Topic: Build Audio-to-Text Model

Chia An Kuo cak580

Abstract

Recently, many papers have discussed the deep learning in computer vision to improve the performance of image detection. I would like to integrate what I have learned in audio signal processing with machine learning. The purpose of this project is to review several models of machine learning, ultimately, decide which model to use for the audio process project. I have selected CNN, ANN, MAX POOLING for literature review and DIGITAL AUDIO EFFECTS textbook as basic knowledge of audio processing. TensorFlow has released the Speech Commands Datasets. It includes 65,000 one-second long utterances of 30 short words, by thousands of different people. The purpose of the project is trying build an audio recognition system that could identify simple words.

Key words: Machine Learning, CNN, Max Pooling, Audio Signal Processing

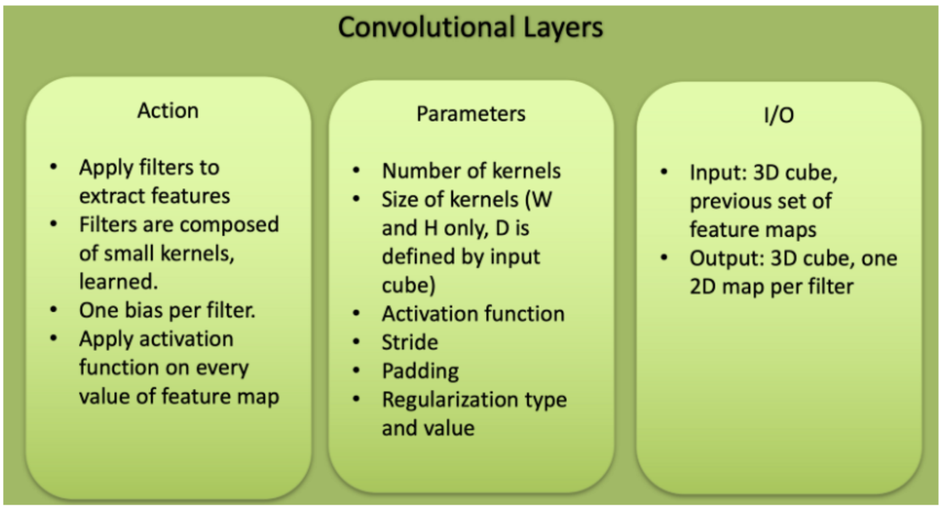
I. Introduction

In nowadays technology, People simply ask the question and Google lays out the entire sentence for us. This tech sounds familiar and fascinating. I can't even remember the last time I type out the entire query on Google Search.
This is the concept of audio-to-text models. Google uses a mix of deep learning and Natural Language Processing (NLP) techniques to parse through our query, retrieve the answer and present it in the form of both audio and text.
In order to understand machine learning, I hope to build audio-to-text model using my Python and deep learning skills myself. It's a fascinating concept which I could integrate what I have learned in DSP. Last year, I have made a project with image processing with deep learning(R-CNN). I think it would be very interested if I could also use the machine learning with audio signal which I learned last semester.
Different from image processing, we need to convert the audio signals to discrete signals through sampling so that it can be stored and processed efficiently in memory so that we can work with them easily.

II. Literature reviews

1. CNN

There are three types of layers in a convolutional neural network: convolutional layer, pooling layer, and fully connected layer.
Convolutional layers are the layers where filters are applied to the original image, or to other feature maps in a deep CNN. This is where most of the user-specified parameters are in the network. The most important parameters are the number of kernels and the size of the kernels.



