CSCI 635 Introduction to Machine Learning

Project 1

Krystian Derhak & Michael Lee

Professor Avenoso
Due: 04/20/2022

File Structure

- project1_writeup.pdf
- project1_code.py
- README.txt
- network1.model
- network2.model
- network_bonus.model

Overview

For this project, we endeavoured to design two convulational neural networks (CNN's) that were able to to successfully classify single note recordings with high accuracy from the NSynth dataset [1]. Further requirements included several figures and graphs which have been included in the Figures and a specified minimum accuracy between 50%-60%. We were able to achieve this with our two models which were built with mostly the same architecture with some subtle differences that allowed us to avoid over-fitting.

Each network contained 3 convolutional layers in total with pooling, dropout and fully connected layers. For a detailed breakdown of each network see figure 1 which shows the layers in detail. Our best network (referred to in the report as network 1) had an accuracy of 57.15% and our worse network (referred to as network 2) had an accuracy of 50.37%. We noticed that network 1 had a higher accuracy, suffered from over fitting and so for network 2 we added L1 and L2 regulators which did fix the over-fitting however resulted in a lower overall accuracy. Given enough time to train our second model we believe we would have a higher accuracy.

We made most of design decisions were based upon our project research paper which built BBNN but tested against competing CNN models [2]. In that paper we read their references that discussed in detail those CNNs [3]. Each of these discussed the general idea of 3 convolutional layers with pooling and fully connected layers. While we did a fair amount of guess and check to finalize our training parameters we did reference a particularly helpful GitHub as a reference for where to start [4].

Training

Before we could begin training we needed to process the data and put it in a form our model could use. This process included trimming the data from 64000 samples to 51200, which was done since endings are just 0s. Afterwards we applied an average pool with a kernel size of 25 and a stride of 25 to further reduce the size to 2048.

In assignment 3 we utilized mini-batches to train on the MNIST datset. Considering that NSynth is a much larger than MNIST we decided that we needed a larger batch size. After researching other related models had settled on on GitHub we decided our batch size should be 1024 [4]. We relied on some standard numbers to start out such as a learning rate of 0.01 but found that the training accuracy remained the same, thus we tried learning rates to 0.001 and 0.0001 and went with 0.001 due to its speed at training.

We initially tried and SGD optimizer with a momentum of 0.9 and 0.0 and found the latter to have better performance. This prompted us to switch to using an Adam optimizer since it incorporates similar mechanisms internally. As indicated while we referenced designs we had to do a fair amount of trial and error to find the best combination that would produce the best results for us.

Results

As mention in our Overview, network 1 had the higher accuracy of 57.15% which we considered to be our "best" network on the grounds of higher accuracy even though it suffered from the overfitting to the validation data. This phenomenon can be seen in Figure 4a where the green line shows training accuracy nearing 80% but testing results stay lower, which is in contrast to network 2 where testing and training accuracies are closer together (Figure 5a).

A better breakdown of the accuracies can be seen in the confusion matrices (Fig 7) where we can see which specific classifications improved between the models. Network 1 excels at bass instruments, and is relatively accurate with keyboards, guitars, organ, and string instruments. Network 2 also does will in those areas however, it struggles more so with classifying guitars and keyboards. So, we can tell that most of the accuracy that is gained is by network 1 is learning the difference between guitar and keyboards.

Discussion

After viewing the accuracies presented by the confusion matrices and overall scores, we asked ourselves if these results were logical. After consulting a breakdown of the NSynth dataset and reproductions of the sound waves we concluded that our model was indeed learning what is expected. Of the top 5 best classified instruments (Bass, Guitar, Keys, Organ, and String) those make up 70.7% of the dataset so of course when over two-thirds of the data represents only half of the instruments those represented will be classified better because the models have more opportunities to look at those musical samples. Thus, it makes sense that for the remaining instruments (brass, flute, mallet, reed and vocals) to not really be classified incorrectly since the model simply knows those are not the main instruments but can not distinguish them. In a sense our model is acting like a binary classifier and pending an initial classification into instruments it knows and does not know and further classifies from there.

