Team :

Preethal Reddy Yellareddygari
Van Nguyen

Introduction

In this assignment, we have learned about training a generative model in audio processing using a VAE model and measuring the result using an FID score.

Our dataset is 100 samples of audio from the genre ‘classical.’ Each audio has 30s duration.

It is essential to understand specific facts about sampling rates before processing data. The sampling rate defines the frequency at which the data is sampled and the number of samples taken per second. For instance, if the sampling rate is 3000, the array size needed to represent the data would be 3000.

Implementation

This .ipynb notebook was developed and executed in a Google collaboratory environment.
Will explain or go through the setup and code implementation. In the **src** folder **A1code.ipynb.**

Library set-up and Data preprocessing

Librosa Library was used to process audio music file. This Library allows us to read audio files and process them into an array of discrete values for audio processing. Using the librosa our data have normalize into range (0,1)
The GTZAN dataset was downloaded from Kaggle and unzipped in Google Colab. The audio files were segmented into 30-second clips at a 3000 Hz sampling rate and split into training and testing datasets. These were converted into TensorFlow Dataset objects in an 8:2 ratio, shuffled, batched, and optionally prefetched for efficiency. The training dataset was shuffled randomly for generalization, while the test dataset remained in order for consistent evaluation.

VAE architecture

- **Encoder:** Encoding compresses input data into a lower-dimensional latent space. It captures the data's essential features and produces a probability distribution over the latent space, which represents the possible mean and log-var of the encoded data.
- **Reparameterization:** VAEs use a reparameterization trick to turn random sampling into a differentiable operation, allowing for gradient-based optimization.

- **Decoder:** The decoder then reconstructs the original data using the latent space provided from the reparameterization
- **Loss Function**
    - **Reconstruction Loss:** the process of measuring the accuracy of decoded data in comparison to the original input data. This help the model learn sample respect to the original dataset
    - **KL _loss:** Measures how much the learned distribution of encoded data points deviates from a target distribution. It encourages the latent space to be close to this prior distribution.

Network Architecture:

Model built from a set of encoder dense layer and decoder dense layers connected via the reparameter function.

Encoder	**Decoder**

```
model.encoder.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 1, 90000)]        0

 conv1d (Conv1D)             (None, 1, 64)             5760064

 resnet1d_block (Resnet1DBl  (None, 1, 64)             4166
 ock)

 conv1d_3 (Conv1D)           (None, 1, 128)            8320

 resnet1d_block_1 (Resnet1D  (None, 1, 128)            16518
 Block)

 conv1d_6 (Conv1D)           (None, 1, 128)            16512

 resnet1d_block_2 (Resnet1D  (None, 1, 128)            16518
 Block)

 conv1d_9 (Conv1D)           (None, 1, 256)            33024

 resnet1d_block_3 (Resnet1D  (None, 1, 256)            65798
 Block)

 flatten (Flatten)           (None, 256)               0

 dense (Dense)               (None, 20)                5140

=================================================================
Total params: 5926060 (22.61 MB)
Trainable params: 5926060 (22.61 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
model.decoder.summary()
```

```
Model: "model_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 10)]              0

 reshape (Reshape)           (None, 1, 10)             0

 resnet1d_block_4 (Resnet1D  (None, 1, 10)             5642
 Block)

 conv1d_transpose_2 (Conv1D  (None, 1, 512)            5632
 Transpose)

 resnet1d_block_5 (Resnet1D  (None, 1, 512)            131338
 Block)

 conv1d_transpose_5 (Conv1D  (None, 1, 256)            131328
 Transpose)

 resnet1d_block_6 (Resnet1D  (None, 1, 256)            32906
 Block)

 conv1d_transpose_8 (Conv1D  (None, 1, 128)            32896
 Transpose)

 resnet1d_block_7 (Resnet1D  (None, 1, 128)            8266
 Block)

 conv1d_transpose_11 (Conv1  (None, 1, 64)             8256
 DTranspose)

 conv1d_transpose_12 (Conv1  (None, 1, 90000)          5850000
 DTranspose)

=================================================================
Total params: 6206264 (23.68 MB)
Trainable params: 6206248 (23.67 MB)
Non-trainable params: 16 (64.00 Byte)
_________________________________________________________________
```

