Music source classification

I. Overview:

1) Dataset

The main goal of this project is to train a machine learning model that can deduce information about the source of a music signal. Specifically, the two questions I'm trying to answer are:

- Which type of instruments was used to produce the audio sample?
- Which method of sound production was used to produce the audio sample?

The data used in this project were taken from the NSynth dataset. Due to the limited time and computing resources, I only used the test set, which consists of 4096 labeled audio samples of musical notes. Each sample is sorted into many different categories, but I only care about the two that correspond to my questions:

- ***instrument_family***: bass, brass, flute, guitar, keyboard, mallet, organ, reed, string, synth, and vocal
- ***instrument_src***: acoustic, electronic, and synthetic

Each label is represented as an integer, which is just its index in the lists above. With a sample rate of 16000 Hz and a duration of 4 seconds, each audio sample is an array of length 64000.

2) Code:

The source code of this project is hosted here: https://github.com/bucket420/AudioClassification. Required dependencies are *tensorflow*, *numpy*, *pandas*, and *matplotlib*. All models were trained with *tensorflow*'s *keras* library. There are two main files:

- ***train.ipynb***: used to train the models
- ***results.ipynb***: used to plot the results

preprocessing.ipynb was used to process the data before training, and *utils.py* includes helper functions. The dataset is uploaded separately here. Trained models are stored in the *models* folder and can be loaded to make predictions.

I learned a lot from *tensorflow*'s official tutorials and took two functions, *get_spectrogram*, and *plot_spectrogram*, from there. These were only used to preprocess and visualize the data. The vast majority of the work is done by myself.

II. Training:

The original dataset was split into a training set and a testing set. The former contains 75% of the data (3072 samples), and the latter contains the remaining 25% (1024 samples).

I initially included a normalization layer in all of my neural networks, but I found that it made little difference in the results, so I removed it.

All models were compiled with the *Adam* optimizer, which decided the optimal learning rate.

1) Instrument classification (first question):

For this question, I turned all audio samples in the dataset into spectrograms and use them as my input feature. Each spectrogram, which can be treated as an image, is an 2d array of size (499, 129). While the original audio signal only contains information about time, a spectrogram also has information about frequencies, which is very useful for audio classification. Furthermore, a spectrogram can be resized easily, which potentially saves computational cost and allows input of arbitrary sizes. As this is a classification problem, I used a convolutional neural network and logistic regression to train my models.

a) CNN model:

The layers are shown in the figure on the right. The *resizing* layer could be used to downsample the input, but it turned out my computer could handle the original size, so I just resized the input to its original size, and *resizing* didn't actually do anything to the training data. By doing this, however, I can use the model later with input of arbitrary sizes.

Only the three *convo2d* layers and the output (*dense*) layer had trainable weights, as all other layers are processing layers. A *maxpooling* layers were placed after each *convo2d* layer, significantly downsampling the problem. Two *dropout* layers were also used to prevent overfitting.

Layer (type)	Output Shape	Param #
resizing (Resizing)	(None, 499, 129, 1)	0
conv2d (Conv2D)	(None, 497, 127, 16)	160
max_pooling2d (MaxPooling2D)	(None, 248, 63, 16)	0
conv2d_1 (Conv2D)	(None, 246, 61, 32)	4640
max_pooling2d_1 (MaxPooling2D)	(None, 123, 30, 32)	0
conv2d_2 (Conv2D)	(None, 121, 28, 64)	18496
max_pooling2d_2 (MaxPooling2D)	(None, 60, 14, 64)	0
dropout (Dropout)	(None, 60, 14, 64)	0
flatten (Flatten)	(None, 53760)	0
dropout_1 (Dropout)	(None, 53760)	0
dense (Dense)	(None, 11)	591371

b) Logistic regression model:

Each spectrogram is quite large in size, so to utilize my GPU and save time, instead of using a library for logistic regression, I used a *keras's* GPU-supported neural network with only processing layers and the output layer. There is no difference between this neural network and logistic regression.

2) Method of sound production classification (second question):

I initially did the exact same thing as in the first question. However, the results were too similar, so I decided to try something new. This time, instead of turning audio signals into spectrograms, I used them directly as input feature. The models were also trained with a convolutional neural network and logistic regression.

a) CNN model:

The layers are shown on the right. There's no *resizing* layer for 1d input, so this CNN only accepted input of size 64000.

Only the two *convo1d* layers and the output (*dense*) layer had trainable weights. Maxpooling was also used, but it didn't downsample as much as the 2d version. As a result, this model took much more time to train.

Layer (type)	Output Shape	Param #
conv1d_6 (Conv1D)	(None, 63998, 32)	128
max_pooling1d_6 (MaxPooling 1D)	(None, 31999, 32)	0
conv1d_7 (Conv1D)	(None, 31997, 64)	6208
max_pooling1d_7 (MaxPooling 1D)	(None, 15998, 64)	0
dropout_6 (Dropout)	(None, 15998, 64)	0
flatten_3 (Flatten)	(None, 1023872)	0
dropout_7 (Dropout)	(None, 1023872)	0
dense_2 (Dense)	(None, 3)	3071619

b) Logistic regression model:

This time, I also used a neural network with only processing layers and output layer in place of logistic regression. The only difference is that the input is now an audio signal rather than a spectrogram.

III. Result:

1) Instrument classification (first question):

a) CNN model:

This is by far the best performing model, with an accuracy of 95% on the testing set. It took only 17 epochs to train.

b) Logistic regression model:

Expectedly, logistic regression didn't do as well as CNN. After 50 epochs, the model had an accuracy of 95% on the training set and 78% on the testing set, which is an indication of overfitting. However, 78% is not a bad result, considering that I was trying to predict 11 classes.

2) Method of sound production classification (second question):

a) CNN model:

This model performed much worse than the one trained with spectrograms even though it was only predicting 3 classes. Its final accuracy was 98% on the training set, but only 78% on the testing set. I was expecting this, since frequencies are important factors that determine the properties of a sound, but they are not directly presented in audio signals. Nevertheless, 78% is still far better than a random guess.

b) Logistic regression model:

This is the worst performing model, with 94% training accuracy and 58% testing accuracy. Considering that a random guess would have an accuracy of 33%, 58% is acceptable, but I wouldn't count on it.