The other discovery we made was after consulting some of the waveforms which further aided in our analysis to answer another question which was why guitar, keyboard (and to an extent mallet) all were often misclassified. This can be best seen looking at figure 10a and b. The keyboard(purple) and the guitar(red) one can see that they share a similar wave pattern, really only differing in overall volume. This fact, along with the knowledge that there are nearly double the amount of keyboard samples (54991 to 35423) that if the model is going to guess between the two, it statistically should guess a keyboard. The same logic applies to keyboard and mallet which can be seen in figures 11 and 12 where the keyboard(purple) and mallet(brown) share very similar waves.

Thus, in conclusion, our models perform to the accuracy requested and looking directly at the cases where they do not adequately perform it makes logical sense due to lack of data and similarity of the data that is present. Further improvements to increase accuracy would ideally be provide a larger dataset that had more samples of the under represented instruments. Additionally we would look at changing up our training regimen, perhaps modifying the training sample to boost the under represented instruments while continuing to avoid over-fitting.

Bonus

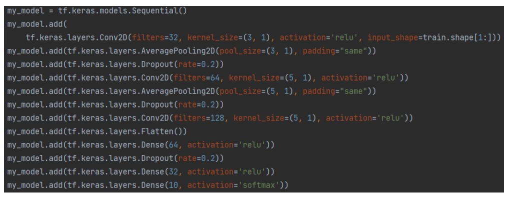
The additional challenge as set for by the project report was to the same dataset and attempt to discriminate between acoustic, electronic, and synthesized instruments. Looking at the NSynth documentation, within each family of instrument there are three different sources; Acoustic, Electronic and Synthetic. Similarly to how the instrument families are broken down, it is not an even spread. For example Bass instruments are nearly 80% synthetic but other instruments like string have 0 representation of synthetic samples.

As can been seen in figure 6a that the overall accuracy is higher at 65.43% and also does not suffer as much from over-fitting that we experienced with our other networks. Upon initial inspection one might assume that it is just our network 2, but there is a slight change where l1 and l2 are 0.0001 instead of 0.001.

We initially tried just the normal network 2, and it yielded an accuracy of 62% which was not terrible but we wanted to see if we could improve it.

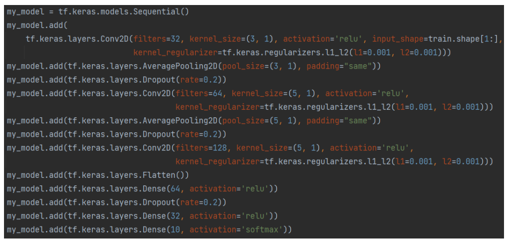
The first thing we attempted to modify was the learning rate, which we changed to 0.01 which did improve the speed but it did not result in a higher accuracy. We also tried to adjust momentum like we did with network 2 but changing the momentum also did not yield to any higher accuracy. We tried changing the dropout rate to 0.5 but this did not seem to improve anything and instead slowed down the learning process compared to our original value of 0.2. Thus, we ultimately found through trial and error that our network 2 was relatively good and changing our l1 and l2 values to 0.0001 increased our accuracy the most.

Figures



```
my_model = tf.keras.models.Sequential()
my_model.add(
    tf.keras.layers.Conv2D(filters=32, kernel_size=(3, 1), activation='relu', input_shape=train.shape[1:]))
my_model.add(tf.keras.layers.AveragePooling2D(pool_size=(3, 1), padding="same"))
my_model.add(tf.keras.layers.Dropout(rate=0.2))
my_model.add(tf.keras.layers.Conv2D(filters=64, kernel_size=(5, 1), activation='relu'))
my_model.add(tf.keras.layers.AveragePooling2D(pool_size=(5, 1), padding="same"))
my_model.add(tf.keras.layers.Dropout(rate=0.2))
my_model.add(tf.keras.layers.Conv2D(filters=128, kernel_size=(5, 1), activation='relu'))
my_model.add(tf.keras.layers.Flatten())
my_model.add(tf.keras.layers.Dense(64, activation='relu'))
my_model.add(tf.keras.layers.Dropout(rate=0.2))
my_model.add(tf.keras.layers.Dense(32, activation='relu'))
my_model.add(tf.keras.layers.Dense(10, activation='softmax'))
```

