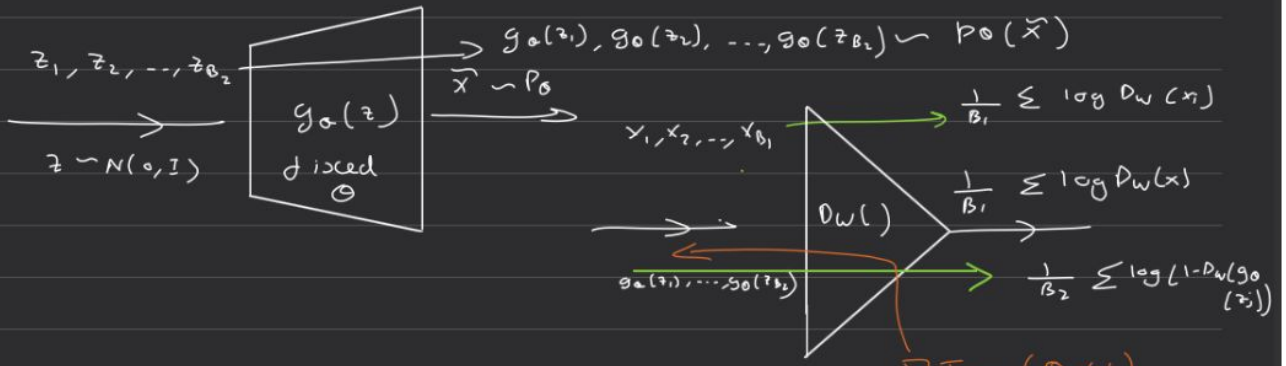
Repeat for each training iteration:

1. Sample minibatch of real data: x1,...,xn~p(data)
2. sample noise vectors: z1,...,zn~pz
3. Compute D(xi) and D(G(zi))
4. Update Discriminator using gradient ascent: $\nabla D[\log D(x)+\log(1-D(G(z)))]$
5. Update Generator using gradient descent: $\nabla G[-\log D(G(z))]$



*) TO train the discriminator:

→ keep Θ constant, $D = \{x_1, x_2, \ldots, x_n\}$



$B = \{x_1, \ldots, x_{B_1}\} \subset D$

one update: $w^{t+1} \leftarrow w^t + \alpha_1 \nabla_w J_{GAN}$

$$J_{GAN}(\theta, w) = \left[\frac{1}{B_1} \sum_{i=1}^{B_1} \log D_w(x_i) + \frac{1}{B_2} \sum_{j=1}^{B_2} \log(1 - D_w(g_\theta(z))) \right]$$

