Report

Authors: A. Jamal, U. Karda, O. Sarde
Instructor: C. Homan
Rochester Institute of Technology

1. Overview:

Two Deep Convolutional Neural Network (CNN) were created to train on and classify 10 different classes of sounds from the Neural Audio Synthesis (NSynth) dataset. These CNNs were of a non-trivial nature, consisting of four or more layers. Preliminary analysis of training data, exploratory analysis of network architecture and accuracy on testset lead to interesting findings on nature of given dataset and its features. Particularly, increase in non-linearity of model through induction of more layers and their dimensions was found to increase model accuracy which inturn also resulted in expansion of model training time. Number of features underconsideration was also found to affect training time in a direct proportion. The predominance of one feature ("audio") to be able to train the model was also noted. Finding optimum balance between number of model layers, acceptable accuracy and acceptable training time through experimentation with different architecture was one of the prime findings. Expected increase in accuracy through increase in number of epochs till point of convergence was also found.

The non-trivial model developed was a preliminary deep CNN with 2 convolutional layers, 1 average pooling layer and finally the dense output layer for classification. This model had Rectified Linear Unit (ReLU) as an activation function for it's convolutional layers to avoid vanishing gradient problem [1] and a softmax activation function for the output layer for categorical classification [2]. It also consisted of a global average pooling layer as a merging layer between the convolutional layers and output layers through reduction of dimensionality.

Layer (type)	Output Shape	Param #
conv1d_128 (Conv1D)	(None, 7999, 64)	192
conv1d_129 (Conv1D)	(None, 7998, 64)	8256
global_average_pooling1d_17	(None, 64)	0
dense_94 (Dense)	(None, 10)	650

Total params: 9,098
Trainable params: 9,098
Non-trainable params: 0

Fig 1. Architecture of First Model

This model had an accuracy of 0.410400390625 through 10 epochs when run on google colab.

The optimized model was built using the architecture of the first model. It consisted of 3 convolutional layers, 1 max pooling layer, 1 dropout layer followed by flattening to adjust for dimensionality. The flattened output was sent to a dense layer before the final classifying output layer. Increase in

non-linearity, reduction of dimensionality and increase in accuracy were achieved by employing Leaky ReLU [3][4] as the activation function for the convolutional layer, utilization of max pooling layer [5] and dropout layers [6] and changing dimension of layers respectively.

```
Model: "sequential_48"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d_133 (Conv1D)          (None, 7999, 64)          192
_________________________________________________________________
conv1d_134 (Conv1D)          (None, 7998, 32)          4128
_________________________________________________________________
conv1d_135 (Conv1D)          (None, 7997, 92)          5980
_________________________________________________________________
max_pooling1d_109 (MaxPoolin (None, 3998, 92)          0
_________________________________________________________________
dropout_106 (Dropout)        (None, 3998, 92)          0
_________________________________________________________________
flatten_24 (Flatten)         (None, 367816)            0
_________________________________________________________________
dense_97 (Dense)             (None, 24)                8827608
_________________________________________________________________
dense_98 (Dense)             (None, 10)                250
=================================================================
Total params: 8,838,158
Trainable params: 8,838,158
Non-trainable params: 0
```

Fig 2. Optimized Model

The accuracy of the optimized model is 0.901611328125 through 10 epochs when run on google colab.

2. Training:

2.1. Feature Analysis :

For preliminary data preprocessing the Synth Lead class is removed from the training data. As the audio feature was found predominant in predicting the classification class the model was based on it. To reduce dimensionality of the feature, we utilized signal resampling from the scipy library to reduce the sample size from 64,000 to 8000 to reduce model training time. This resampling is based on fourier series and produces acceptable reproduction [7].

2.2. Criterion Functions :

Generic Deep CNN architectures were developed initially. Then through trial and error both models were developed based on methodology mentioned in [8].