Convolutional Layers

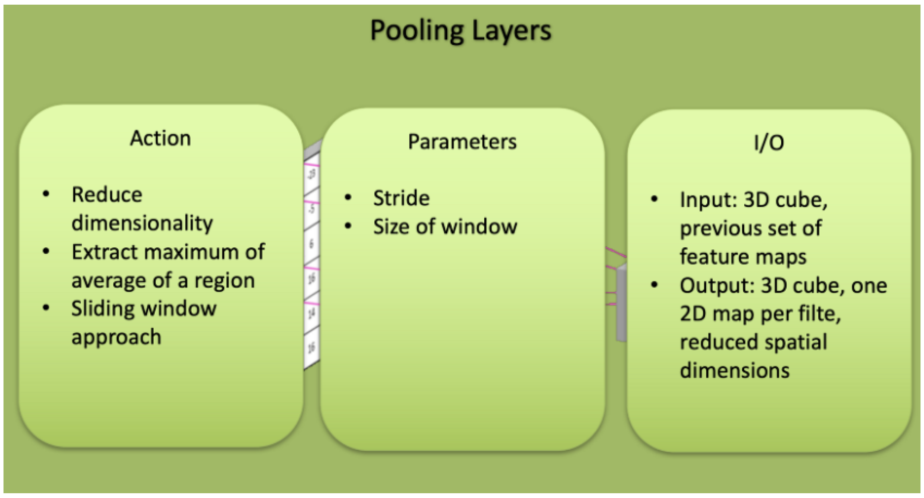
Action
- Apply filters to extract features
- Filters are composed of small kernels, learned.
- One bias per filter.
- Apply activation function on every value of feature map

Parameters
- Number of kernels
- Size of kernels (W and H only, D is defined by input cube)
- Activation function
- Stride
- Padding
- Regularization type and value

I/O
- Input: 3D cube, previous set of feature maps
- Output: 3D cube, one 2D map per filter

Features of a convolutional layer.

Pooling layers are similar to convolutional layers, but they perform a specific function such as max pooling, which takes the maximum value in a certain filter region. It is typically used to reduce the dimensionality of the network.
Pooling is a way to take large images and shrink them down while preserving the most important information in them.


Pooling Layers

Action

- Reduce dimensionality
- Extract maximum of average of a region
- Sliding window approach

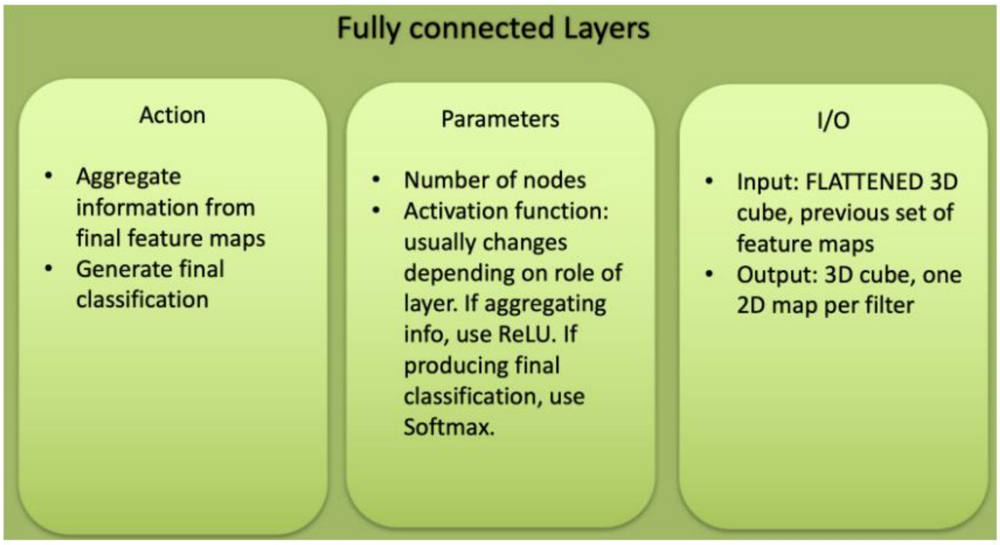
Parameters

- Stride
- Size of window

I/O

- Input: 3D cube, previous set of feature maps
- Output: 3D cube, one 2D map per filte, reduced spatial dimensions

Fully connected layers are placed before the classification output of a CNN and are used to flatten the results before classification.