In the set of dense layers, we use a ResNet1D block for regularization, adding dropout and batch normalization.

```
from keras.saving import register_keras_serializable

@register_keras_serializable()
class Resnet1DBlock(tf.keras.Model):
    def __init__(self, kernel_size, filters, type='encode'):
        super(Resnet1DBlock, self).__init__()

        if type == 'encode':
            self.conv1a = layers.Conv1D(filters, kernel_size, 2, padding="same")
            self.conv1b = layers.Conv1D(filters, kernel_size, 1, padding="same")
            self.norm1a = tfa.layers.InstanceNormalization()
            self.norm1b = tfa.layers.InstanceNormalization()
        elif type == 'decode':
            self.conv1a = layers.Conv1DTranspose(filters, kernel_size, 1, padding="same")
            self.conv1b = layers.Conv1DTranspose(filters, kernel_size, 1, padding="same")
            self.norm1a = tf.keras.layers.BatchNormalization()
            self.norm1b = tf.keras.layers.BatchNormalization()
        else:
            return None

    def call(self, input_tensor):
        x = tf.nn.relu(input_tensor)

        x = self.conv1a(x)
        x = self.norm1a(x)
        x = layers.LeakyReLU(0.4)(layers.Dropout(0.3)(x))
        #x = layers.LeakyReLU(0.4)(x)
        x = self.conv1b(x)
        x = self.norm1b(x)
        x = layers.LeakyReLU(0.4)(layers.Dropout(0.3)(x))
        #x = layers.LeakyReLU(0.4)(x)

        x += input_tensor
        return tf.nn.relu(layers.Dropout(0.3)(x))
        #return tf.nn.relu(x)
```

FID score for music generated

The FID (Fréchet Inception Distance) measures the similarity between the distribution of generated data and real data. Lower FID scores indicate a closer similarity between the distributions, and Higher FID scores indicate how far the two distributions of data.

Implementation of FID

- **First, we use the inception model to calculate the activation of the audio. This will take the data down and sample it to important feature space.**
- **Then we calculate the probability of each activation distribution to see how far they are from each other.**
- **How do we select the Inception model?**

- **We have selected an encoder for the VAE model with a large latent space of 20. This encoder will aid us in computing the input to important feature space, as the model has learned to position those files well. To ensure fairness in our evaluation, we will use the same encoder model throughout. .**

- How we evaluate model

  First, we calculated the FID of a sample of the test set relative to a generated test set . The same was repeated for using a training sample as well, which led us to compare how close our generated audio sample was to the sample and check if the model was overfitting or not.

  We understood that our initial model was overfitting because the scores of FID of a sample of train-to-generated train audio were too high when compared to a sample of test vs generated test audio.

VAE improvement (a better model)

Dropout layer:

Dropout is a neural network method to prevent overfitting. During training, some neurons in a layer are randomly set to zero based on a predefined probability. This encourages the network to learn robust features useful with many different neuron subsets. At test time, dropout is not applied, but the weights are adjusted to compensate for dropout during training. The technique improves model generalization for better performance with unseen data. So we have a total of 3 dropout layers in our training function.

Increase in latent dimension from 2->10:

A higher-dimensional latent space can capture more variations in the data; the model might be able to encode more complex patterns and features.

Beta parameter:

Total_loss = kl_loss*beta + reconstruction_loss. Adjusting this parameter allows for more control over the learning process and the properties of the learned latent space.

By adjusting beta >1, Minimizing KL divergence between the approximate posterior and the prior distribution in the latent space can lead to a more disentangled latent representation, but it can

also result in reduced reconstruction accuracy as the model prioritizes organizing the latent space over reconstructing the input data.

By adjusting beta <1: Reducing pressure on the model to match the prior distribution can improve reconstruction fidelity but may result in a less regularized and interpretable latent space with a higher risk of overfitting.

With beta =1, is the right trade-off between KL_loss and reconstrctuon_loss.