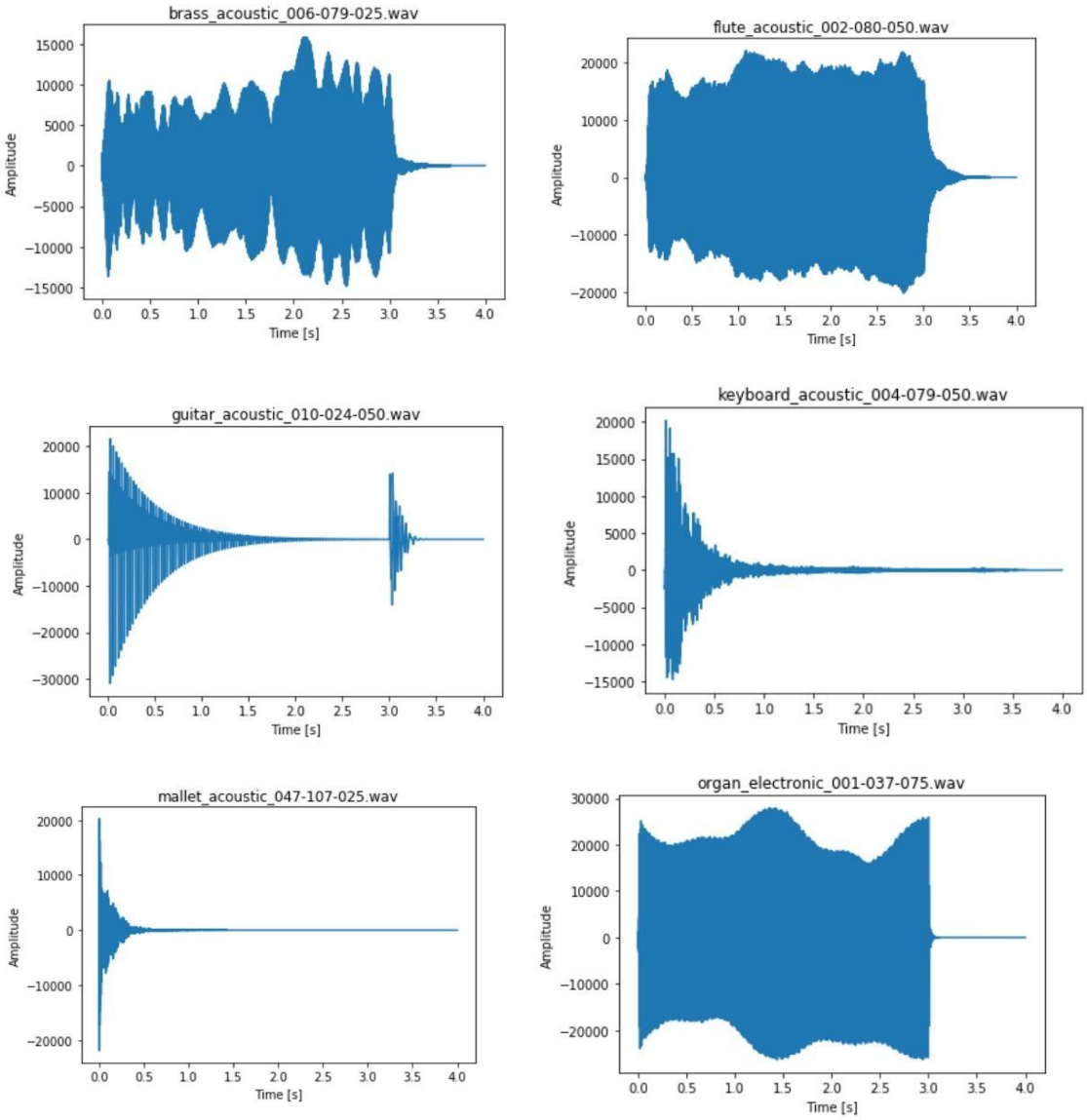
To generate the optimized model from the simple model, the activation function was changed from the ReLu function to the Leaky Relu function to account for the “dying ReLu problem”. Addition of another convolutional layer with increased output nodes was found to increase model accuracy due to induced nonlinearity from the change in number of nodes between two layers. To account for increase in training time due to addition of such a convolutional layer, a max pooling layer was added to reduce dimensionality and thus mitigate the increase in training time [5]. A dropout layer was added to account for model overfitting before flattening the output of the convolutional architecture. Through experimentation a core layer of increasing nodes before the final output layer was found to further enhance accuracy without affecting the training time on a large scale.