Fully connected Layers

Action

- Aggregate information from final feature maps
- Generate final classification

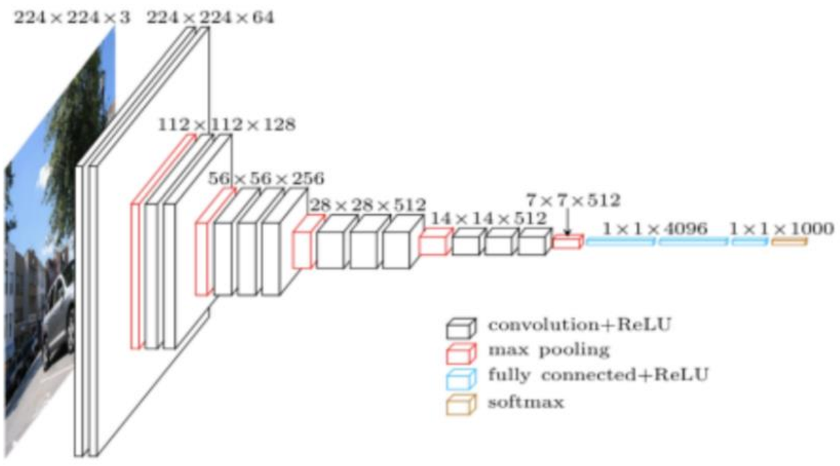
Parameters

- Number of nodes
- Activation function: usually changes depending on role of layer. If aggregating info, use ReLU. If producing final classification, use Softmax.

I/O

- Input: FLATTENED 3D cube, previous set of feature maps
- Output: 3D cube, one 2D map per filter

Features of a fully connected layer.


$224 \times 224 \times 3$
$224 \times 224 \times 64$
$112 \times 112 \times 128$
$56 \times 56 \times 256$
$28 \times 28 \times 512$
$14 \times 14 \times 512$
$7 \times 7 \times 512$
$1 \times 1 \times 4096$
$1 \times 1 \times 1000$
convolution+ReLU
max pooling
fully connected+ReLU
softmax

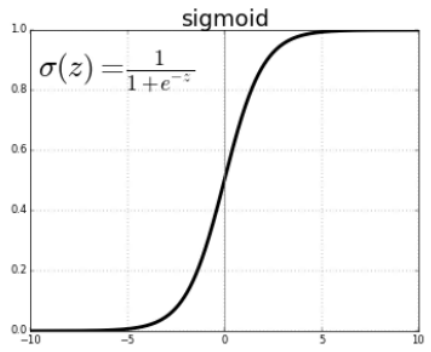
The architecture of a standard CNN.

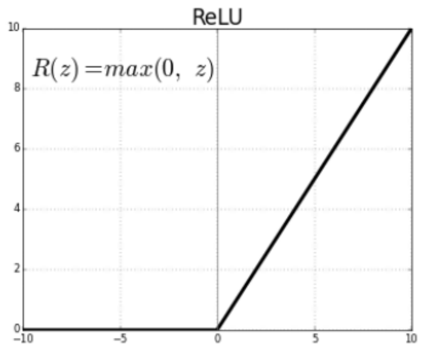
2. What do CNN layers learn?

Each CNN layer learns filters of increasing complexity.
The first layers learn basic feature detection filters.
The middle layers learn filters that detect parts of objects.
The last layers have higher representations: they learn to recognize full objects, in different shapes and positions.

3. Relu activation function/Sigmoid activation function

What is an Activation function?
We picked an activation function for each layer. It takes that ((w • x) + b) and calculates a probability. Then it sets a threshold to determine whether the neuron ((w • x) + b) should be 1 (true) or (0) negative.
Relu is 1 for all positive values and 0 for all negative ones.
Sigmoid uses the logistic function, 1 / (1 + e**z) where z = f(x) = ((w • x) + b).





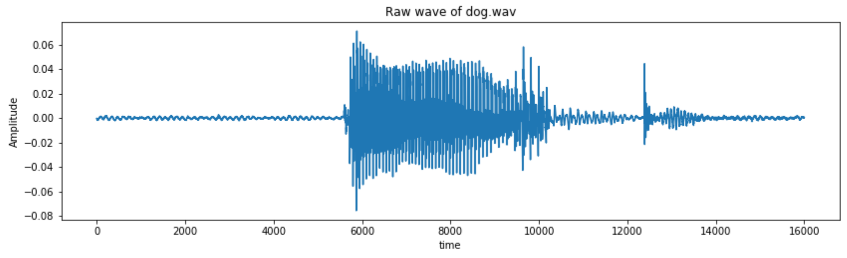
The most successful non-linearity for CNN's is the Rectified Non-Linear unit (ReLU), which combats the vanishing gradient problem occurring in sigmoids. ReLU is easier to compute and generates sparsity.

