Formatting instructions for STAT4984

Virginia Tech
ericzou@vt.edu

Abstract

In response to the growing consumer demand for understanding the sound-signature tuning of in-ear monitors (IEMs), I present and compare two convolutional neural network (CNN) approaches for automatic classification. I compiled a dataset of 555 distinct IEM models with 696 frequency-response measurements sourced from Super* Review, transforming each measurement into both a normalized numerical vector and a cleaned graph image after artifact removal. A 1D CNN trained directly on the numerical data and a 2D CNN trained on the corresponding images were evaluated using a stratified 60/20/20 split for training, validation, and testing. The 1D CNN achieved a test accuracy of 91 % and an F1 score of 0.874, while the 2D CNN achieved 89.2 % accuracy and an F1 score of 0.857. These results indicate that numerical-data-based classification slightly outperforms image-based classification, yet both methods are usable.

Github Link: `https://github.com/ericz0u/IEMClassification`

1 Introduction

IEMs, or In Ear Monitors are a type of earbud that sits inside the ear and seals the ear canal with a silicone or foam tip, providing better control of acoustics and noise isolation. IEMs have been used by music artists for a long time, but have become much more popular in consumer markets and audiophile communities in recent years. Apple's Airpod Pro line is an example of a more mainstream IEM, where they employ a silicone tip to seal with the ear. A common problem for consumers buying an IEM is that each IEM is tuned differently and stores that let people freely test IEMs are extremely uncommon. This means that the only way most people can get an idea of an IEM's sound signature before purchasing is looking at measurements made by reviewers and other members of the community.

Figure 1: Measurement Graph for 64Audio A6t by Reviewer Crinacle

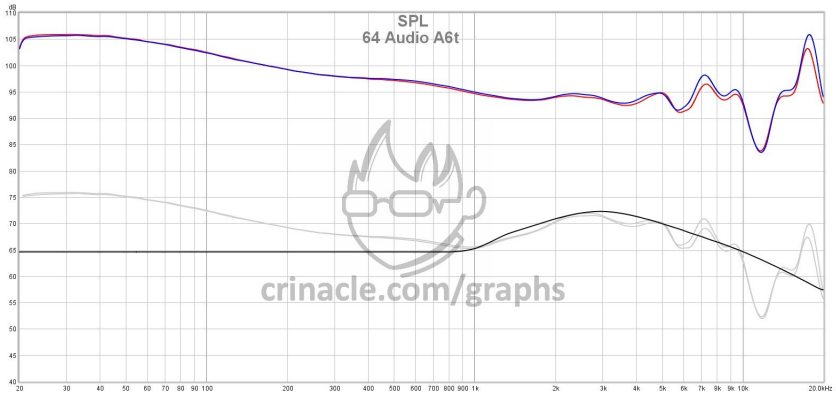


Figure 1 shows a graph from reviewer Crinacle of the 64audio A6t IEM. The x-axis is the frequency of the IEM in hz, and the y-axis is the volume in decibles at the frequency. Lower frequencies sound deeper to the ear, and higher frequencies higher. Common ranges cited in reviews are the "lows" from 20hz to 250hz, "mids" from 250hz to 2khz, and "highs" from 2khz to 20khz. The tuning shapes I will be trying to classify are V-shape, Bright, warm, and neutral. Figure 1 shows a V-shape, where it has elevated lows, recessed mids, and elevated highs. A bright tuning would be neutral bass and mids but with a higher treble area. A warm sound signature would be slightly elevated lows. A neutral sound signature would be everything in relative balance. In figure 1 the bottom line is a neutral tuning for reference. All the shapes are relative to neutral.

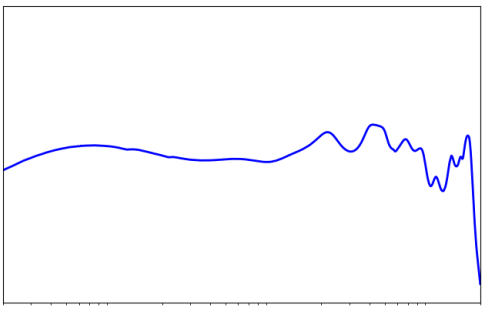
2 Related work

While I couldn't find anything directly related to this work, one thing I wanted to know was whether I should be using the image data or transforming it to numerical data. I read Performance Comparison of 1D and 2D Convolutional Neural Networks for Real-Time Classification of Time Series Sensor Data, a paper by researchers from the University of Ulsan comparing a 2d CNN to a 1d CNN. The conclusion was actually that they had the same accuraccy training time, which was suprising to me. One thing though was that the 2d CNN had a 10x slower inference time, which makes sense. I think that for my purposes the inference time won't matter too much, so I will try to fit both.