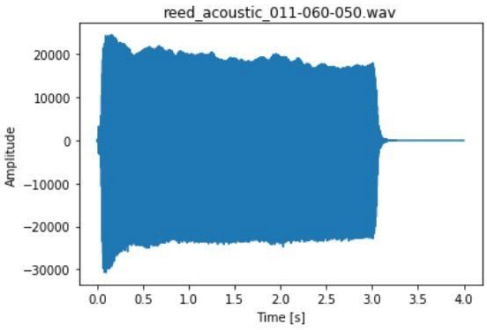
2.3. Training regimen:

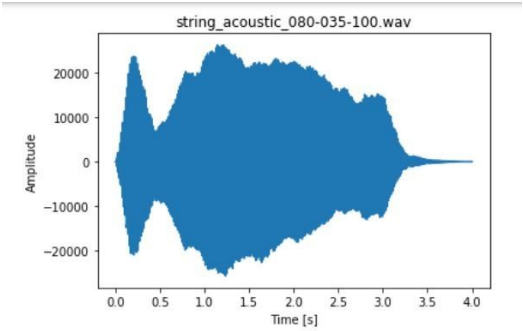
For training the models, we started by training it on our university server using a GPU but we found it to be quite slow and we came across some invalid data in the training data which stopped the training. Then we shifted to Google Colab, where we uploaded the validation data and test data but we lacked disk space to download or upload the 70 GB training data. Hence we ended up training our models on the validation data and testing on the test data. For validation, we split our training data into the ratio of 0.67 : 0.33 and got our results. Intentional reduction of batch size from the server side of google colab resulting in false accuracy, which inturn might affect unsuspecting users was found.

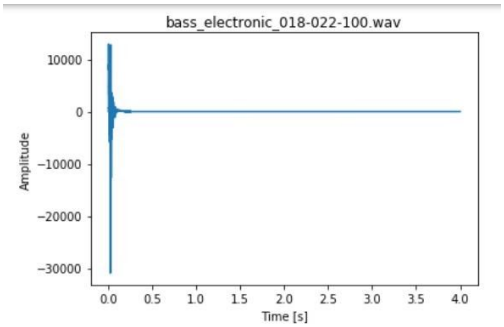
3. Data Visualization:

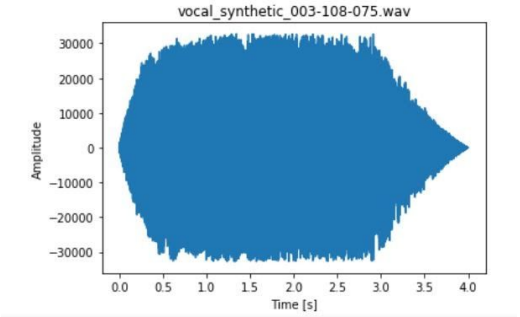
Visualization of a 1-D audio waveform:





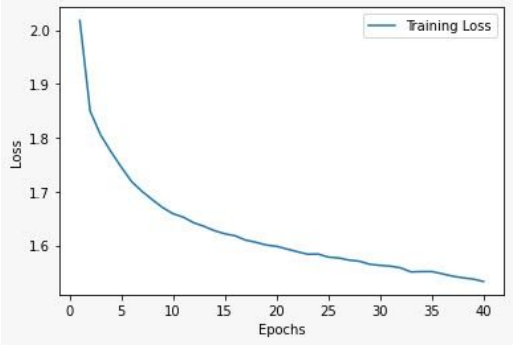


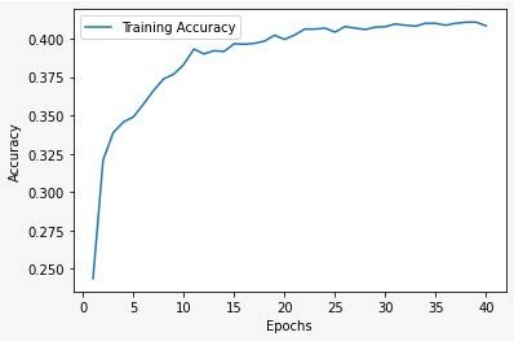




4. Results :

Visualizations of results for our preliminary model:





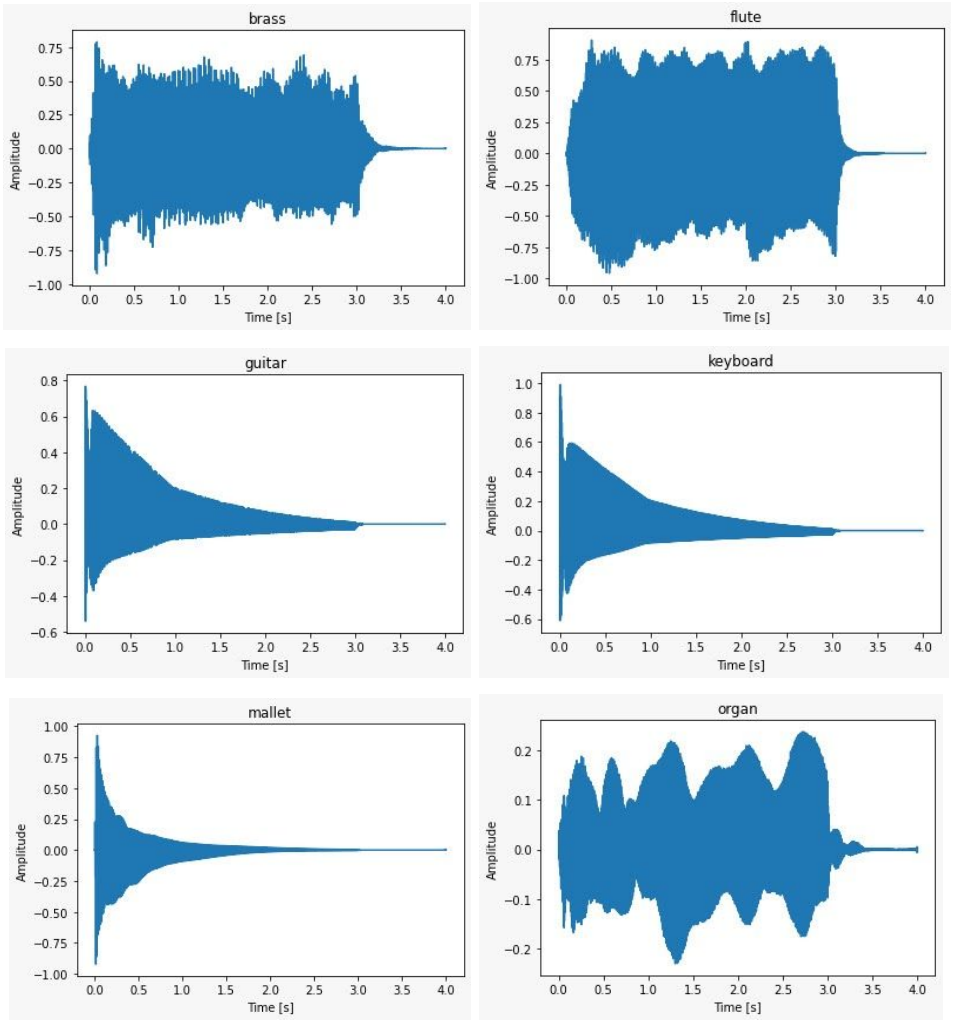
Confusion matrix:

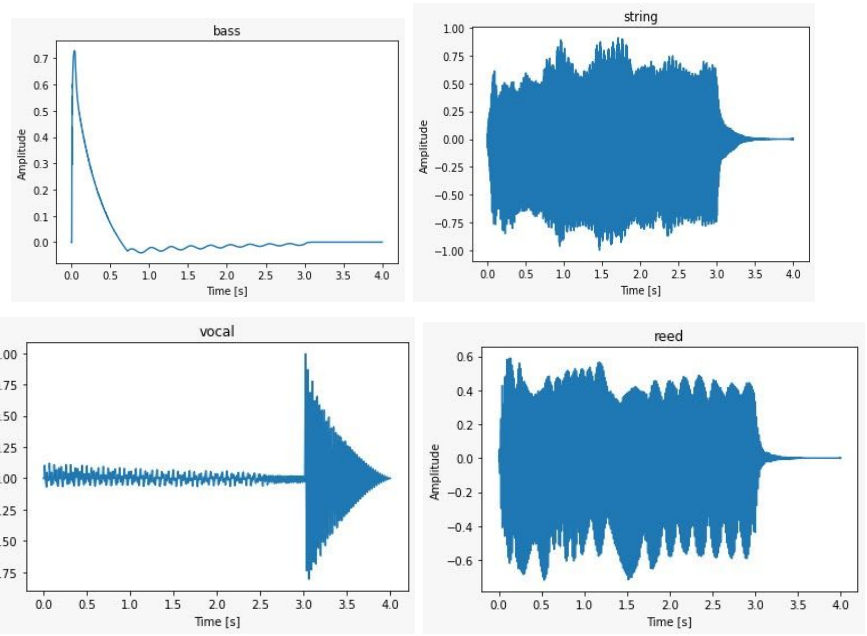
```
[[461  14   5   4 317   0  24  12   4   2]
 [  9 153   7   0  77   0   8  14   0   1]
 [ 44  43   7   0  84   0   2   0   0   0]
 [ 77  25   9  13 482   0  22   9  15   0]
 [ 99  12   0  38 612   0   3   0   2   0]
 [ 44   0   0   8 150   0   0   0   0   0]
 [ 29  82  11   1  97   0 242  25   6   9]
 [  1  62   1   0 100   0  19  51   0   1]
 [ 48  44   0   4 126   0  34   2  38  10]
 [  0  10   0   0   3   0  16   8   0 104]]
```

