Voice Cloning with Tacotron2

Creating a voice cloning system using deep learning involves several steps, including data preparation, model training, and synthesis. Here, I'll provide a simplified Python code outline using the Tacotron2 and WaveGlow models from NVIDIA, which are commonly used for text-to-speech (TTS) applications.

Prerequisites

1. **Install Dependencies**: Make sure you have Python installed, along with PyTorch, and NVIDIA's Tacotron2 and WaveGlow repositories.

bash

```
pip install torch
pip install numpy scipy
git clone https://github.com/NVIDIA/tacotron2.git
cd tacotron2
pip install -r requirements.txt
git clone https://github.com/NVIDIA/waveglow.git
cd waveglow
pip install -r requirements.txt
```

Step 1: Data Preparation

You need a dataset consisting of audio files and their corresponding transcripts. For simplicity, we'll assume you have a dataset in the format required by Tacotron2.

Step 2: Preprocess the Data

Preprocessing involves converting audio files into mel-spectrograms and normalizing the transcripts.

python

```
# Preprocessing script (simplified version)
import os
from tacotron2.data_function import TextMelLoader, TextMelCollate
from torch.utils.data import DataLoader

# Set the paths
dataset_path = "path_to_your_dataset"
filelist = "filelist.txt"  # This file should contain lines of "<audio_path>|<transcript>"

# Create data loader
trainset = TextMelLoader(filelist, dataset_path)
collate_fn = TextMelCollate(n_frames_per_step=1)
train_loader = DataLoader(trainset, num_workers=1, shuffle=True,
                          sampler=None, batch_size=32, pin_memory=False,
                          drop_last=True, collate_fn=collate_fn)
```

Step 3: Train Tacotron2 Model

Train the Tacotron2 model on the preprocessed data.

```python
import torch
from tacotron2.model import Tacotron2
from tacotron2.train import train

# Initialize the model
model = Tacotron2()

# Load the dataset
train_loader = ...

# Train the model
train(model, train_loader)
```

Step 4: Synthesize Speech with Tacotron2 and WaveGlow

Once the Tacotron2 model is trained, use it to generate mel-spectrograms from text, and then use WaveGlow to convert mel-spectrograms into audio.

```python
import torch
from tacotron2.model import Tacotron2
from tacotron2.text import text_to_sequence
from waveglow.denoiser import Denoiser

# Load Tacotron2 and WaveGlow models
tacotron2 = Tacotron2()
waveglow = torch.load('path_to_waveglow_model')
denoiser = Denoiser(waveglow)

# Function to generate speech
def generate_speech(text, tacotron2, waveglow, denoiser):
    sequence = np.array(text_to_sequence(text, ['english_cleaners']))[None, :]
    sequence = torch.from_numpy(sequence).to(device='cuda', dtype=torch.long)

    with torch.no_grad():
        mel_outputs, mel_outputs_postnet, _, alignments = tacotron2.inference(sequence)
        audio = waveglow.infer(mel_outputs_postnet, sigma=0.666)
        audio = denoiser(audio, strength=0.01)[:, 0]

    return audio.cpu().numpy()

# Generate speech
text = "Hello, how are you?"
audio = generate_speech(text, tacotron2, waveglow, denoiser)

# Save to file
import soundfile as sf
sf.write('output.wav', audio, 22050)
```

Notes

1. **Data Quality**: The quality of the cloned voice depends heavily on the quality and quantity of your training data.

2. **Training Time**: Training a Tacotron2 model from scratch can take a significant amount of time and computational resources. Consider using pre-trained models if available.
3. **Fine-Tuning**: If you have a pre-trained model, you can fine-tune it on your specific dataset to adapt it to your voice cloning task.

This outline provides a basic framework. For a production-level system, you'd need to handle more aspects such as model checkpoints, data augmentation, error handling, and more robust preprocessing.

The Tacotron2 model primarily focuses on converting text to speech and is trained to capture general prosody and intonation patterns from the training data. However, it does not explicitly control or recognize specific pitch variations inherent in different voices. For detailed pitch control and recognition, additional steps are necessary.

Enhancing Tacotron2 for Pitch Control

To enhance the ability of the Tacotron2 model to recognize and control pitch variations, you can incorporate pitch features during training and synthesis. Here is how you can do it:

1. **Extract Pitch Contours**: During preprocessing, extract pitch (F0) contours from the audio files.
2. **Augment the Dataset**: Include pitch information in the dataset.
3. **Modify the Tacotron2 Model**: Adjust the Tacotron2 model to accept pitch information as an additional input.
4. **Train with Pitch Features**: Train the model with the augmented dataset that includes pitch features.

