NSynth Classification using CNN

Project 1

Tejas Raval - tr7550@rit.edu
Anvesh Mateti- am2972@rit.edu

Professor: Christopher Homan
Course:CSCI.635 - Intro to Machine Learning
Rochester Institute of Technology

Table of Contents

1) Project Overview

In this report, we will focus on classifying musical notes using machine learning.The objective of the project is to classify the Nsynth Data Set single note recording into 10 instrument family names (e.g. brass, bass, flute, guitar, keyboard, mallet, organ, reed, string, synth_lead, and vocal) using a Convolutional Neural Network. Each recording of a musical note will be 4 seconds long and with the sample rate of 16kHz.

The NSynth dataset provides training, validation, and testing files. Each file has two formats: TRF files with serialized tensorflow protocol buffers and JSON files containing non-audio features alongside 16-bit PCM Wav files. For this project, we will be using the TRF format.

Data set Analysis

We start by analyzing all 3 datasets to get the sense of the distribution of class of the audio sample in each dataset. Index of the instrument family are 0-Bass, 1-Brass, 2-Flute, 3-Guitar, 4-Keyboard, 5-Mallet, 6-Organ,7-Reed,8-String,9-Synth Lead,10-Vocal.

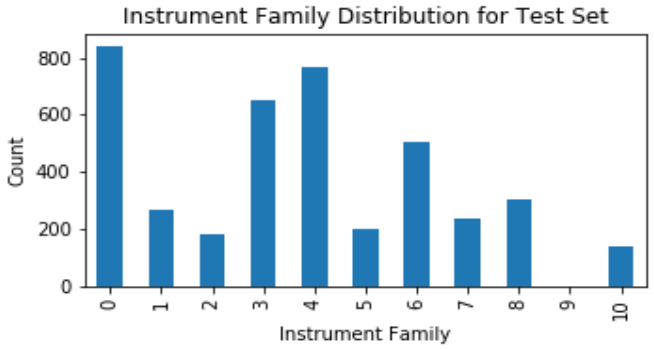

Figure 1: Distribution of audio samples of test set

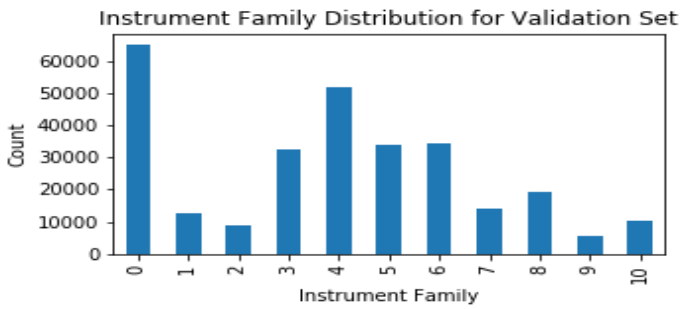

Figure 2 : Distribution of audio samples of validation set

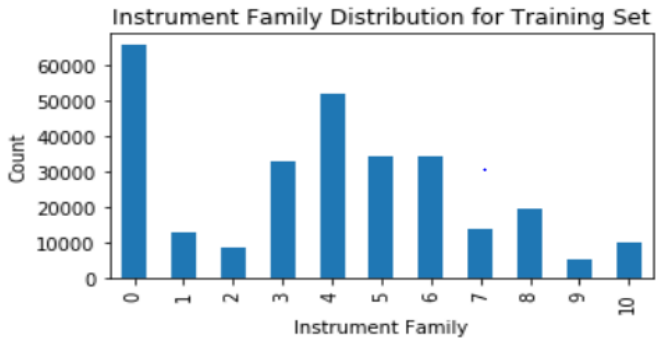

Figure 3: Distribution of audio samples of training set

We see that the samples in 3 datasets are more or less balanced. Also we noticed that the audio samples in the dataset are in ordered fashion. And hence it's important to shuffle the dataset as we have used ~20% of the training data to train one of our models.
Note: Class 9 Synth_led is abest from the train data and hence we will be ignoring it.

2) Training

Pre-Processing

We have primarily focused “audio” data in the TFR file. It's a list of audio samples represented as floating point values in the range [-1,1]. As the audio is sampled at 16KHZ, the size of each list is 64000. We have used Scipy.signal method to downsample them to 8KHZ for both the models. One of the reasons for doing this is to reduce the input dimension to the network to decrease the computing load on the processor. For the model 2, features like Pitch, Velocity, Qualities and Instrument Strings are used.