Classification report:

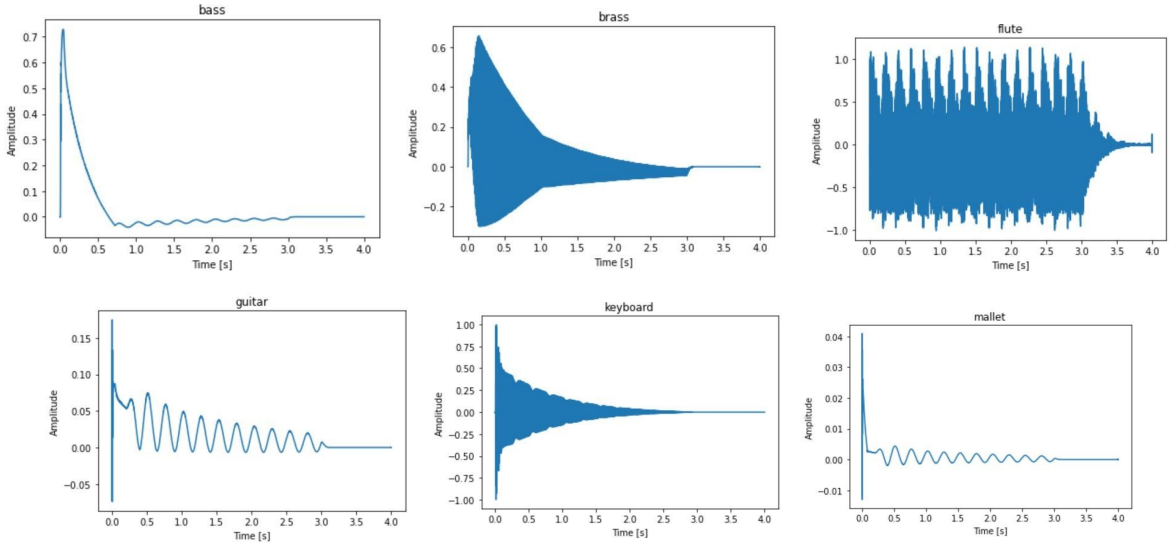
	precision	recall	f1-score	support
bass	0.57	0.55	0.56	843
brass	0.34	0.57	0.43	269
flute	0.17	0.04	0.06	180
guitar	0.19	0.02	0.04	652
keyboard	0.30	0.80	0.43	766
mallet	0.00	0.00	0.00	202
organ	0.65	0.48	0.56	502
reed	0.42	0.22	0.29	235
string	0.58	0.12	0.20	306
vocal	0.82	0.74	0.78	141
accuracy			0.41	4096
macro avg	0.41	0.35	0.33	4096
weighted avg	0.41	0.41	0.36	4096

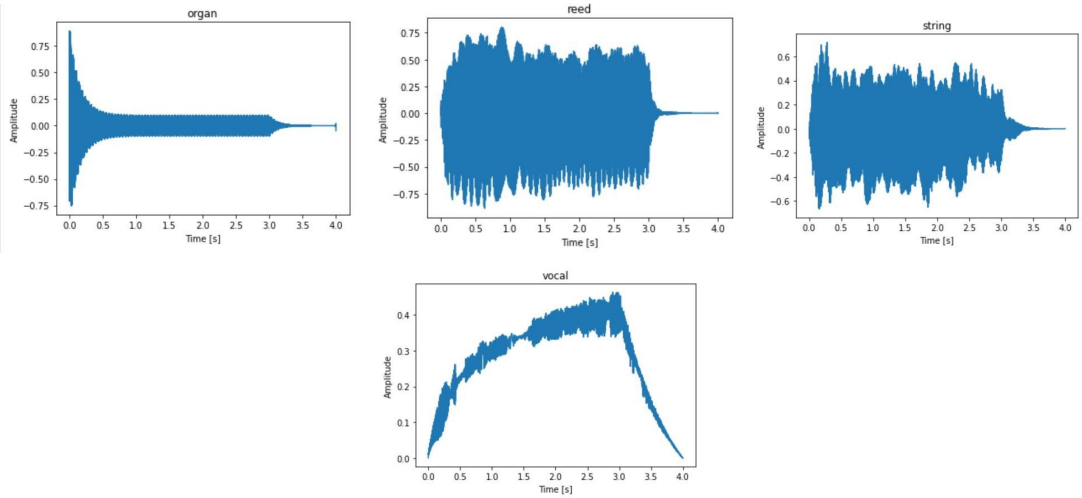
Waveforms for samples of each class where probability is either greater than .9 or less than .1 (preliminary model):