Step-by-Step Implementation

Step 1: Extract Pitch Contours

You can use tools like `pyworld` to extract pitch contours.

```python
import pyworld as pw
import numpy as np
import librosa

def extract_pitch(audio_path):
    y, sr = librosa.load(audio_path, sr=None)
    _f0, t = pw.dio(y, sr)
    f0 = pw.stonemask(y, _f0, t, sr)
    return f0
```

Step 2: Augment the Dataset

Modify the preprocessing script to include pitch information.

python

```python
# Preprocessing script with pitch extraction (simplified version)
import os
from tacotron2.data_function import TextMelLoader, TextMelCollate
from torch.utils.data import DataLoader
import numpy as np
import librosa
import pyworld as pw

# Set the paths
dataset_path = "path_to_your_dataset"
filelist = "filelist.txt"  # This file should contain lines of "<audio_path>|<transcript>"

class TextMelPitchLoader(TextMelLoader):
    def __getitem__(self, index):
        audiopath, text = self.audiopaths_and_text[index]
        mel = self.get_mel(audiopath)
        pitch = self.get_pitch(audiopath)
        text = self.get_text(text)
        return (text, mel, pitch)

    def get_pitch(self, audiopath):
        y, sr = librosa.load(audiopath, sr=None)
        _f0, t = pw.dio(y, sr)
        f0 = pw.stonemask(y, _f0, t, sr)
        return torch.tensor(f0, dtype=torch.float32)

# Create data loader
trainset = TextMelPitchLoader(filelist, dataset_path)
collate_fn = TextMelCollate(n_frames_per_step=1)
train_loader = DataLoader(trainset, num_workers=1, shuffle=True,
                          sampler=None, batch_size=32, pin_memory=False,
                          drop_last=True, collate_fn=collate_fn)
```

Step 3: Modify the Tacotron2 Model

Adjust the Tacotron2 model to accept pitch information.

python

```python
from tacotron2.model import Tacotron2
import torch.nn as nn
import torch

class Tacotron2WithPitch(Tacotron2):
    def __init__(self):
        super(Tacotron2WithPitch, self).__init__()
        self.pitch_embedding = nn.Linear(1, self.encoder.embedding_dim)

    def forward(self, inputs):
        text_inputs, mels, pitches = inputs
        embedded_text = self.embedding(text_inputs).transpose(1, 2)
        embedded_pitch = self.pitch_embedding(pitches.unsqueeze(-1))
        encoder_inputs = embedded_text + embedded_pitch.transpose(1, 2)
        encoder_outputs = self.encoder(encoder_inputs)
        return super(Tacotron2WithPitch, self).forward((encoder_outputs, mels))

# Initialize the modified model
model = Tacotron2WithPitch()
```

Step 4: Train with Pitch Features

Train the model with the augmented dataset that includes pitch features.

```python
from tacotron2.train import train

# Load the dataset
train_loader = ...

# Train the model
train(model, train_loader)
```

Step 5: Synthesize Speech with Pitch Control

When generating speech, include pitch information along with text input.

```python
def generate_speech_with_pitch(text, pitch, tacotron2, waveglow, denoiser):
    sequence = np.array(text_to_sequence(text, ['english_cleaners']))[None, :]
    sequence = torch.from_numpy(sequence).to(device='cuda', dtype=torch.long)
    pitch = torch.tensor(pitch, dtype=torch.float32).to(device='cuda')

    with torch.no_grad():
        mel_outputs, mel_outputs_postnet, _, alignments = tacotron2.inference((sequence,
pitch))
        audio = waveglow.infer(mel_outputs_postnet, sigma=0.666)
        audio = denoiser(audio, strength=0.01)[:, 0]

    return audio.cpu().numpy()

# Example pitch values (you can extract or define these)
pitch = [100, 120, 110, 130, 120]  # Dummy pitch values

# Generate speech
text = "Hello, how are you?"
audio = generate_speech_with_pitch(text, pitch, tacotron2, waveglow, denoiser)

# Save to file
import soundfile as sf
sf.write('output_with_pitch.wav', audio, 22050)
```

Notes

- **Pitch Extraction**: Accurate pitch extraction is crucial for capturing natural prosody.
- **Model Training**: Adding pitch features increases the complexity of the model, requiring more data and training time.
- **Fine-Tuning**: Fine-tune the model with specific voices to improve the quality and naturalness of the synthesized speech with pitch control.

This enhanced model should be able to better recognize and synthesize speech with more accurate pitch variations, improving the overall naturalness of the generated voice.