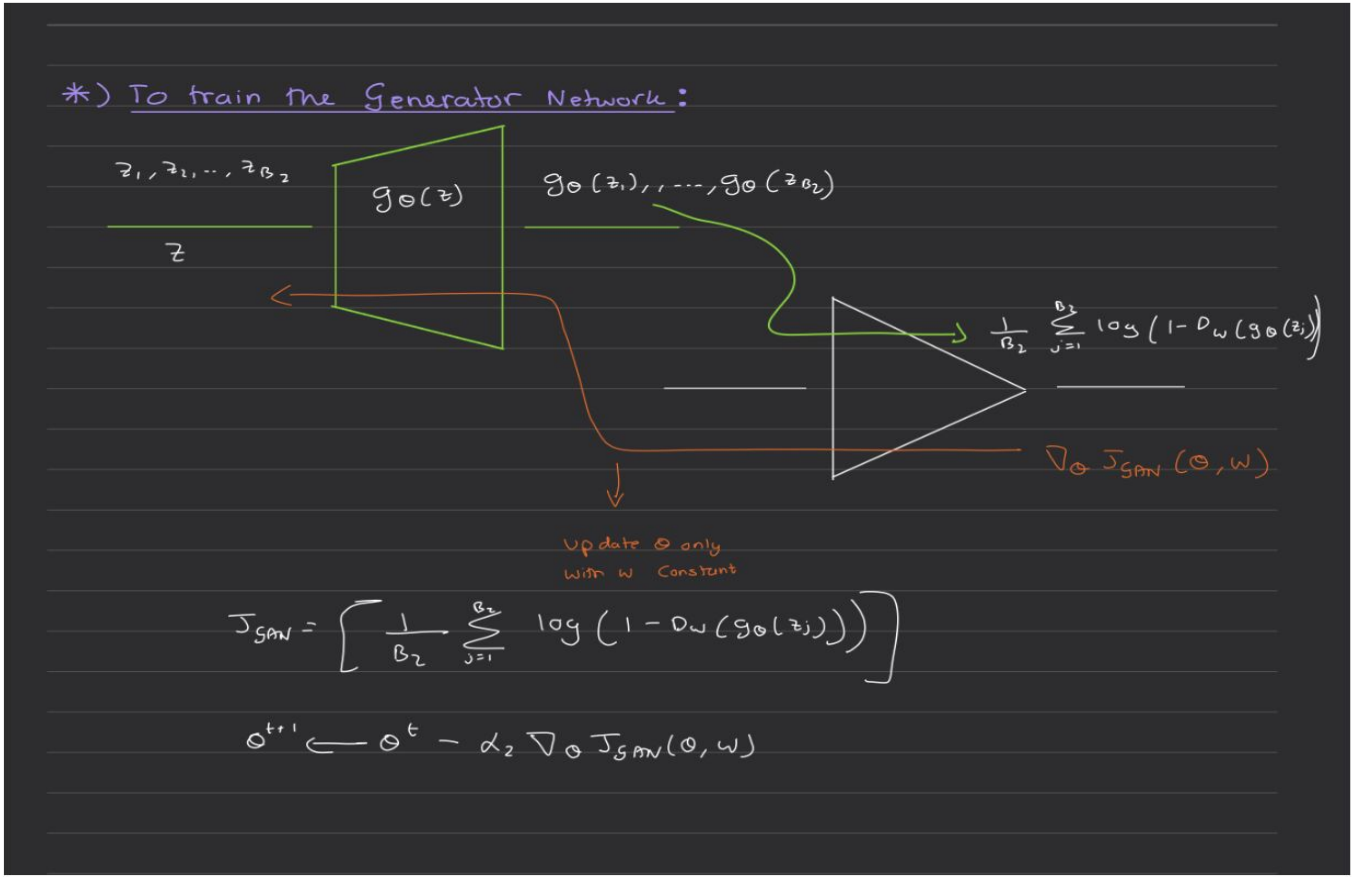


Figure 3: Train Generator network from notes

ScalerGAN

Goal: Alter speech speed (slow down/speed up) while preserving:
- Speaker identity (timbre, pitch).
- Intelligibility and naturalness

Input:

A Mel-spectrogram **matrix X ∈ ℝ^(F×N)**
• *F*: number of frequency bins (e.g., 80 Mels)
• *N*: number of time frames
• Desired scaling rate **r** (*r > 1*: speed up, *r < 1*: slow down)

Output:

A scaled Mel-spectrogram **Y ∈ ℝ^(F×M)**
• $M = \lfloor N \cdot r \rfloor$ new time frames

Learn a generative function G : (X,,R) → X* that given a finite-length sequence and the desired rate r will generate a finite sequence with a scaled duration according to the given rate r.

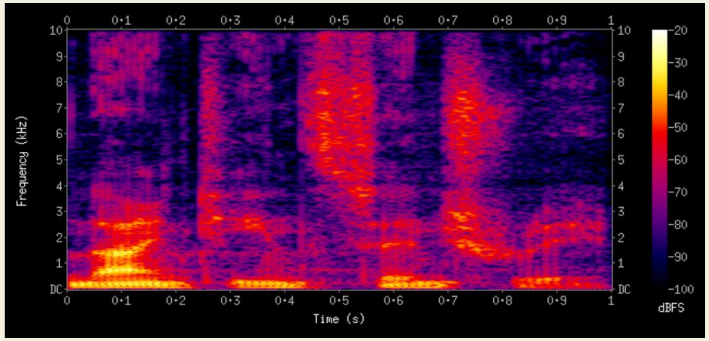
The most trivial implementation of the function G is by resampling the original signal x
Problems? Loses speaker identity Distorts pitch Produces artifacts

Why Mel-Spectrograms for Speech TSM

What is a Mel Spectrogram?

