Audio Clustering Deep Learning Report

1. Introduction

Deep neural networks are popular for various image processing or NLP tasks. In recent times, however, research focused on audio tasks using deep learning techniques has seen a surge. Some of the deep learning techniques have been adopted from image processing tasks, however audios are quite different as they are one-dimensional time series signal which is different from two-dimensional images. Deep learning methods with audio as input are important as audio is a very prevalent medium in our daily lives. In this project, the main objective was to train a deep neural network for the purpose of either feature extraction, clustering or both. The aim was to use this network to successfully extract feature from audio samples and cluster them into 20 clusters. Clustering could be part of the network or a separate function could be used. The rest of the report is organized as follows: Section 1 introduces the project, section 2 explains the architecture used. The details of the dataset and training as well as graphical results and analysis are showcased in section 3 and section 4 concludes the paper.

2. Architecture

I conducted a total of 50 experiments with various learning rates, batch sizes, model structures, layers, types of layers etc. The final model structure I used is shown in Figure 1. I converted the audio samples into spectrograms and saved them as images. Then I used an autoencoder to compress and reconstruct the images. The autoencoder consists of the encoder and decoder. The encoder and decoder are block based i.e. I split the original image (spectrogram) into equal sized blocks of size (144, 144, 3) and input those to the encoder. The encoder compresses the original block to size (9, 9, 1) which is a compression ratio of 0.0013. Figure 2 shows the image of size (288, 432, 3) being split into 6 equal sized blocks. Since there were 671 samples, my dataset consisted of 4026 total blocks. I then trained the autoencoder model to compress and reconstruct the spectrograms. I used the encoder to predict the original spectrograms. The output is the compressed version of the samples thereby extracting important features. I used these features to cluster the audio samples using K-Means algorithm into 20 possible clusters.