We have used beta >1, beta = 200

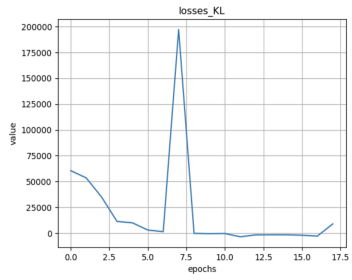
Training set size :

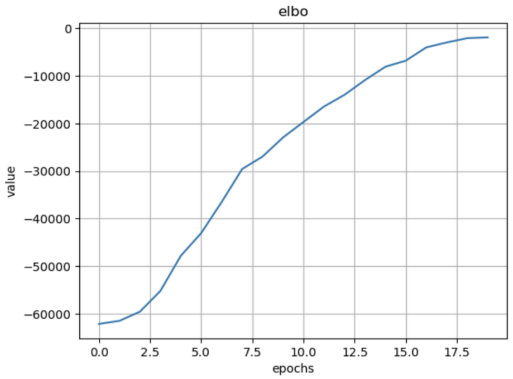
Increased training set size for better interpretability

Result + Analysis

Graph Results for base Model

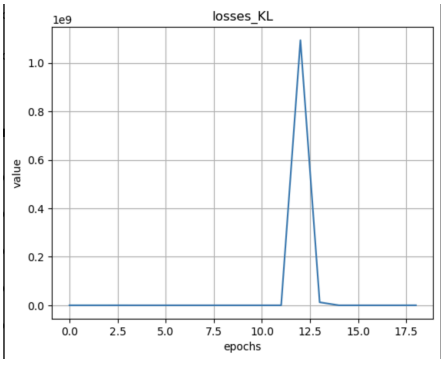
No dropout , beta=1 , latent=2

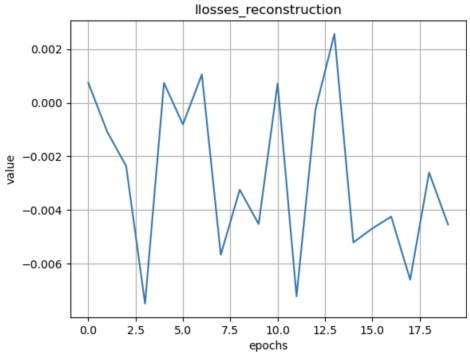


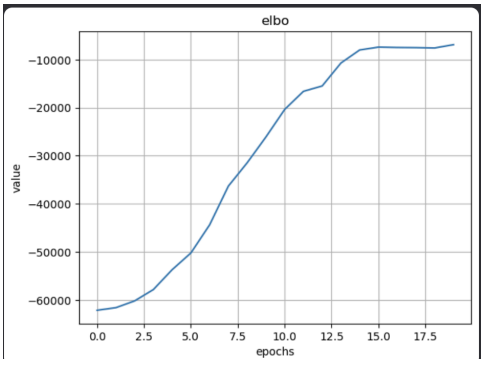


Graph Results for base Model

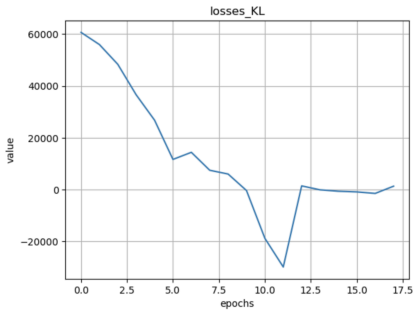
-with 3 dropout layers, beta =1, latent =2