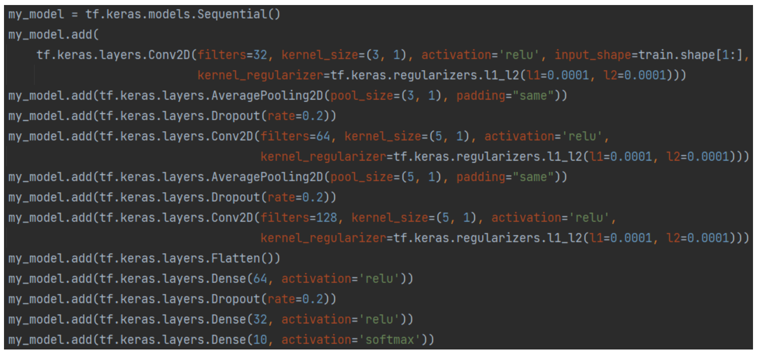
(a) Network 1



```
my_model = tf.keras.models.Sequential()
my_model.add(
    tf.keras.layers.Conv2D(filters=32, kernel_size=(3, 1), activation='relu', input_shape=train.shape[1:],
                           kernel_regularizer=tf.keras.regularizers.l1_l2(l1=0.001, l2=0.001)))
my_model.add(tf.keras.layers.AveragePooling2D(pool_size=(3, 1), padding="same"))
my_model.add(tf.keras.layers.Dropout(rate=0.2))
my_model.add(tf.keras.layers.Conv2D(filters=64, kernel_size=(5, 1), activation='relu',
                                    kernel_regularizer=tf.keras.regularizers.l1_l2(l1=0.001, l2=0.001)))
my_model.add(tf.keras.layers.AveragePooling2D(pool_size=(5, 1), padding="same"))
my_model.add(tf.keras.layers.Dropout(rate=0.2))
my_model.add(tf.keras.layers.Conv2D(filters=128, kernel_size=(5, 1), activation='relu',
                                    kernel_regularizer=tf.keras.regularizers.l1_l2(l1=0.001, l2=0.001)))
my_model.add(tf.keras.layers.Flatten())
my_model.add(tf.keras.layers.Dense(64, activation='relu'))
my_model.add(tf.keras.layers.Dropout(rate=0.2))
my_model.add(tf.keras.layers.Dense(32, activation='relu'))
my_model.add(tf.keras.layers.Dense(10, activation='softmax'))
```

(b) Network 2

Figure 1: Network Construction



```
my_model = tf.keras.models.Sequential()
my_model.add(
    tf.keras.layers.Conv2D(filters=32, kernel_size=(3, 1), activation='relu', input_shape=train.shape[1:],
                           kernel_regularizer=tf.keras.regularizers.l1_l2(l1=0.0001, l2=0.0001)))
my_model.add(tf.keras.layers.AveragePooling2D(pool_size=(3, 1), padding="same"))
my_model.add(tf.keras.layers.Dropout(rate=0.2))
my_model.add(tf.keras.layers.Conv2D(filters=64, kernel_size=(5, 1), activation='relu',
                                    kernel_regularizer=tf.keras.regularizers.l1_l2(l1=0.0001, l2=0.0001)))
my_model.add(tf.keras.layers.AveragePooling2D(pool_size=(5, 1), padding="same"))
my_model.add(tf.keras.layers.Dropout(rate=0.2))
my_model.add(tf.keras.layers.Conv2D(filters=128, kernel_size=(5, 1), activation='relu',
                                    kernel_regularizer=tf.keras.regularizers.l1_l2(l1=0.0001, l2=0.0001)))
my_model.add(tf.keras.layers.Flatten())
my_model.add(tf.keras.layers.Dense(64, activation='relu'))
my_model.add(tf.keras.layers.Dropout(rate=0.2))
my_model.add(tf.keras.layers.Dense(32, activation='relu'))
my_model.add(tf.keras.layers.Dense(10, activation='softmax'))
```