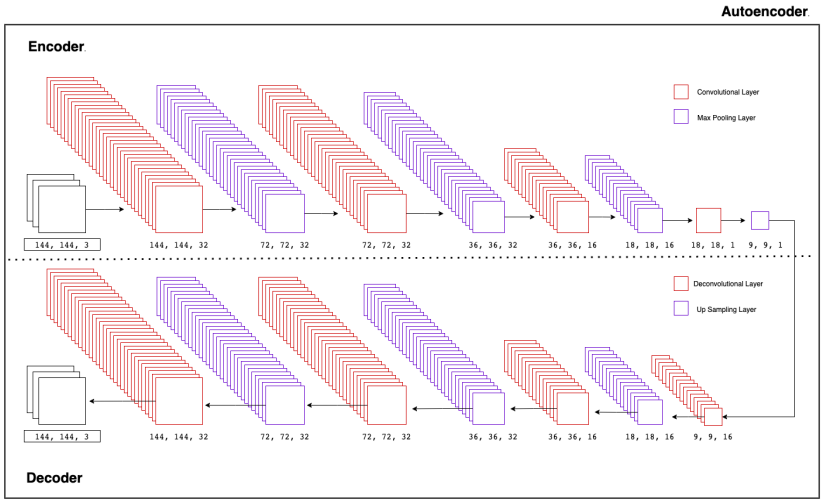


Figure 1: Model structure

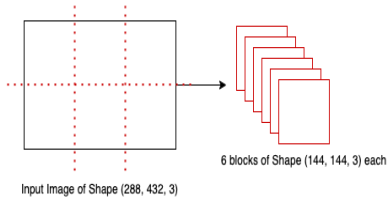


Figure 2: Block based method

3. Experimental Results & Analysis

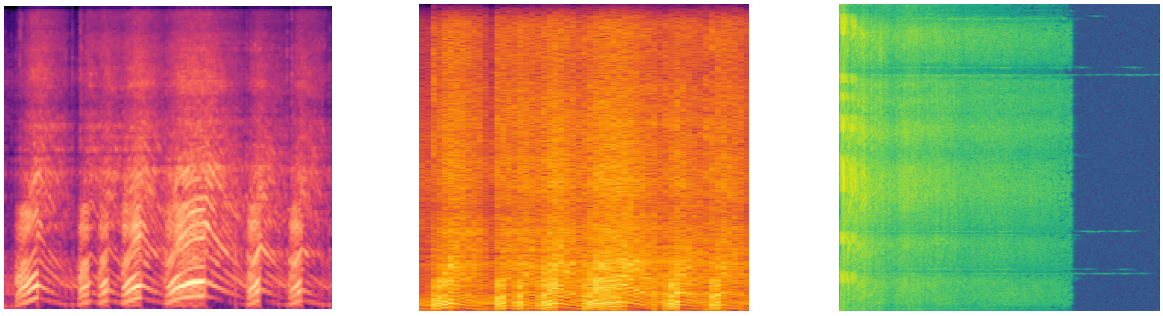
In this section, I explain the training and implementation details, dataset used, explain the hyperparameter tuning done before and during training and showcase the results of the experiments. My final result on the CLP was at rank 10 on the leader board with NMI score (on 50% of the audio samples) equivalent to 0.3943.

3.1 Dataset, Implementation Details & Spectrograms

The dataset consisted of 671 audio samples of duration 5 seconds. There were no labels given as the audios had to be clustered into 20 possible clusters. Training was done as showcased in the notebook using a GPU which took about twenty minutes on average for one experiment where the model trained for 180 to 200 epochs and stopped based on early stopping which studied the mean squared error loss function with a delta of 0.009 and patience of 15.

There are many ways to import and utilize audio data in machine learning. I chose to convert each audio sample into a spectrogram and save them as images. I then used those images to train the autoencoder model to compress and reconstruct the spectrograms. I used the encoder to predict on the original spectrograms to output the compressed version of the samples thereby extracting important features. I used these features to cluster the audio samples using K-Means algorithm. I tried a spectrogram; log spectrogram and Mel spectrogram as can be seen in Figure 3.

Mel spectrogram seemed like the best choice as its NMI score was highest. Figure 4 showcases the NMI score for clustering for each set of spectrograms used. Since (a) Mel spectrograms gave the highest score, I chose to conduct the rest of the experiments with them.



(a) Mel Spectrogram (b) Spectrogram (c) Log Spectrogram

Figure 3: Spectrograms

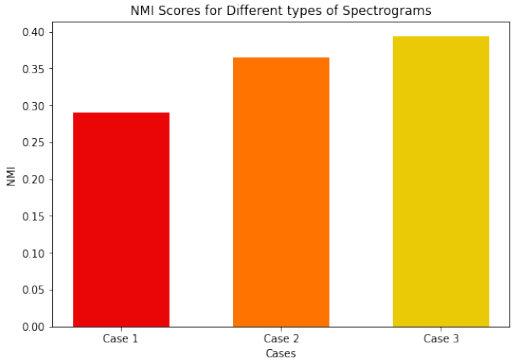


Figure 4: Trying different spectrograms

3.2 Hyperparameter Tuning & Results

I tried various model structures for both the encoder and the decoder. There were three main cases I tried for each. Figures 5 and 6 showcase the NMI score (on 50% of the set) for each layer choice for the encoder and decoder respectively. Each case is described below the respective figures. Since case 3 worked best, I chose MaxPooling and Upsampling layer for compression and reconstruction and the convolutional and deconvolutional layers for increasing and decreasing feature maps. I also tried various batch sizes and batch size 4 worked best as shown in Figure 7. Figure 8 shows the training loss for the autoencoder model used. Figure 9 showcases the progressive improvement as I conducted more experiments and tuned the parameters and altered the structure based on the performance of the previous experiments.