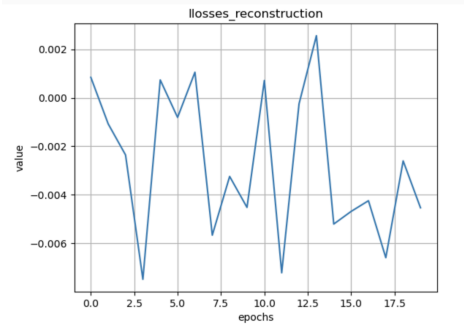


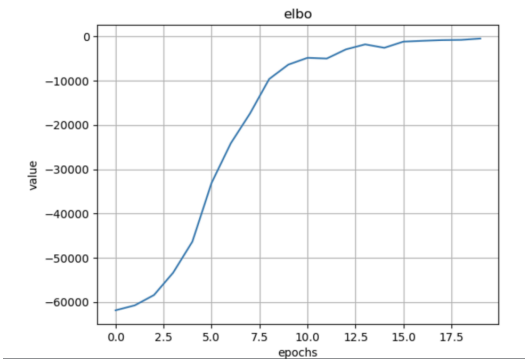




-with 3 dropout layers, beta =1, latent =20



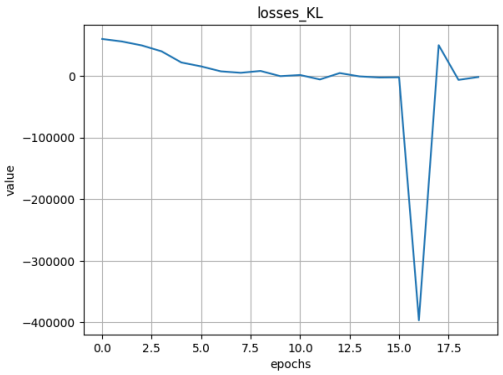


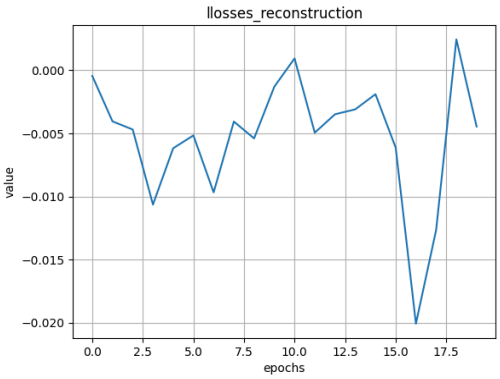


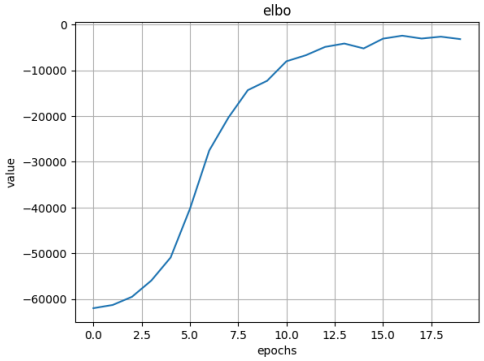
Graph Results for better VAE model

3 dropout layers, beta= 100, latent_dim = 10

-







20 epochs and sr =3k for each model

Model (sr , epoch)	FID score of Test set	FID score of Train set
Base model (no drop out , beta =1 latent = 2	101	23
Base model (drop out , beta = 1 , latent = 2)	68.55	8.6

Base model (drop out, beta = 1,latent = 20)	14.6	49.3
Model 2 (drop out , beta = 10,latent = 10)	111.5	61.1
Model 2 (drop out , beta = 100,latent = 10)	96	20.1

Conclusion

- In the beginning, we experimented with a small dataset consisting of 20 data samples in training, which was the same as the solution provided on Kaggle. We found that the models were easy to learn and produced audio waveforms with a sample rate of 3000 and minimal background noise. To prevent overfitting, we implemented dropout and increased the latent space of the model.
- As we try to work with the full audio sample provided, we are facing increasing difficulties with the model training. Even after trying methods like dropout or increasing latent space, there is still noise and frequent vanishing gradients, possibly due to the high KL loss.
- Our solution involves splitting audio data into small chunks and higher sampling rate to facilitate training of the model, with the intention that it can learn specific patterns from these chunks. However, the model often outputs similar results, which can be attributed to the latent space not being trained effectively enough to interpret the data.
- We then increase the weight of kl_loss. As a result, the model performs better in decoding the latent space.As a result in the large dataset , we are able to push the vanishing gradient problem from the based line happened around 8 epoch to our best model which is 16 epochs.

References

1 . B. (2021, December 27). *Generate music with Variational AutoEncoder*. Kaggle. https://www.kaggle.com/code/basu369victor/generate-music-with-variational-autoencoder

2 . Brownlee, J. (2019, October 10). *How to Implement the Frechet Inception Distance (FID) for Evaluating GANs*. MachineLearningMastery.com. https://machinelearningmastery.com/how-to-implement-the-frechet-inception-distance-fid-from-scratch/