Homework3

Variational Auto Encoder

Part 2

Course:

Deep Learning

Supervisor:
Dr. Mansoori

Atefe Rajabi

Spring 2024

Index

Introduction

Blind Source Separation (BSS) is a pivotal technique in signal processing aimed at isolating individual components from a composite signal without prior knowledge of the components or their mixing process. This technique is extensively applied across various fields including audio signal processing and biomedical signal analysis, showcasing its capability to elucidate complex data structures through artificial intelligence methodologies.

Variational Autoencoder

Variational Autoencoders (VAEs) are a class of generative models that learn to encode data into a compressed latent space and reconstruct it back to the original domain. The model architecture consists of two main components: an encoder that maps the input data to a probabilistic latent space, and a decoder that reconstructs data from this space. By optimizing both the reconstruction loss and the Kullback-Leibler divergence, VAEs ensure a balanced approach to data generation, making them ideal for complex tasks like image separation.

Problem Definition

For Part 2 of this assignment, the challenge is to employ a Variational Autoencoder to tackle the task of Blind Source Separation with a focus on audio data. Specifically, the task is to separate mixed audio recordings, each composed of a human singing voice overlaid or blended with background music. The objective is to fine-tune a VAE that accepts these mixed audio spectrograms as input and outputs the distinct components of the vocal and music signals.

Dataset

IRMAS Dataset:

The IRMAS dataset includes musical audio excerpts with annotations of the predominant instrument(s) present. It is intended to be used for training and testing methods for the automatic recognition of predominant instruments in musical audio. For this assignment, we will focus on separating human singing voice (voi) from background music.

Implementation Detail:

ConvEncoder Class

The `ConvEncoder` class is a convolutional neural network module designed to encode input data into a latent space representation. It consists of several convolutional layers, each followed by a ReLU activation function. These layers progressively reduce the spatial dimensions of the input while increasing the number of feature maps, effectively capturing hierarchical features. The encoded features are then passed through four separate convolutional layers to produce the means and variances of two latent variables (`mean1`, `var1`, `mean2`, and `var2`). These latent variables represent compressed and abstracted versions of the input data, suitable for generating new samples in the decoder stage.

ConvDecoder Class

The `ConvDecoder` class is a convolutional neural network module designed to decode latent space representations back into the original data format. It consists of a series of upsampling layers, each followed by convolutional layers with ReLU activations. These layers progressively increase the spatial dimensions of the input while refining the features to reconstruct the original data. The final layer uses an adaptive average pooling operation to match the original input dimensions, followed by additional convolutional layers to produce the final reconstructed output. This design ensures the decoder can effectively reverse the encoding process and generate high-quality reconstructions.

ConVAE Class

The `ConVAE` class integrates the `ConvEncoder` and two `ConvDecoder` modules to form a complete Variational Autoencoder (VAE) model. The encoder compresses the input data into two sets of latent variables (means and variances). New samples are generated from these latent variables by adding random noise, scaled by the exponential of the variance. These samples are then passed through the two decoders to produce two separate reconstructions (`recons1` and `recons2`). The final reconstructed output (`recons`) is the average of these two reconstructions. The `ConVAE` class returns the final reconstruction, the individual component reconstructions, and the latent variables for further processing and loss calculation, enabling effective training and evaluation of the VAE model.

Audio Data:

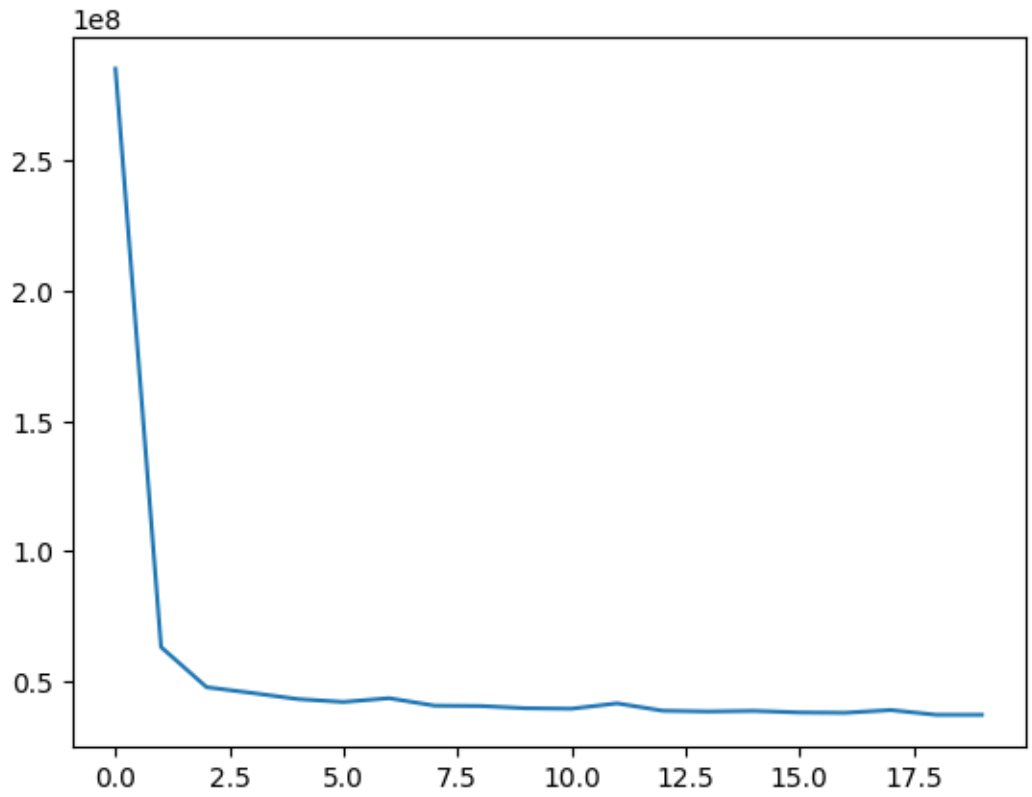
The `AudioData` class is a custom dataset designed for loading and processing audio files for use in training a machine learning model. It inherits from the PyTorch `Dataset` class and requires a list of file names as input. The class employs several audio processing transformations to convert the audio files into a suitable format for model training. The `__init__` method initializes the dataset with the list of file names and defines a transform to convert the audio data to tensors and normalize them. The `__len__` method returns the number of audio files in the dataset. The `__getitem__` method loads an audio file given an index, resamples it to a consistent sample rate of 44100 Hz, converts the waveform to a spectrogram using a short-time Fourier transform (with an FFT size of 512), and transforms the amplitude spectrogram to a decibel scale for better visualization and model input. The method returns the processed spectrogram, ready for use in training a neural network.

Train Model Function:

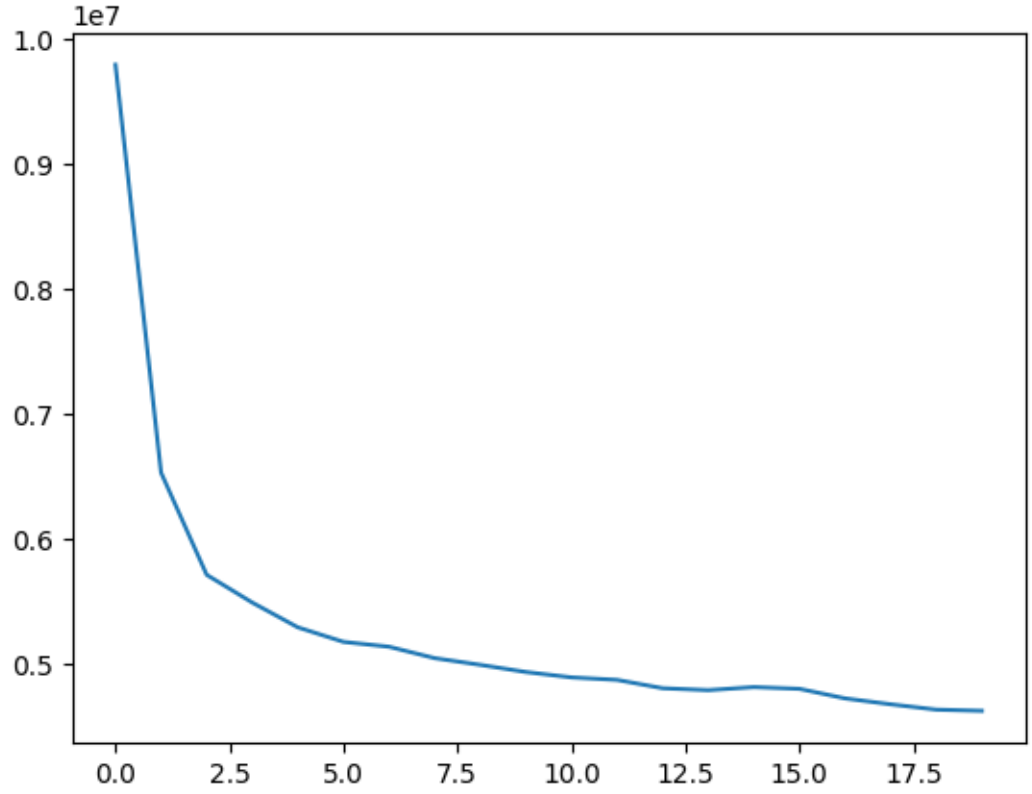
The `train_model` function is designed to train a Variational Autoencoder (VAE) model by iterating through the training and validation datasets for a specified number of epochs. It moves the model to the specified device (CPU or GPU) and initializes the best model state and minimum validation loss. During each epoch, the function calculates the training and validation losses by performing forward passes, computing the loss using the provided criterion, and updating the model parameters through backpropagation. The function tracks the average training and validation losses, printing them at the end of each epoch. If the current validation loss is lower than the previously recorded minimum validation loss, it updates the best model state. Finally, it loads the best model state and returns the lists of training and validation losses for further analysis.