Figure 2: Bonus Network Construction

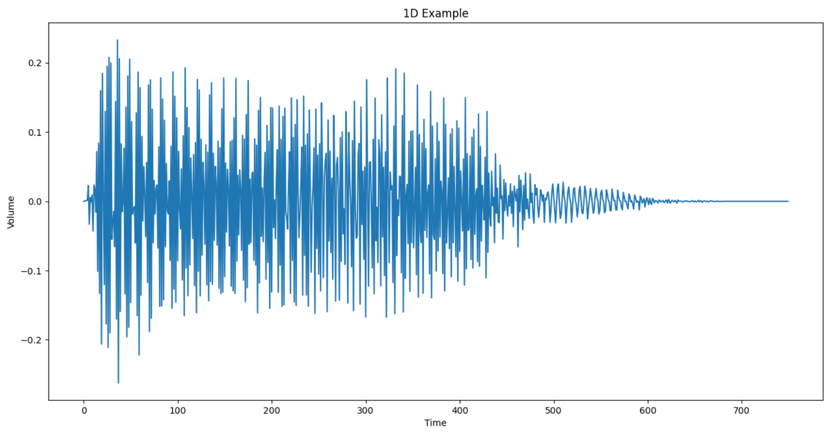
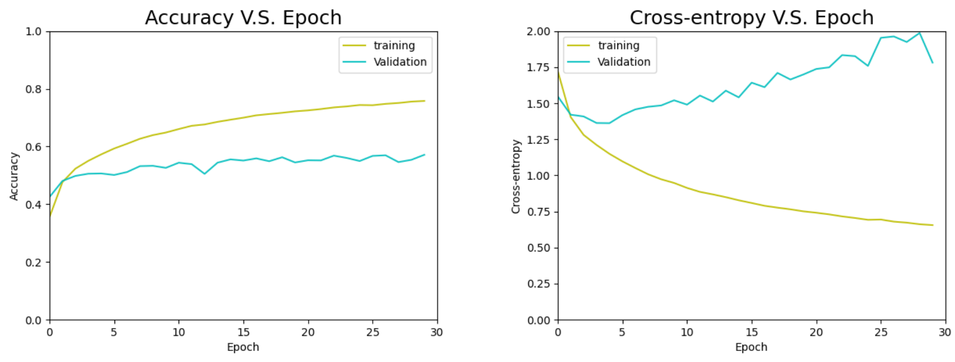