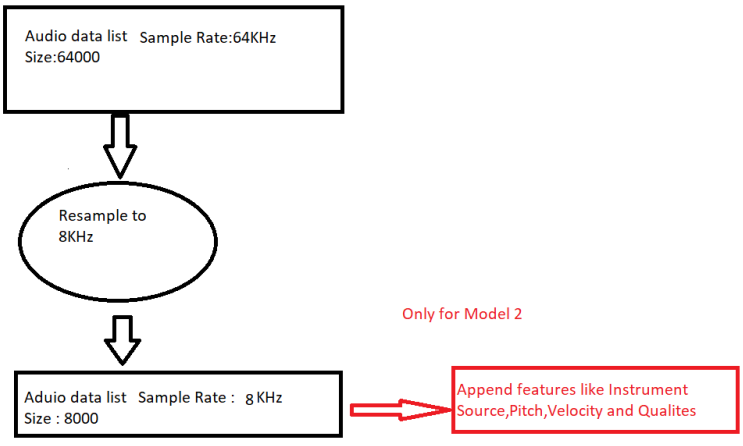

Figure 4:: Data Pre-processing workflow

3) Model

For **model 1**, we have used the sampling frequency of 8KHz. Audio list is the only feature which we have used for the training regimen. Model 1 is trained on 56284 samples considering the processing time and the availability of the GPUs.

It consists of 2 1D convolutional layers with filters size 64 and kernel size 3 and activation function of Relu. 1D function is used as the audio list is of 1 dimension and the audio also contains just one channel. Then a dropout is added of size 0.5 along with the MaxPooling1D with the pool size 2. Then a Dense layer of 100 is added with the Relu activation function. At the end, the softmax is added to classify the audio sample to a family. While trying out modifications.optimizations to this architecture, we first added the 2 Conv1D with filter=32 and kernel=3 along with no Dropout layer. Here, the accuracy achieved was below 25%. Later after some trial and error we came with the current model which gave us the accuracy of 35.47% on the test dataset and ~50% on the validation dataset. As said above, this was achieved with just 56284 audio samples(~19% of the actual training data). And hence we think if the model was trained on the entire dataset along with more number of epochs, accuracy would have been better.

```
model = Sequential()
model.add(Conv1D(filters=64, kernel_size=3, activation='relu', input_shape=(8000,1)))
model.add(Conv1D(filters=64, kernel_size=3, activation='relu'))
model.add(Dropout(0.5))
model.add(MaxPooling1D(pool_size=2))
model.add(Flatten())
model.add(Dense(100, activation='relu'))
model.add(Dense(11, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
history=model.fit(XTrain, yTrain, validation_data=(XValidate,yValidate),epochs=70, batch_size=100)
_, accuracy = model.evaluate(Xtest, ytest)
```

Figure 5: Model 1 Network Code

For **model 2**, we have used the sampling frequency of 8KHz. Along with the audio list, we have used Instrument source, pitch, velocity and qualities as added features. Model 2 is trained on the validation data provided by Nysnth. Also, we have changed the architecture of the model.

Here, we have reduced the filter size in the second conv1D to 32. And then adding a max pooling layer as the number of the features are increased. As a part to try out optimizations and modifications, we had added one more dense layer of 100 just before the dense layer of 50,but this increases the computation time and hence we decided to drop it and try a different way. This model gave 70.9% accuracy on the test dataset and ~20% accuracy on the validation dataset.

```
model = Sequential()
model.add(Conv1D(filters=64, kernel_size=3, activation='relu', input_shape=(8013,1)))
model.add(Conv1D(filters=32, kernel_size=3, activation='relu'))
model.add(MaxPooling1D(pool_size=2))
model.add(Dropout(0.5))
model.add(MaxPooling1D(pool_size=2))
model.add(Flatten())
model.add(Dense(50, activation='relu'))
model.add(Dense(11, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
history=model.fit(Xvalid, yvalid, validation_split=0.33 ,epochs=70, batch_size=100,verbose=1)
_, accuracy = model.evaluate(Xtest, ytest)
```

```
Epoch 69/70
8494/8494 [==============================] - 80s 9ms/sample - loss: 0.0193 - accuracy: 0.9942 - val_loss: 39.1711 - val_accuracy: 0.2309
Epoch 70/70
8494/8494 [==============================] - 83s 10ms/sample - loss: 0.0373 - accuracy: 0.9907 - val_loss: 35.0008 - val_accuracy: 0.1859
4096/4096 [==============================] - 10s 3ms/sample - loss: 12.3645 - accuracy: 0.7090
0.7089844
Accuracy: 70.90
```