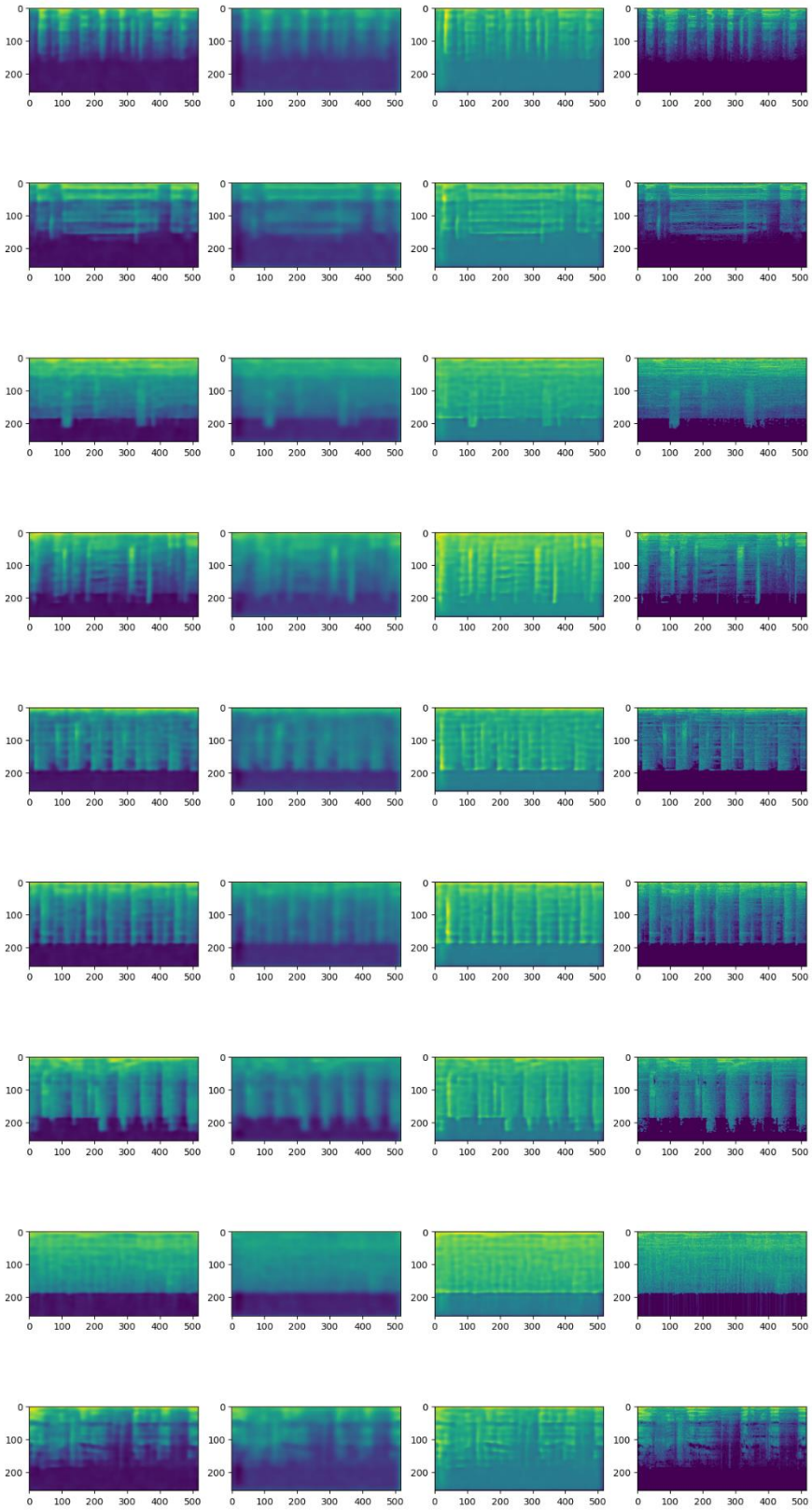
Results:

Train Loss:



Validation Loss:




0
100
200
0 100 200 300 400 500