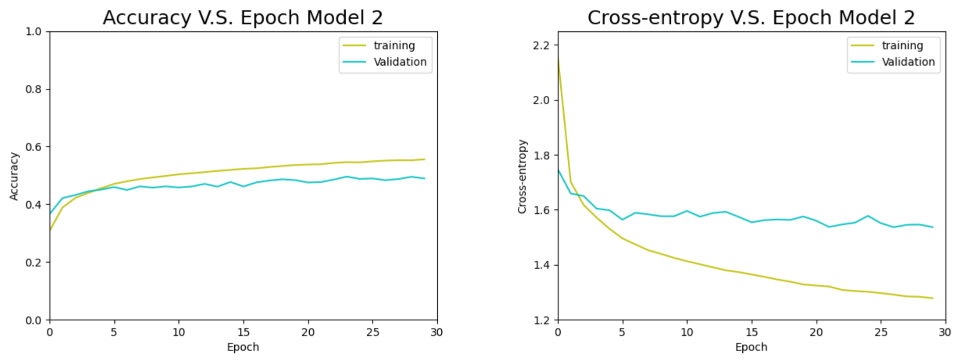
Figure 3: 1-D audio waveform



(a) Learning Curve

(b) Cross Entropy

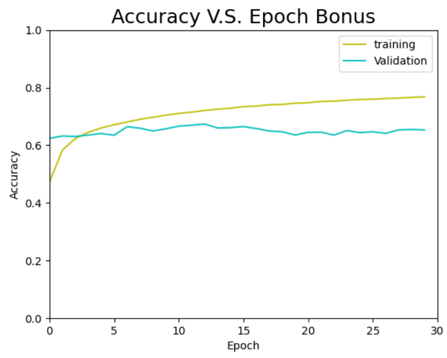
Figure 4: Learning Curves Network 1



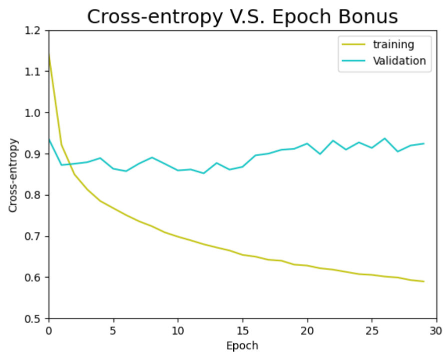
(a) Learning Curve

(b) Cross Entropy

Figure 5: Learning Curves Network 2

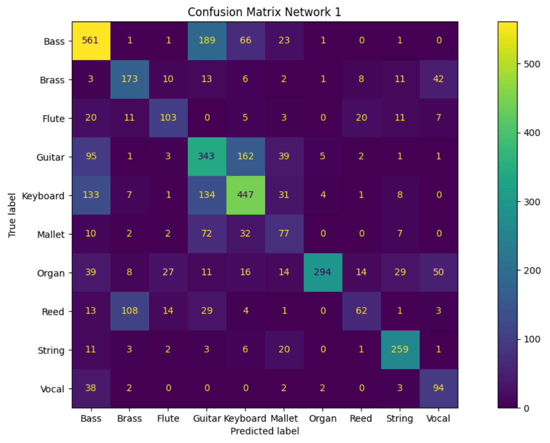


(a) Learning Curve

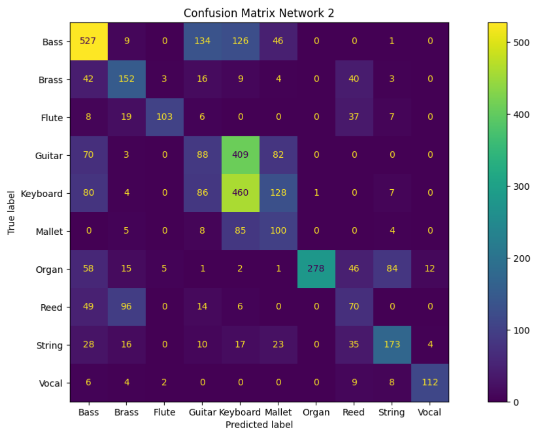


(b) Cross Entropy

Figure 6: Learning Curves Bonus Network

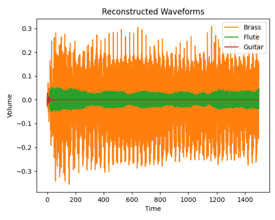


(a) Confusion Matrix Network 1

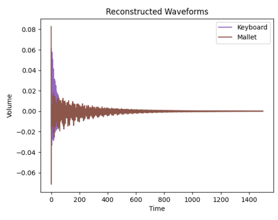


(b) Confusion Matrix Network 2

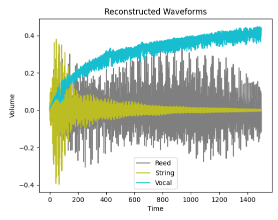
Figure 7: Confusion Matrices



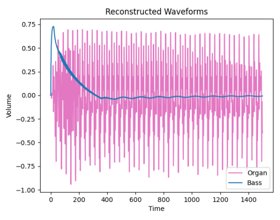
(a) Group 1



(b) Group 2

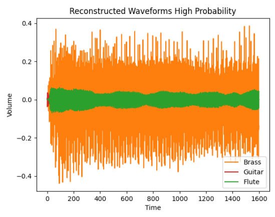


(c) Group 3

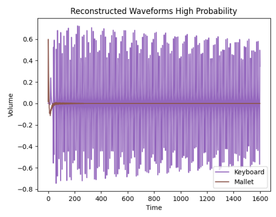


(d) Group 4

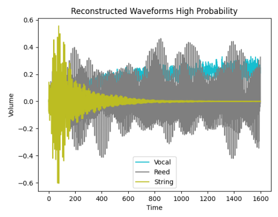
Figure 8: Waveforms of each class



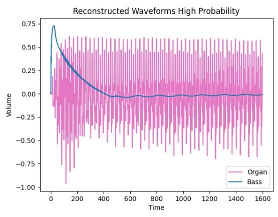
(a) Group 1



(b) Group 2

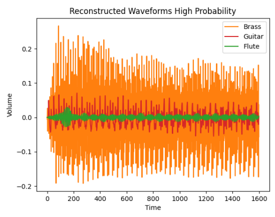


(c) Group 3

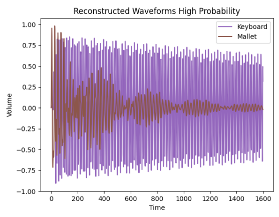


(d) Group 4

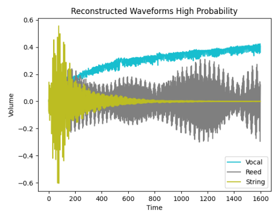
Figure 9: Waveforms with a high probability network 1



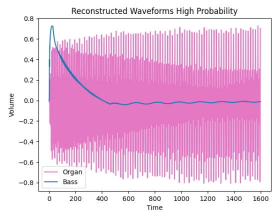
(a) Group 1



(b) Group 2

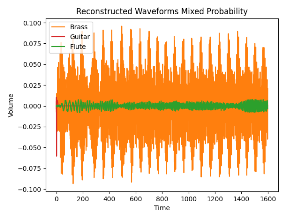