A visual representation of sound that shows:

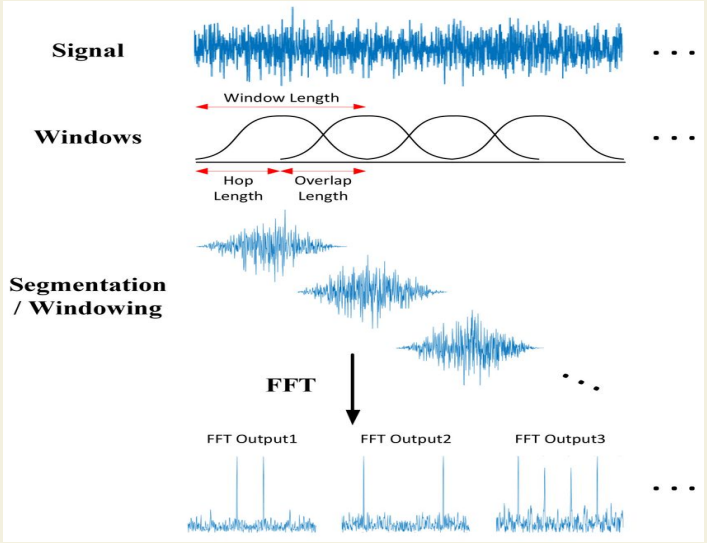
- **Time** on the horizontal axis (when sounds occur).
- **Frequency** on the vertical axis (which pitches are present).
- **Intensity/Energy** shown by color (brighter = louder).



How is it Made?

The process converts a raw waveform into a Mel Spectrogram:

1. **Framing & Windowing:** Chop the audio into short, overlapping segments (e.g., 25ms).
2. **Fourier Transform (STFT):** Calculate the frequency spectrum for each segment.
3. **Mel Scaling:** Apply a filter bank that warps the frequencies to the **Mel scale**, which mimics human hearing.
4. **Log Compression:** Convert amplitude to a logarithmic (dB) scale, as we perceive loudness logarithmically.



Why Use Mel-Spectrograms as inputs?

5. **Perceptually Relevant:** The Mel scale emphasizes crucial speech frequencies (like formants) and de-emphasizes less important ones, just like the human ear does.
6. **Reduced Dimensionality:** They provide a compact, information-rich summary of the audio, making the model's job more efficient than using raw waveforms or linear spectrograms.

Overall Architecture of the Model

1. Generator (G)

Input: A speech spectrogram $\bar{x}$ and a desired time-scaling rate r

Operation:

- Applies a linear interpolation on the time axis by factor r
- Passes the interpolated spectrogram through a U-Net to refine and enhance it

Output: A time-scaled spectrogram $\bar{y}$ corresponding to the new rate r

2. Multi-Scale Discriminator (D)

Goal: Distinguishes between real and generated (time-scaled) spectrograms

Design:

1. Consists of multiple sub-discriminators, each operating at a different temporal scale
2. Uses downsampling and patch-level classification to assess local realism at various resolutions

Training Signal: Guides the generator to produce spectrograms that are indistinguishable from real ones

3. The Cycle-Consistency Decoder

Input: Takes the the time-scaled spectrogram $\bar{y}$ back and the inverse of the time rate i.e. r^{-1}

Process:

- The time-scaled spectrogram $\bar{y}$ is passed back to the same generator G with an inverse rate 1/r
- Generates a reconstructed spectrogram $\bar{x}'$

Objective: The reconstructed spectrogram should match the original input:

$G(G(X, r), 1/r) \approx X$

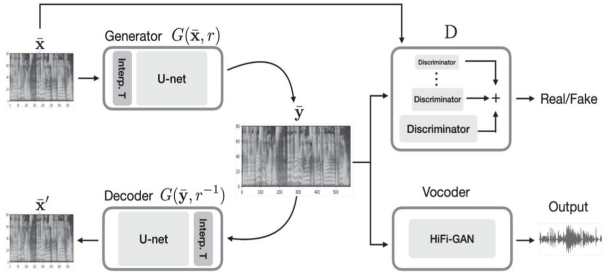


Fig. 1. The ScalerGAN model. The generator G gets a spectrogram $\bar{\mathbf{x}}$ and a rate r as input, and generates a new time-adjusted spectrogram $\bar{\mathbf{y}}$. During training, the resulting spectrogram is further processed by a discriminator D that gets as input either $\bar{\mathbf{x}}$ or $\bar{\mathbf{y}}$ and decides whether the input is real or fake. Parallel to that, a decoder gets $\bar{\mathbf{y}}$ as input and tries to reconstruct $\bar{\mathbf{x}}$ back from it, where the decoder is implemented using the same generator G. At inference time, we use a vocoder to convert the spectrogram into a waveform.

The Generator

What it does

- Core transformation unit
- Takes an input spectrogram and target rate
- Produces a **time-scaled spectrogram Y = G(X, r)**

How it works

- **Input:** Mel-spectrogram **X** and desired rate **r**
- **Architecture:** U-Net with skip connections
- **Output:** New spectrogram **Y**, length scaled by **r** ($M = \lfloor N \cdot r \rfloor$)
- **Key Function:** Reshapes the temporal axis while preserving frequency content & harmonic structure

---

U-Net Architecture

1. Encoder: Hierarchical Feature Extraction

- **Purpose:** Projects input spectrogram into a latent feature space
- **Process:** Strided convolutions ↓ temporal resolution, ↑ channel depth (feature richness)
- **Why for TSM?** Learns abstract, contextual representations (phonemes, syllables)

2. Bottleneck: High-Level Representation Learning

- **Purpose:** Core transformation zone for reorganizing features
- **Process:** 6 residual blocks apply non-linear transformations
- **Why for TSM?** Interprets scaling rate **r** and plans temporal expansion/contraction
- Residual connections → stable deep training

3. Decoder: Temporal Upsampling & Reconstruction

- **Purpose:** Synthesizes output spectrogram of target length **M**
- **Process:** Transposed convolutions ↑ temporal resolution, ↓ channel depth
- **Why for TSM?** Performs temporal resampling while reconstructing structure from bottleneck features

4. Skip Connections: Preserving Spectral Fidelity

- **Purpose:** Bypass bottleneck → transfer encoder features directly to decoder
- **Why critical for TSM?**
  - Without: outputs are temporally correct but spectrally blurry
  - With: outputs retain sharp harmonic structures & formants → naturalness + speaker identity preserved

The Discriminator

Core Concept: Why Multi-Scale?