Figure 6: Model 2 Network Code

4) Results

Model 1

Model Accuracy and Loss

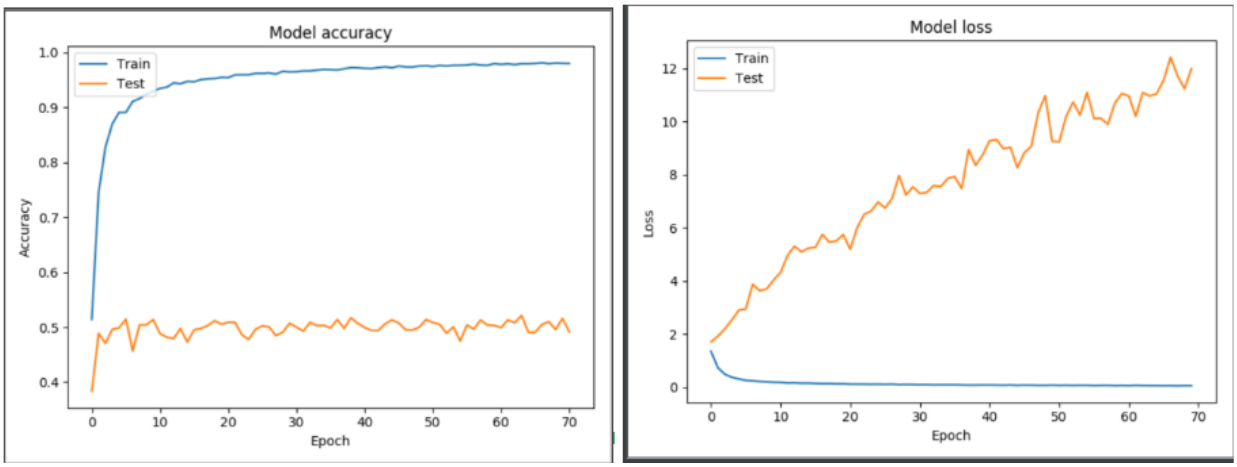

Figure 7 : Training and validation graphs for model1

Though we got good accuracy in the training phase, the accuracy was less during the testing phase. We think that the distribution of the shuffled training dataset selected for the training didn't have sufficient samples for giving better accuracy on the testing dataset.

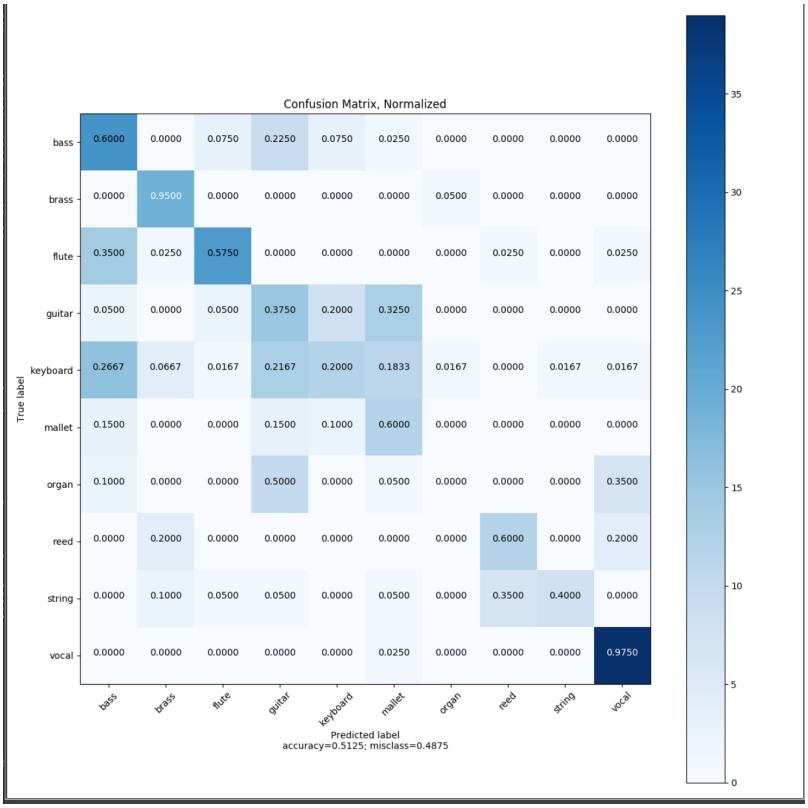

Figure 8: Model 1 Confusion Matrix

It's clear from the confusion matrix that organs like vocal,bass , brass flute and mallet, reed had a good classification compared to the others. We think that the shuffled dataset may have a higher concentration of these samples and hence their predictions were correct.