To extract voice details in real-time and clone it using a deep learning model, you need a pipeline that includes capturing live audio, extracting features (such as pitch and mel-spectrograms), and then using a trained model to generate the cloned voice. Here's how you can set up such a pipeline using Python:

Prerequisites

Ensure you have the necessary packages installed:

```bash
pip install torch numpy scipy soundfile librosa pyaudio webrtcvad
```

Step-by-Step Implementation

Step 1: Capture Live Audio

Use `pyaudio` to capture live audio from a microphone.

```python
import pyaudio
import numpy as np

def record_audio(duration=5, sample_rate=22050):
    p = pyaudio.PyAudio()
    stream = p.open(format=pyaudio.paInt16, channels=1, rate=sample_rate, input=True,
frames_per_buffer=1024)

    frames = []
    for _ in range(0, int(sample_rate / 1024 * duration)):
        data = stream.read(1024)
        frames.append(np.frombuffer(data, dtype=np.int16))

    stream.stop_stream()
    stream.close()
    p.terminate()

    audio_data = np.hstack(frames)
    return audio_data, sample_rate

audio_data, sample_rate = record_audio()
```

Step 2: Extract Features in Real-Time

Extract mel-spectrograms and pitch from the captured audio.

```python
import librosa
import pyworld as pw

def extract_features(audio_data, sample_rate):
    y = audio_data.astype(np.float32)

    # Extract mel-spectrogram
    mel_spectrogram = librosa.feature.melspectrogram(y, sr=sample_rate, n_fft=1024,
```

```
    hop_length=256, n_mels=80)
        mel_spectrogram = librosa.power_to_db(mel_spectrogram, ref=np.max)

        # Extract pitch
        _f0, t = pw.dio(y, sample_rate)
        f0 = pw.stonemask(y, _f0, t, sample_rate)

        return mel_spectrogram, f0

    mel_spectrogram, f0 = extract_features(audio_data, sample_rate)
```

Step 3: Load Pre-trained Tacotron2 and WaveGlow Models

Load pre-trained models for synthesis.

python

```
    import torch
    from tacotron2.model import Tacotron2
    from waveglow.denoiser import Denoiser

    # Load the models
    tacotron2 = Tacotron2()
    tacotron2.load_state_dict(torch.load('tacotron2_statedict.pt'))
    tacotron2.eval().cuda()

    waveglow = torch.load('waveglow_statedict.pt')['model']
    waveglow.eval().cuda()
    denoiser = Denoiser(waveglow).cuda()
```

Step 4: Clone the Voice Using the Models

Use the extracted features to synthesize the cloned voice.

python

```
    def synthesize_voice(text, mel_spectrogram, pitch, tacotron2, waveglow, denoiser):
        sequence = np.array([text_to_sequence(text, ['english_cleaners'])])[None, :]
        sequence = torch.from_numpy(sequence).to(device='cuda', dtype=torch.long)

        mel_spectrogram = torch.tensor(mel_spectrogram).unsqueeze(0).to(device='cuda',
    dtype=torch.float32)
        pitch = torch.tensor(pitch).unsqueeze(0).to(device='cuda', dtype=torch.float32)

        with torch.no_grad():
            mel_outputs, mel_outputs_postnet, _, alignments = tacotron2.inference(sequence)
            audio = waveglow.infer(mel_outputs_postnet, sigma=0.666)
            audio = denoiser(audio, strength=0.01)[:, 0]

        return audio.cpu().numpy()

    # Example text for synthesis
    text = "Hello, how are you?"
    cloned_audio = synthesize_voice(text, mel_spectrogram, f0, tacotron2, waveglow, denoiser)

    # Save to file
    import soundfile as sf
    sf.write('cloned_output.wav', cloned_audio, sample_rate)
```

Additional Considerations

1. **Latency**: Real-time processing introduces latency. Ensure the system is optimized for minimal delay.
2. **Model Fine-Tuning**: For better results, fine-tune the Tacotron2 and WaveGlow models with the target voice data.
3. **Voice Activation Detection (VAD)**: Use VAD (like `webrtcvad`) to handle silent periods effectively during real-time processing.

Optional: Voice Activation Detection

Implement VAD to handle silent periods.

```python
import webrtcvad

def apply_vad(audio_data, sample_rate):
    vad = webrtcvad.Vad()
    vad.set_mode(3)  # Aggressive mode

    frame_duration = 30  # 30 ms
    frame_size = int(sample_rate * frame_duration / 1000)
    frames = [audio_data[i:i + frame_size] for i in range(0, len(audio_data), frame_size)]

    voiced_frames = []
    for frame in frames:
        if len(frame) < frame_size:
            frame = np.pad(frame, (0, frame_size - len(frame)), 'constant')
        is_speech = vad.is_speech(frame.tobytes(), sample_rate)
        if is_speech:
            voiced_frames.append(frame)

    voiced_audio = np.hstack(voiced_frames)
    return voiced_audio

voiced_audio = apply_vad(audio_data, sample_rate)
mel_spectrogram, f0 = extract_features(voiced_audio, sample_rate)
```

This pipeline provides a framework to capture, process, and clone voice in real-time. It includes steps for capturing live audio, extracting features, loading models, and synthesizing speech. Adjustments may be needed based on specific requirements and hardware capabilities.