(c) Group 3

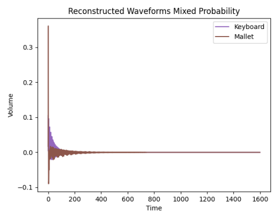


(d) Group 4

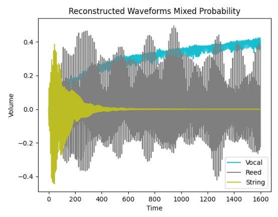
Figure 10: Waveforms with a high probability network 2



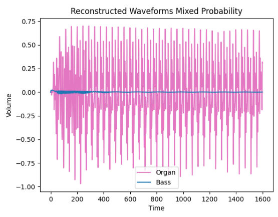
(a) Group 1



(b) Group 2

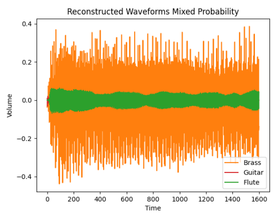


(c) Group 3

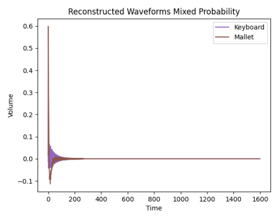


(d) Group 4

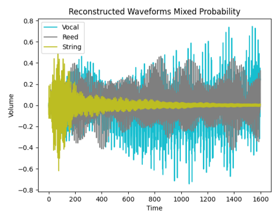
Figure 11: Waveforms with a mixed probability network 1



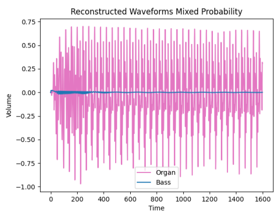
(a) Group 1



(b) Group 2



(c) Group 3



(d) Group 4

Figure 12: Waveforms with a mixed probability network 2

References

[1] J. Engel, “The nsynth dataset,” Apr 2017. [Online]. Available: https://magenta.tensorflow.org/datasets/nsynth

[2] C. Liu, L. Feng, G. Liu, H. Wang, and S. Liu, “Bottom-up broadcast neural network for music genre classification,” *Multimedia Tools and Applications*, vol. 80, no. 5, p. 7313–7331, 2020.

[3] K. Choi, G. Fazekas, M. B. Sandler, and K. Cho, “Transfer learning for music classification and regression tasks,” *CoRR*, vol. abs/1703.09179, 2017. [Online]. Available: http://arxiv.org/abs/1703.09179

[4] NadimKawwa, “Nadimkawwa/nsynth: Instrument classification on the nsynth dataset using supervised learning and cnns.” [Online]. Available: https://github.com/NadimKawwa/NSynth