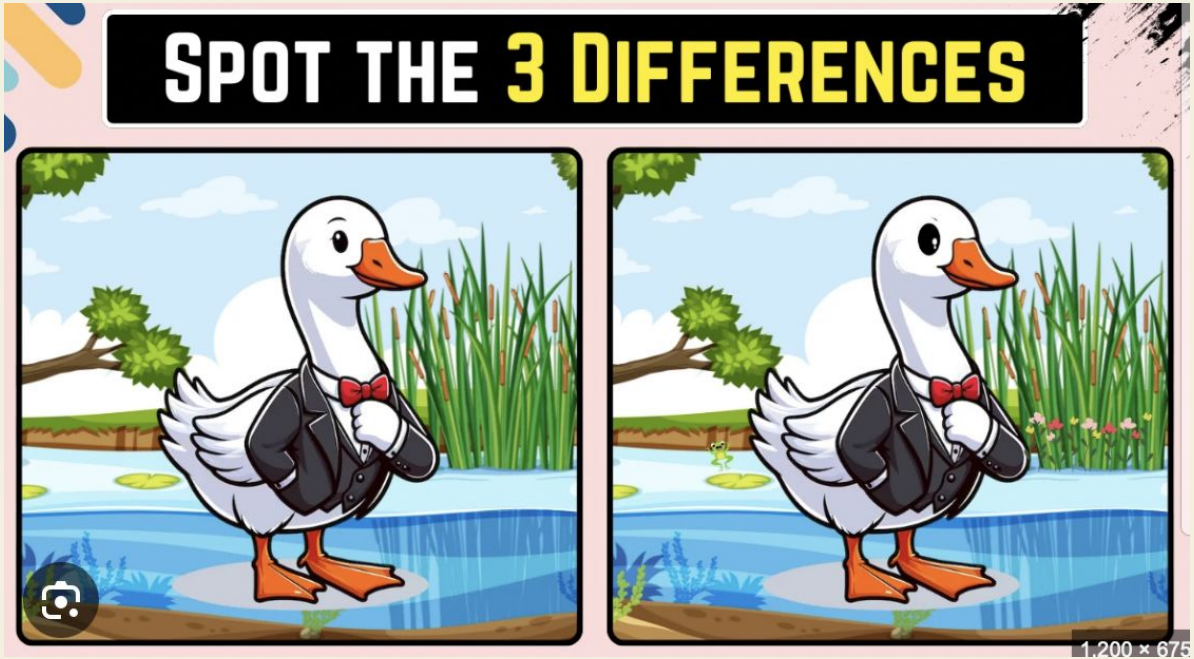
- **Goal:** Ensure generated spectrogram **Y = G(X, r)** is realistic across all time–frequency resolutions
- **High-Resolution (original scale):**
  - Judges fine details → formant sharpness, consonant clarity, harmonic texture
  - Catches local blurring/noise
- **Low-Resolution (downscaled):**
  - Judges macro-structure → rhythm, prosody, energy contours
  - Catches unnatural pauses, rhythm issues
- **Inspiration:** From InGAN → realism across spatial frequencies; directly applicable since spectrograms are 2D time–frequency images

Architecture: How It's Implemented

- **Discriminator D** = ensemble of **K** sub-discriminators {D_1, D_2, …, D_K}
- **Input Processing:**
  - D_1 → original resolution
  - D_2 … D_K → progressively downsampled (avg pooling, factors 2, 4, …)
- **Sub-Discriminator Design:**
  - Each D_k is PatchGAN-style classifier
  - Strided convolutions for downsampling
  - Outputs 2D matrix of probabilities (not a single scalar)

PatchGAN. Why?

- Judges spectrograms patch-by-patch, not globally
- Each matrix element = probability that local patch is real
- Enforces local + global coherence → suppresses artifacts in structured data like spectrograms


SPOT THE 3 DIFFERENCES
1,200 × 675

Multi-Scale Discriminator: Exact Data Flow & Patchwise Analysis

Initial Conditions:

- Real Spectrogram (Target): $\mathbf{X_{real}}$
- of shape [1, 80, 256] (Channels, Freq, Time).
- Fake Spectrogram (Input): Y_fake = G(X, 1.5)
  of shape [1, 80, 384].
- Scaling Factors: \in {1, 1.2, 1.2^2 1.44, 1.2^3 , 1.2^4 }.
  These are the factors for spatial downscaling.

Step 1: Define Target Size: The target size for the input to discriminator `D_s` is the **original spatial size divided by s**.

- For `X_real` (256×80): `T_s(X_real)` resizes it to `[256/s, 80/s]`.
- For `Y_fake` (384×80): `T_s(Y_fake)` resizes it to `[384/s, 80/s]`.
- Example: For s=1.44, `X_real` is resized to ~[178, 56] and `Y_fake` to ~[267, 56].*

The Transformation Operator (T): This is typically a **bilinear interpolation** (or bicubic). Its purpose is to **smoothly and differentially downscale** the spectrogram, simulating a lower-resolution "view".

The final discriminator is a weighted sum over all outputs,

$$D(\bar{\mathbf{x}}) = \sum_s \alpha_s T^{s\times s}\left(D^s(T^{s^{-1}\times s^{-1}}(\bar{\mathbf{x}})\right),$$

Step 2: Patchwise Probability Calculation per Sub-Discriminator

Each sub-discriminator `D_s` is a **convolutional network** (e.g., a series of 4×4 strided conv layers).