3 Dataset and Features

I have a dataset of 555 different IEMs, their sound signature(String, either V-shape, U-shaped, Bright, warm, or neutral) a CSV of 696 measurements of the freqency repsonse from 20hz to 20khz and a graph made from the CSV. I collected the data from measurements done by Super*Review, a popular IEM reviewer. There was no data for sound signatures, so I had to create this part on my own. It is quick for someone to look at a graph and be able to tell what the sound signature is, but I didn't want to spend hours doing it myself so I tried to automate this part. I first tried using chatGPT's API to look at the graphs and classify them, but it was way too inconsistent with the labeling so I did it by grouping the frequencies into seven categories: sub bass from 20-60hz, mid bass from 60 to 250z, low mids from 250t to 500hz, mids from 500hz to 2khz, upper mids from 2kh to 4khz, presence from 4khz to 6khz, and brilliance from 6khz to 12000khz, then used if statements to define the classes. This approach is definitely the weakest part of the project right now, as the classification will only be as accurate as this so it won't be entirely accurate, but the the purposes of this project its unrealistic for me to go and manually label them all. This approach is accurate enough though and should be enough in 99% of the IEMs.

Figure 2: Graphical Measurement



I did some preprocessing to remove the graph lines and logo. Figure 2 shows the images I'm using for training. The image here is created from the CSV data, put onto a graph. The reason I'm going the extra step to turn it into an image is that for most IEM measurments reviewers will only post the image rather than the numerical data, making the actual numerical data hard to find. To be more useful a image classifier would be more versatile.

When training I split the data into 60% training, 20% validation and 20% testing.

4 Methods

I will be using a 1D CNN for the numerical data and a 2D CNN for the image data.

Convolutional neural networks (CNNs) are a class of deep learning models designed specifically to process data with a known grid-like topology, such as images. Unlike fully connected networks, which scale poorly to high-dimensional inputs, CNNs exploit two key ideas: local connectivity (receptive fields) and parameter sharing (convolutional filters).

Convolutional layers. A convolutional layer applies F learnable 3D filters (kernels) of spatial size $K \times K$ across the input volume $X \in \mathbb{R}^{H \times W \times D_{\text{in}}}$ to produce an output volume $Y \in \mathbb{R}^{H_{\text{out}} \times W_{\text{out}} \times F}$. The response at output position (i, j) for filter f is

$$Y[i, j, f] = \sum_{d=1}^{D_{\text{in}}} \sum_{u=1}^{K} \sum_{v=1}^{K} K^{(f)}[u, v, d] \; \cdot \; X\big[i \cdot S + u - P,\; j \cdot S + v - P,\; d\big] + b^{(f)}\,, \qquad (1)$$

where S is the *stride*, P the *zero–padding*, and $b^{(f)}$ the bias for filter f :contentReference[oaicite:2]index=2:contentReference[oaicite:3]index=3. The spatial dimensions of the output are given by

$$H_{\text{out}} = \big\lfloor \tfrac{H-K+2P}{S} \big\rfloor + 1, \qquad W_{\text{out}} = \big\lfloor \tfrac{W-K+2P}{S} \big\rfloor + 1\,. \qquad (2)$$

Pooling layers. To reduce spatial resolution and control overfitting, CNNs interleave pooling layers. A common choice is max-pooling with receptive field K and stride S, which maps each spatial patch to its maximum:

$$Y[i, j, d] \;=\; \max_{0 \le u, v < K} \; X\big[i \cdot S + u,\; j \cdot S + v,\; d\big]\,, \qquad (3)$$

resulting in

$$H'_{\text{out}} = \big\lfloor \tfrac{H-K}{S} \big\rfloor + 1, \quad W'_{\text{out}} = \big\lfloor \tfrac{W-K}{S} \big\rfloor + 1.$$

