notebook

September 24, 2025

[57]:
```python
import os, torch, torchaudio, librosa, numpy as np
import torch.nn as nn
import torch.nn.functional as F
import matplotlib.pyplot as plt
import soundfile as sf
```

[58]:
```python
class ConvAutoencoder(nn.Module):
    def __init__(self, latent_dim=128):
        super().__init__()
        self.enc = nn.Sequential(
            nn.Conv2d(1, 32, 3, 2, 1), nn.ReLU(),
            nn.Conv2d(32, 64, 3, 2, 1), nn.ReLU(),
            nn.Conv2d(64, 128, 3, 2, 1), nn.ReLU(),
            nn.Conv2d(128, 256, 3, 2, 1), nn.ReLU(),
        )
        self.fc_enc = nn.Linear(256*8*14, latent_dim)
        self.fc_dec = nn.Linear(latent_dim, 256*8*14)

        self.dec = nn.Sequential(
            nn.ConvTranspose2d(256, 128, 4, 2, 1), nn.ReLU(),
            nn.ConvTranspose2d(128, 64, 4, 2, 1), nn.ReLU(),
            nn.ConvTranspose2d(64, 32, 4, 2, 1), nn.ReLU(),
            nn.ConvTranspose2d(32, 1, 4, 2, 1),
        )

    def forward(self, x):
        B, C, H, W = x.shape
        z = self.enc(x)
        z_flat = z.view(B, -1)
        latent = self.fc_enc(z_flat)

        out = self.fc_dec(latent)
        out = out.view(B, 256, 8, 14)
        out = self.dec(out)

        out = F.interpolate(out, size=(H, W), mode="bilinear",␣
↪align_corners=False)
```

```
        return out, latent
```

[59]:
```
device = "cuda" if torch.cuda.is_available() else "cpu"
model = ConvAutoencoder(latent_dim=128).to(device)

state_dict = torch.load("checkpoints/conv_autoencoder.pth", map_location=device)
model.load_state_dict(state_dict)
model.eval()

print("  Model loaded")
```

```
 Model loaded
```

[60]:
```
SR = 22050
N_FFT = 1024
HOP = 512
N_MELS = 128
SEG_DUR = 5.0
SEG_SAMPLES = int(SR * SEG_DUR)

mel = torchaudio.transforms.MelSpectrogram(
    sample_rate=SR,
    n_fft=N_FFT,
    hop_length=HOP,
    n_mels=N_MELS
)

def preprocess(path, duration=SEG_DUR):
    wav, sr = torchaudio.load(path)
    if wav.shape[0] > 1:
        wav = wav.mean(dim=0, keepdim=True)  # mono
    if sr != SR:
        wav = torchaudio.functional.resample(wav, sr, SR)
    wav = wav.squeeze(0)

    # pad/trim
    target_len = int(SR * duration)
    if len(wav) < target_len:
        wav = torch.nn.functional.pad(wav, (0, target_len - len(wav)))
    else:
        wav = wav[:target_len]

    mel_spec = mel(wav.unsqueeze(0))
    logmel = torch.log(mel_spec + 1e-6)
    return logmel.unsqueeze(0)  # [1, 1, n_mels, time]

def beat_align(pathA, pathB):
    yA, srA = librosa.load(pathA, sr=SR)
```

```
    yB, srB = librosa.load(pathB, sr=SR)

    tempoA, _ = librosa.beat.beat_track(y=yA, sr=srA)
    tempoB, _ = librosa.beat.beat_track(y=yB, sr=srB)

    tempoA = float(tempoA)
    tempoB = float(tempoB)

    print(f"Tempo A: {tempoA:.2f} BPM, Tempo B: {tempoB:.2f} BPM")

    if tempoB > 0:
        rate = tempoA / tempoB
        # STFT of song B
        D = librosa.stft(yB)
        # Time-stretch in STFT domain
        D_stretch = librosa.phase_vocoder(D, rate=rate, hop_length=HOP)
        # Invert back to waveform
        yB_aligned = librosa.istft(D_stretch, hop_length=HOP)
    else:
        print("  Could not estimate tempo for song B, skipping time-stretch")
        yB_aligned = yB

    return yA, yB_aligned
```

[61]:
```
def interpolate_latents(fileA, fileB, steps=10):
    xA = preprocess(fileA).to(device)
    xB = preprocess(fileB).to(device)

    with torch.no_grad():
        _, zA = model(xA)
        _, zB = model(xB)

    latents = []
    for alpha in np.linspace(0, 1, steps):
        z_mix = (1 - alpha) * zA + alpha * zB
        latents.append(z_mix)
    return latents
```

[62]:
```
def decode_latents(latents):
    outputs = []
    with torch.no_grad():
        for z in latents:
            out = model.fc_dec(z)
            out = out.view(-1, 256, 8, 14)
            out = model.dec(out)
            out = F.interpolate(out, size=(N_MELS, int(SEG_SAMPLES/HOP)),
                                mode="bilinear", align_corners=False)
```

```python
            spec = out.squeeze(0).cpu().numpy()
            spec = np.exp(spec) - 1e-6  # invert log
            wav = librosa.feature.inverse.mel_to_audio(
                spec, sr=SR, hop_length=HOP, n_fft=N_FFT
            )
            outputs.append(wav)
    return outputs
```

[65]:

```python
fileA = "input/billie_jean.mp3"
fileB = "input/get_lucky.mp3"

dir_song_aligned = "output/songB_aligned.wav"

# Align BPMs
yA, yB = beat_align(fileA, fileB)

# Save aligned version of songB for consistency
sf.write(dir_song_aligned, yB, SR)

# Interpolate in latent space
latents = interpolate_latents(fileA, dir_song_aligned, steps=8)

# Decode to audio
outputs = decode_latents(latents)

# Concatenate into one transition
transition = np.concatenate([o.astype(np.float32).flatten() for o in outputs])

print("Shape:", transition.shape, "dtype:", transition.dtype)

sf.write("output/transition.wav", transition, SR)
print("  Transition saved as transition.wav")
```

```
/var/folders/48/jjk7q1v14vj5tssyzvrpzs5r0000gn/T/ipykernel_57750/9749218.py:41:
DeprecationWarning: Conversion of an array with ndim > 0 to a scalar is
deprecated, and will error in future. Ensure you extract a single element from
your array before performing this operation. (Deprecated NumPy 1.25.)
  tempoA = float(tempoA)
/var/folders/48/jjk7q1v14vj5tssyzvrpzs5r0000gn/T/ipykernel_57750/9749218.py:42:
DeprecationWarning: Conversion of an array with ndim > 0 to a scalar is
deprecated, and will error in future. Ensure you extract a single element from
your array before performing this operation. (Deprecated NumPy 1.25.)
  tempoB = float(tempoB)
```

```
Tempo A: 117.45 BPM, Tempo B: 117.45 BPM
Shape: (876544,) dtype: float32
 Transition saved as transition.wav
```