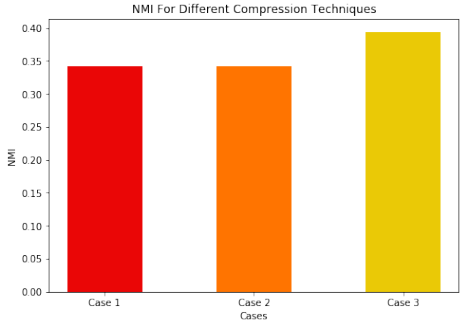


Figure 5: Trying encoder structures

Case 1: Using Convolutional strides to compress data
Case 2: Using Flat layer at the end of encoder to compress data
Case 3: Using Max Pooling layers to compress data

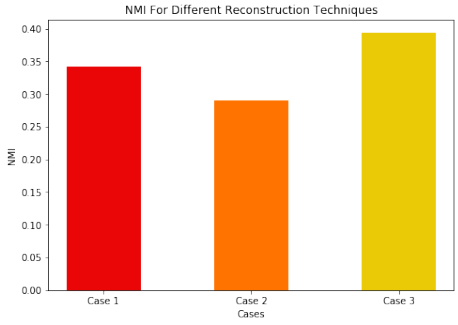


Figure 6: Trying decoder structures

Case 1: Using Deonvolutional strides to reconstruct data
Case 2: Using Reshape layer at the beginning of decoder to reconstruct data
Case 3: Using UpSampling layers to reconstruct data

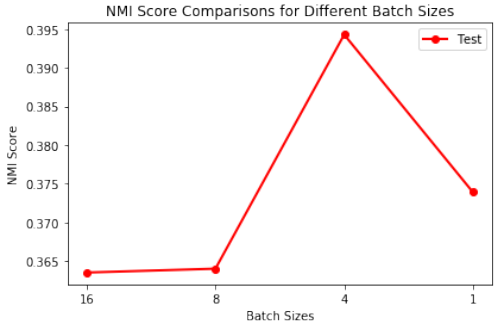


Figure 7: Trying different batch sizes

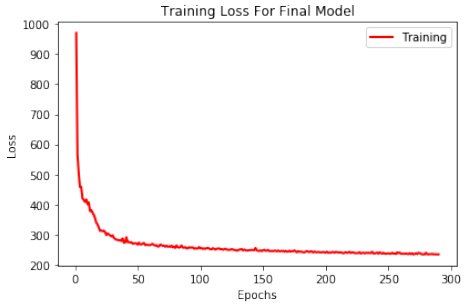


Figure 8: Training Loss Plot for Autoencoder

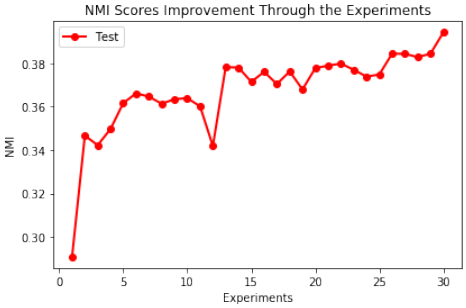


Figure 9: NMI Scores Improvement

3.3 Running the Code

Training and testing both can be done through the notebook I've submitted with my prediction file on CLP. I've included all code I used including loading the audio samples, converting them to spectrograms, saving those spectrograms as images, the model structure, the parameters used, training the model, evaluating the model. To run the code, it would be beneficial to set the root path to wherever you would like to save the model and results or load the model from.

4. Conclusion

There are many possible networks that can be used for the purpose of encoding and decoding images i.e. compressing and reconstructing the input image. For this project, I chose a 18-layer autoencoder neural network structure which extracted features that were clustered by K-Means and achieved an NMI score of 0.3943 on 50% of the audio samples. I hope to use what I've learned through this project to train a better model with higher accuracy in the near future and hopefully incorporate clustering within the network itself. Through this project, I learned about audio, its features, how to analyze audio samples and use them as input to a neural network, spectrograms and training an autoencoder from scratch.

References

[1] Anastasiu, David. COEN 342 Class Slides. Santa Clara University, 2021

[2] Matplotlib tutorial: https://matplotlib.org/

[3] Keras Autoencoders: https://blog.keras.io/building-autoencoders-in-keras.html

[4] Audio Data Analysis Using Deep Learning with Python (Part 1). https://www.kdnuggets.com/2020/02/audio-data-analysis-deep-learning-python-part-1.html