A typical CNN alternates convolution–ReLU–pooling blocks before *flattening* the final feature maps into a vector and feeding them into one or more fully connected layers to produce the class scores:

$$\text{FC}: \quad \mathbf{z} = \text{ReLU}\big(W_{\text{fc}}\, \text{vec}(Y_{\text{last}}) + b_{\text{fc}}\big).$$

For my 1D CNN implementation I am using two sequential blocks with max-pooling, followed by a small fully connected head. First, the input—which has a single channel passed through a 1D convolution that produces 32 feature maps (kernel size 3, padding 1 to preserve length), then batch normalization, a ReLU nonlinearity, and a 20% dropout. A second identical conv–BN–ReLU layer refines those 32 features before a max-pool (kernel 2, stride 2) halves the temporal resolution. The next block repeats this pattern but expands to 64 channels (with a slightly lower 15% dropout), followed by another conv–BN–ReLU and a second max-pool to quarter the original length. The resulting tensor is flattened into a vector, projected via a linear layer to 128 dimensions, activated with ReLU and 30% dropout for regularization, and finally mapped through a second linear layer to produce the 4 output classes.

For my 2D CNN implementation I am using four sequential blocks to extract spatial features and reduce resolution, followed by a small fully connected head for classification. The network begins with a 3×3 convolution that maps the single input channel to 16 feature maps (padding=1 to preserve dimensions), passes through batch normalization and a ReLU nonlinearity, and then applies 2×2 max-pooling to halve the spatial size. This pattern repeats three more times, each time doubling the number of channels (16→32→64→128) and halving the height and width—from 224×224 down to 14×14 after four pools. A 30% dropout is applied before the classifier to regularize. The resulting tensor of shape [128×14×14] is flattened into a vector of length 128·14·14, projected through a linear layer to 128 units with ReLU and dropout, and finally fed into a second linear layer that produces the 4 output classes.

5 Results

The metrics I chose to look at were acuraccy to see how the model was performing, and F1 score to make sure the classes are evently looked at. I looked at the confusion matricies too to see how the model was seeing different classes. I was having trouble getting the model to converge at first, but after a couple experimenting runs increasing the depth and making the learning rate dynamic I was able to get it to converge. It seemed to be overfitting a bit though, so I added some dropout. The final model got 91% accuraccy and 87.4% F1 score on testing.

Figure 3: Confusion Matrix for 1D CNN

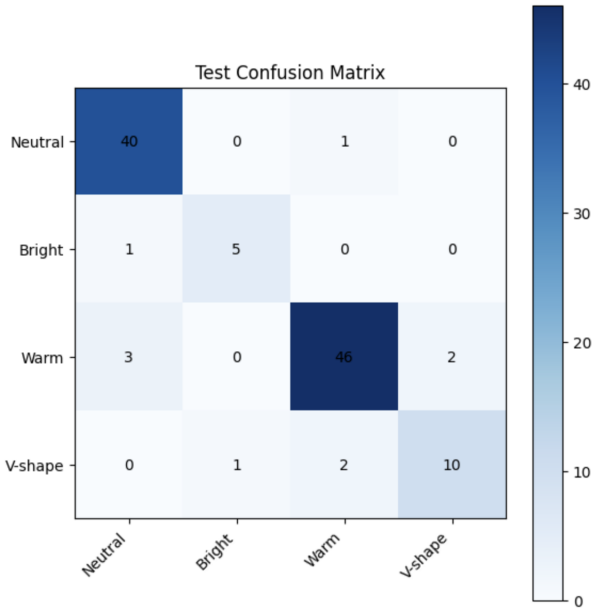


Figure 4: Training vs Validation Loss for 1D CNN

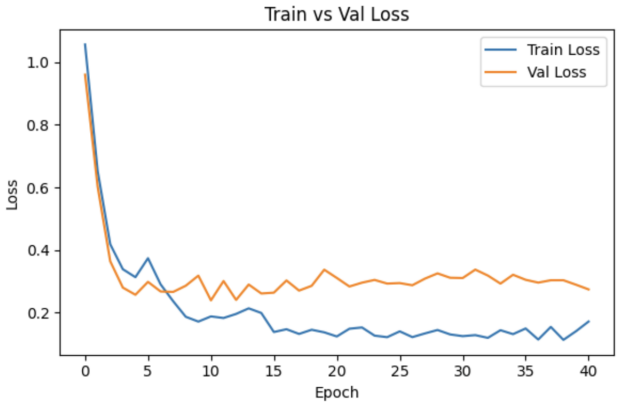


Figure 3 shows the 1d CNN's confusion matrix, it is relatively accurate with some mixups between warm and neutral. The train and val loss looks to be decent. At first it would get to a really high train accuraccy and the val loss would improve slowly, implying that the model was overfitting. I changed up the dropout values and the issue was mostly solved.