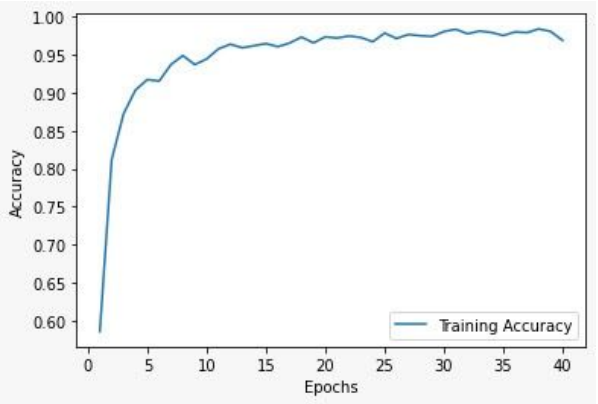


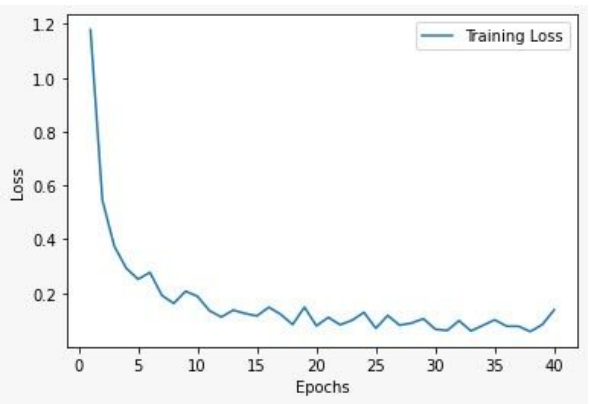
Waveforms for samples near the decision boundary for each class where the probability for the correct class is slightly higher/lower than other class (preliminary model):





Visualizations of results for our optimized model:





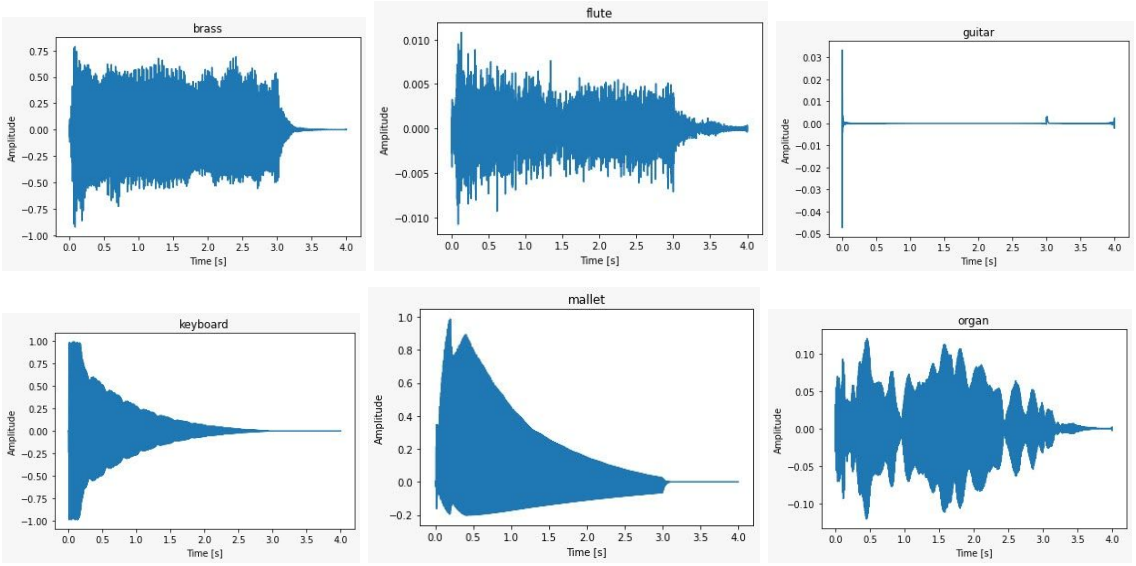
Confusion matrix:

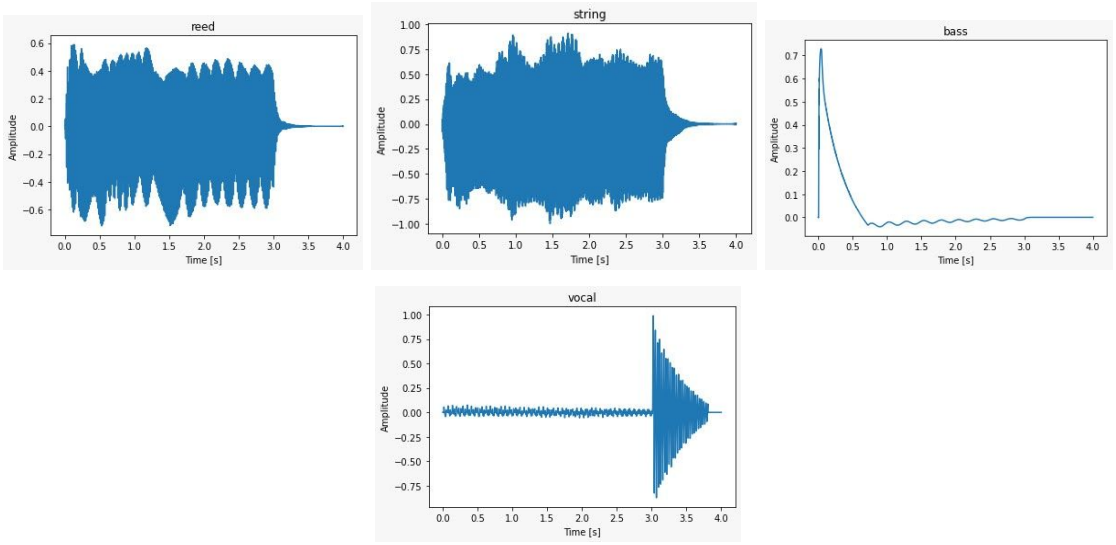
```
[[813   0   0   6  14   8   2   0   0   0]
 [  0 214   4   2   1   0  16  25   3   4]
 [  0   2 163   0   1   0  11   1   1   1]
 [  3   0   0 597  42   6   1   0   0   3]
 [ 13   0   0  50 678  15   6   1   3   0]
 [  8   0   0   3  20 166   5   0   0   0]
 [  1   2   3   4   4   0 456  15   3  14]
 [  0  17   1   0   2   0  11 199   3   2]
 [  6   4   1   4   5   2  10   6 268   0]
 [  0   0   0   0   0   0   2   0   0 139]]
```

Classification report:

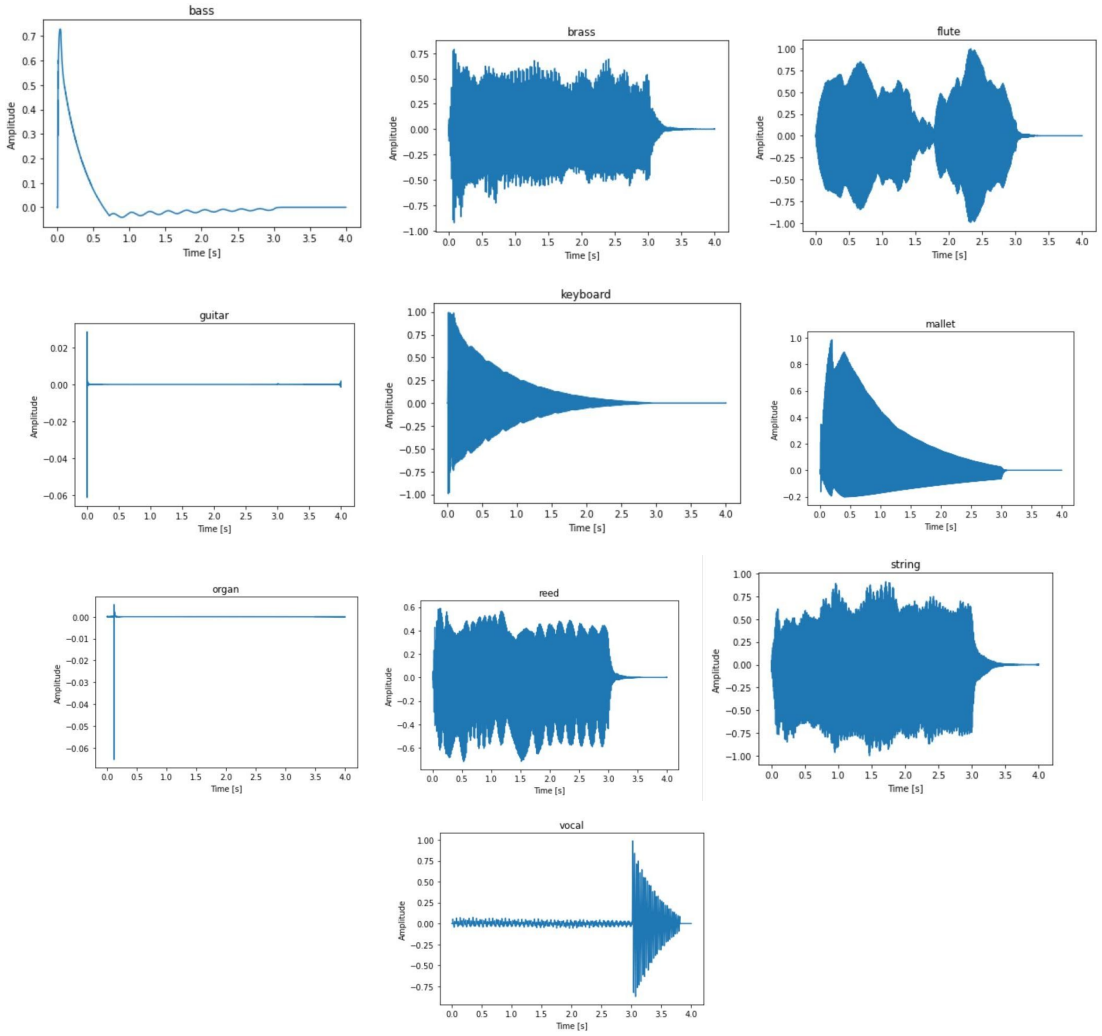
	precision	recall	f1-score	support
bass	0.96	0.96	0.96	843
brass	0.90	0.80	0.84	269
flute	0.95	0.91	0.93	180
guitar	0.90	0.92	0.91	652
keyboard	0.88	0.89	0.88	766
mallet	0.84	0.82	0.83	202
organ	0.88	0.91	0.89	502
reed	0.81	0.85	0.83	235
string	0.95	0.88	0.91	306
vocal	0.85	0.99	0.91	141
accuracy			0.90	4096
macro avg	0.89	0.89	0.89	4096
weighted avg	0.90	0.90	0.90	4096

Waveforms for samples of each class where probability is either greater than .9 or less than .1 (optimized model):





Waveforms for samples near the decision boundary for each class where the probability for the correct class is slightly higher/lower than other class (optimized model):



5. Discussion:

Immense increase in accuracy was found between the initial model and the optimized model. As the optimized model catered to the shortcomings of the initial model through use of optimized activation functions such as LeakyReLU instead of ReLU to tackle possible vanishing gradients, deliberate induction of non-linearity through reducing and increasing number of nodes between the convolutional layers and addition of extra layers in convolution and classification stages made feasible by dimensionality reduction through max pooling after the convolutional stage allowing for acceptable train times were some of the attempts which were successful to improve the initial model. The optimized further employed a dropout layer[6] to avoid overfitting before flattening. Experimentation led to finding that an extra layer between the flattened layer and output classification layer leads to increase in accuracy without affecting the training time to a vast extent.

References:

1. https://en.wikipedia.org/wiki/Vanishing_gradient_problem
2. https://en.wikipedia.org/wiki/Softmax_function
3. https://en.wikipedia.org/wiki/Rectifier_(neural_networks)#Leaky_ReLU
4. https://cs231n.github.io/neural-networks-1/
5. https://cs231n.github.io/convolutional-networks/
6. https://en.wikipedia.org/wiki/Convolutional_neural_network#Dropout
7. https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.resample.html
8. https://machinelearningmastery.com/how-to-configure-the-number-of-layers-and-nodes-in-a-neural-network/
9. Jesse Engel, Cinjon Resnick, Adam Roberts, Sander Dieleman, Douglas Eck, Karen Simonyan, and Mohammad Norouzi. "Neural Audio Synthesis of Musical Notes with WaveNet Autoencoders." 2017.
10. https://keras.io/
11. https://www.tensorflow.org/api_docs/python/
12. https://www.freecodecamp.org/news/an-intuitive-guide-to-convolutional-neural-networks-260c2de0a050/
13. https://medium.com/@mikesmales/sound-classification-using-deep-learning-8bc2aa1990b7
14. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html
15. https://magenta.tensorflow.org/datasets/nsynth