- **Input:** The resized spectrogram from `T_s` of size `[1, Freq/s, Time/s]`.
- **Process:** The convolutional network processes this input. The key design is that the **final layer is a convolution (not a linear layer)**.
- **Output:** The output of `D_s` is not a single number. It is a **2D feature map** (e.g., of size `[1, 4, 8]`). Each value in this `[4, 8]` grid corresponds to the **logit** for a specific "patch" or receptive field in the *resized* input.
- **Patchwise Reality:** This means `D_s` produces a **dense map of "realness" scores**. It can identify that, e.g., the patch covering timeframes 10-20 and frequency bins 40-50 looks fake, while the rest looks real. This provides a incredibly rich and precise gradient signal for the generator.

Step 3: Combining Scales via Weighted Sum

The outputs of all 5 sub-discriminators are not independent; they are combined into a final, unified "realness" map.

1. **Upsample Discriminator Outputs:** The output of each `D_s` (e.g., the `[4, 8]` grid) To combine them, we must **upsample each output back to the *original* resolution** of the spectrogram being judged.
   - We apply the inverse transformation: `T_s^{-1}`.
   - This results in 5 matrices (`0_s=1, 0_s=1.2,.., 0_s=2.07`), all at the original resolution (e.g., `[80, 384]` for `Y_fake`), representing the realness score for every potential patch.
2. **Learnable Weighted Sum:** The final discriminator output is a weighted sum of these upsampled maps:
   - `α_s` are **learnable parameters**.
   - The network learns **how much to trust the judgment of each scale**. For example, it might learn that the finest scale (s=1) is most important for judging harmonic sharpness, while a mid-scale (`s=1.44`) is best for judging rhythmic consistency.

Training Objective: Least-Squares GAN (LSGAN) Loss

Discriminator's Goal: Maximize $L^{Ls}(G, D)$

The discriminator **D** aims to maximize its ability to distinguish real from fake. This is done by minimizing the following loss function:

$$L^{Ls}(G, D) = E_{\bar{x}\sim pdata(\bar{x})}[(D(\bar{x}) - J)^2] + E_{\bar{x}\sim pdata(\bar{x}),\, r\sim p(r)}[(D(G(\bar{x}, r)) - 0)^2]$$

Term 1: Real Loss (L_real)

$$E_{\bar{x}\sim pdata(\bar{x})}[(D(\bar{x}) - J)^2]$$

- **Calculation:** For a batch of real spectrograms, the discriminator's output D(real) is a matrix of patch-wise predictions. We calculate the Mean Squared Error (MSE) between this matrix and a matrix of all ones (J).
- **Interpretation:** This loss is minimized when D(real) outputs values close to 1 for every patch. It teaches the discriminator to assign the "real" label to ground-truth data.

Term 2: Fake Loss (L_fake)

$$E_{\bar{x},r}[(D(G(\bar{x}, r)) - 0)^2]$$

- **Calculation:** For the same batch of real spectrograms, we pass them through the generator G with a random rate r to get fake spectrograms G(real, r). The discriminator's output D(fake) is compared to a matrix of all zeros (0) using MSE.
- **Interpretation:** This loss is minimized when D(fake) outputs values close to 0 for every patch. It teaches the discriminator to assign the "fake" label to generated data.

Discriminator's Total Loss: L^D = L_real + L_fake

The discriminator's weights are updated to minimize L^D.

2. Generator's Goal: Minimize $L^{Ls}(G, D)$

The generator **G** aims to minimize the discriminator's ability to catch fakes. It does this by minimizing its own specific loss function:

$$\mathbf{L^G = E_{\bar{x},r}[(D(G(\bar{x}, r)) - J)^2]}$$

- **Calculation:** The generator's loss is calculated on the same batch of fake spectrograms. However, here we calculate the MSE between D(fake) and a matrix of all ones (J).
- **Interpretation:** This loss is minimized when D(fake) is fooled into outputting values close to 1. It teaches the generator to produce outputs that are indistinguishable from real data, according to the discriminator.

Generator's Adversarial Loss: $L^G\_adv = E[(D(G(\bar{x}, r)) - J)^2]$
The generator's weights are updated to minimize L^G_adv.