The 2d CNN presented a lot more trouble and I spent significantly more time to get it to work. Once it did work though, it got similar results.

The model got a testing accuraccy of 89.19% and a F1 score of 85.71%. Figure 5 shows the confusion matrix for the 2d CNN. Interesting was that warm-neutral was still the most confused area, but

Figure 5: Confusion Matrix for 2D CNN

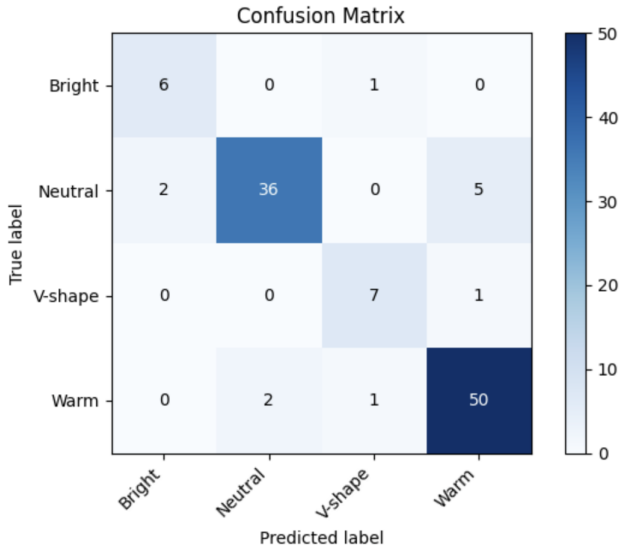
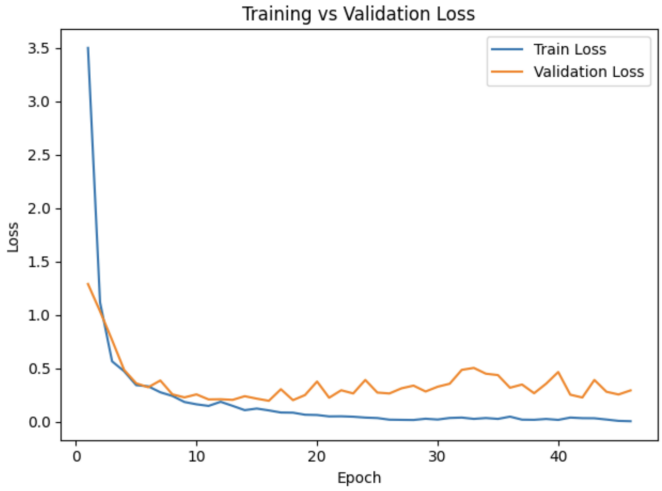


Figure 6: Training vs Validation Loss for 2D CNN



looking at the way they were generated they are very similar so this is likely a dataset issue. The training and validation losses were also decent.

6 Conclusion and discussion

I fitted two models - a 1D and 2D CNN on my data of 555 IEM frequency responses to classify between V-shape, Warm, Bright, and neutral. As was suggested in the paper for the related work section, the results were very similar between the two models. The 1D CNN was slightly better, but it could have been entirely due to the model training and how I defined the models. Both achieved around 90% testing accuracy, and warm and neutral were the two most confused classes in the confusion matricies. If I had more time and team members then besides the data, I would also try some other methods, simpler ones like KNN or and SVM or a more complex one like LSTM-CNN.

References

S. M. Shahid, S. Ko and S. Kwon, "Performance Comparison of 1D and 2D Convolutional Neural Networks for Real-Time Classification of Time Series Sensor Data," 2022 International Conference on Information Networking (ICOIN), Jeju-si, Korea, Republic of, 2022, pp. 507-511, doi: 10.1109/ICOIN53446.2022.9687284. keywords: Training;Solid modeling;Two dimensional displays;Time series analysis;Real-time systems;Data models;Time measurement;convolutional neural network;classification;time-series data,

Super* Review. (n.d.). Squiglink – IEM frequency response database. Retrieved May 10, 2025, from https://squig.link