Model 2

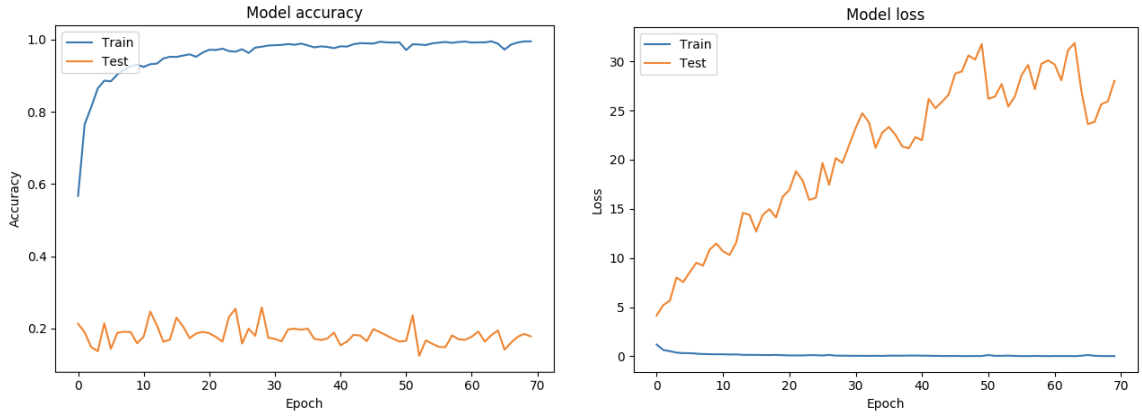

Figure 9: Left: Model 2 accuracy, Right: Model 2 loss

Figure 10: Model 2 Confusion matrix

It can be seen from the confusion matrix that most of the class have good prediction ratio compared to the 1st model's confusion matrix.

5) Discussion

#Note 1: Probabilities waveforms are in respective model folders
#Note 2: 1D audio waveforms is the folder 1DAudio-Waveforms

As per the results, our model 2 performed better over model 1. We think that happened because of 2 reasons.

1) We had added 4 extra features in the train dataset. This helped for better classification.
2) The distribution of samples in the test and validation dataset is nearly the same.

But if the model 1 was trained with the entire dataset along with the higher number of epochs, it would have given a better accuracy.

Moreover, for the model 1, we noticed that it was somewhat better to predict the samples with that higher variation in the amplitude(i.e. more peaks and deep valleys). Although, that's not correct for keyboard and the mallet class and we were unable to find out the reason behind this.
Also, for the model 2, we analyzed that it worked well with waveforms which cover less area on the Amplitude-time axis.

6) References

- **N. Ackermann, "Introduction to 1D Convolutional Neural Networks in Keras for Time Sequences," *Medium*, 14-Sep-2018. [Online]. Available: https://blog.goodaudience.com/introduction-to-1d-convolutional-neural-networks-in-keras-for-time-sequences-3a7ff801a2cf.**
- **J. Brownlee, "Display Deep Learning Model Training History in Keras," *Machine Learning Mastery*, 03-Oct-2019. [Online]. Available: https://machinelearningmastery.com/display-deep-learning-model-training-history-in-keras/.**
- **grfiv4, "Plot a Confusion Matrix," *Kaggle*, 12-Apr-2017. [Online]. Available: https://www.kaggle.com/grfiv4/plot-a-confusion-matrix.**
- **A. Hussain, A. Rosebrock, Jaffar, Hilman, M. Kabare, A. Gay, X. Wang, Alejandro, P. Agarwal, K. Klein, Manuel, E. Khan, Martin, Jakub, and Ibrahim, "Keras Conv2D and Convolutional Layers," *PyImageSearch*, 14-Feb-2020. [Online]. Available: https://www.pyimagesearch.com/2018/12/31/keras-conv2d-and-convolutional-layers/.**
- **"Keras Conv1D: Working with 1D Convolutional Neural Networks in Keras," *MissingLink.ai*. [Online]. Available: https://missinglink.ai/guides/keras/keras-conv1d-working-1d-convolutional-neural-networks-keras/.**
- **NadimKawwa, "NadimKawwa/NSynth," *GitHub*. [Online]. Available: https://github.com/NadimKawwa/NSynth.**
- **Qiaoran-Abbate, "qiaoran-abbate/NSynthClassificationRNNs," *GitHub*, 28-Feb-2019. [Online]. Available: https://github.com/qiaoran-abbate/NSynthClassificationRNNs/.**
- **A. Rosebrock, G. Pierce, D. Bonn, K. Girish, J. Viguerie, P. Rolbiecki, B. Kabakov, J. F. Winter, O. Rangel, X. Zhang, Mohanad, S. T. Bukhari, Hassan, Oliver, and Niko, "Keras vs. tf.keras: What's the difference in TensorFlow 2.0?," *PyImageSearch*, 22-Feb-2020. [Online]. Available: https://www.pyimagesearch.com/2019/10/21/keras-vs-tf-keras-whats-the-difference-in-tensorflow-2-0/**
- **"TFRecord and tf.Example : TensorFlow Core," *TensorFlow*. [Online]. Available: https://www.tensorflow.org/tutorials/load_data/tfrecord#reading_a_tfrecord_file_2.**