3. The Full Objective & Weight Update Process

The complete training goal is a minimax game with an additional constraint:

$$\mathbf{\min^G \max^D [\ L^{Ls}(G, D) + \lambda\ L^R(G)\ ],\ \lambda = 0.1}$$

Where $\mathbf{L^R(G)}$ is the Cycle-Consistency Reconstruction Loss (e.g., L1 loss between original and reconstructed spectrogram).

Reconstruction Loss

- Cycle consistency:
  We first time-scale x → y = G(x, r), then scale it back y → x′ = G(y, 1/r)
- Goal:
  Ensure that reconstructed x′ is close to the original x
- Why?
  Prevents mode collapse (e.g., G generating repetitive or blurry outputs)

The second loss function is associated with the decoder. We would like minimize the L_1 distance between $\bar{\mathbf{x}}' = G(\bar{\mathbf{y}}, r^{-1})$ and $\bar{\mathbf{x}}$ so as to prevent convergence to trivial solutions:

$$\mathcal{L}_R(G) = \|G(G(\bar{\mathbf{x}},\ r),\ r^{-1}) - \bar{\mathbf{x}}\|_1\ , \qquad (2)$$

Experiments and Results

1. Datasets & Preprocessing

- **Primary Dataset: LJSpeech**
  - 13,100 utterances (24 hrs), single female speaker, 22.05 kHz, 16-bit PCM
- **Generalization Test: VCTK Subset**
  - 28 speakers (14 M, 14 F) → Evaluates unseen voice adaptation
- **Feature Extraction (Mel-spectrograms):**
  - n_fft = 1024, hop_size = 256, n_mels = 80
  - fmin = 0, fmax = 8000 (Covers speech-relevant frequencies)

2. Experimental Setup

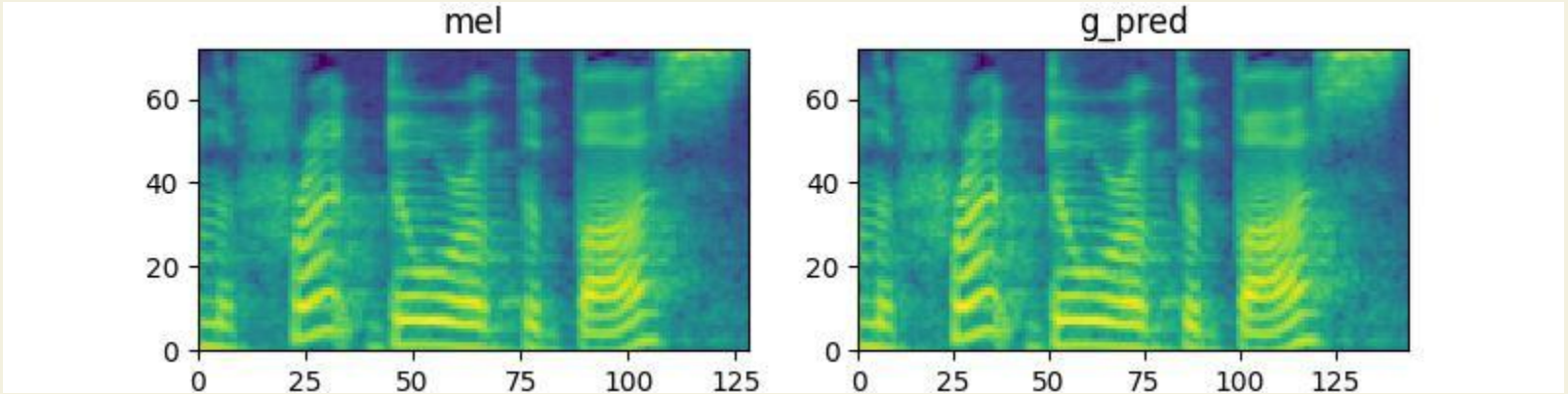
- **Training:** Random segments of 256 frames
- **Inference:** Full spectrograms
- **Architecture:**
  - Generator: U-Net with 6 residual blocks (bottleneck)
  - Discriminator: 5 sub-discriminators (multi-scale)
- **Hyperparameters:**
  - Batch Size = 24 | Learning Rate = 5e-5 (Adam, β_1=0.5, β_2=0.999)
  - Epochs = 500 | λ (Reconstruction Weight) = 0.1
- **Curriculum Learning:**
  - Scaling factor *r* gradually expanded from [1.0, 1.0] → [0.3, 1.8]
- **Stability Measures:**
  - Checkpoint resuming
  - Inverse steps added post-epoch 200 (cycle consistency)

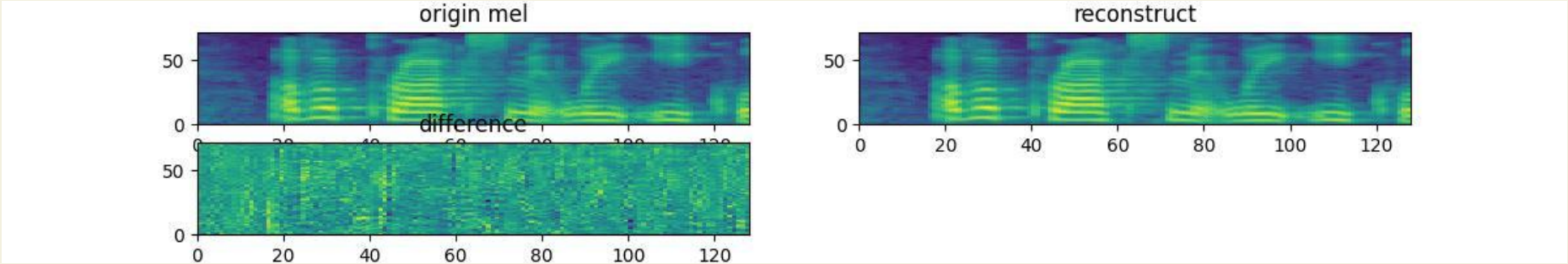
4. Key Results

- **Successful Convergence (Epoch 500):**
  - G_loss = 0.2486
  - D_loss_real = 0.2496
  - D_loss_fake = 0.2533
  - Rec_loss = 0.1111
- **Qualitative Outputs:**
  - Temporal Scaling → Generator adjusts duration while preserving harmonics (Fig. 4)
  - Cycle Consistency → Low reconstruction error (Fig. 5), content identity maintained
  - Loss Curves → Stable adversarial training (Fig. 6)
  - Generalization → Effective scaling on unseen VCTK speakers

4. Inference Phase

- At test time, the generated spectrogram $\bar{y}$ is converted back into an audio waveform using a neural vocoder (HiFi-GAN)


mel
g_pred
60
40
20
0
0
25
50
75
100
125


origin mel
reconstruct
difference
50
0
20
40
60
80
100
120

FastSpeech: Ideas for Content-Aware TSM

Limitations of Uniform Time-Scaling

ScalerGAN performs **content-agnostic scaling**, applying the same temporal transformation globally. This can introduce artifacts:

- **Over-compression** of long phonemes (e.g., vowels)
- **Over-extension** of short phonemes (e.g., plosives: /p/, /t/, /k/)
- **Loss of prosody**: Natural rhythm, stress, and pauses are not preserved
- **Reduced naturalness** despite perceptual realism in spectrograms

Proposed Solution: Integrate FastSpeech Components

To enable **phoneme-aware duration control**, we propose hybridizing ScalerGAN with two key modules from **FastSpeech**:

1. **Duration Predictor (DP)**
   - A lightweight convolutional network
   - **Predicts natural phoneme durations** (in frames) from input features
   - Trained using alignments from a pre-trained teacher model (e.g., Tacotron 2)
2. **Length Regulator (LR)**
   - **Expands/compresses phoneme-level features** based on predicted durations
   - Uses a trainable duration scale factor
   - (α) for global speed control

Example:

- Phonemes:
- [p,o,t]
- Durations:
- [2,5,3] → Expanded:
- $[p,p,o,o,o,o,o,t,t,t]$

Thank You