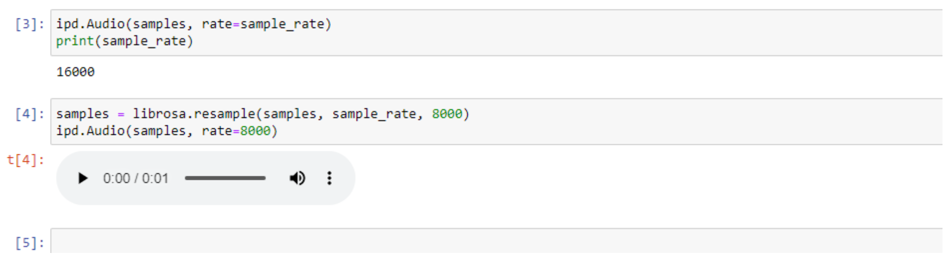
4. Random Forest

The RF is the ensemble of decision trees. Each decision tree, in the ensemble, process the sample and predicts the output label. Decision trees in the ensemble are independent. Each can predict the final response.

III. Methods

1. Importing libraries in Python
   - os
   - librosa #for audio processing
   - numpy
   - matplotlib.pyplot
   -scipy import wavfile #for audio processing
2. Visualizing the audio data, Sampling the data and resampling.
   Visualized the data to make sure the dataset is loaded.



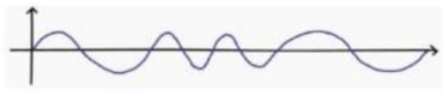


```
[3]: ipd.Audio(samples, rate=sample_rate)
     print(sample_rate)

     16000

[4]: samples = librosa.resample(samples, sample_rate, 8000)
     ipd.Audio(samples, rate=8000)
t[4]:
     ▶ 0:00 / 0:01

[5]:
```

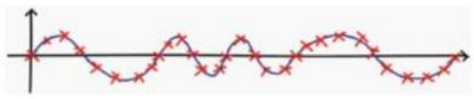
From the graph above, the sampling rate is 16,000 Hz. Resample it to 8000 Hz, since most of the speech frequencies are present at 8000 Hz.

(Why sampling the audio signal?
Since audio signal is continuous, we need to convert the analog signals to digital signals so that we can work with them easily. This is called sampling the signal, in order to make it become discrete. The sampling rate or sampling frequency is the number of samples selected per second.)

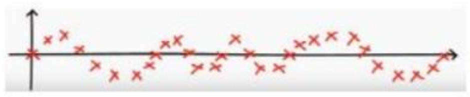
Step 1: Analog audio signal - Continuous representation of signal



Step 2: Sampling - Samples are selected at regular time intervals

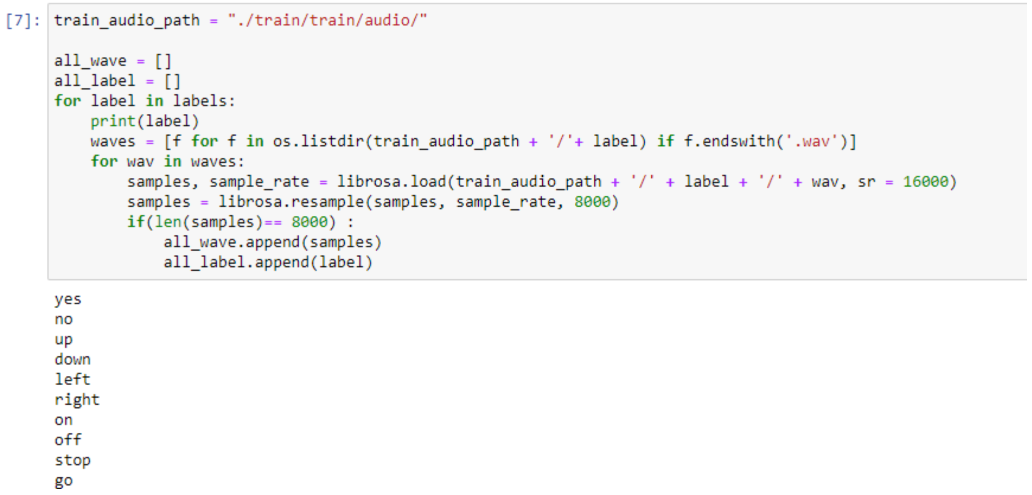


Step 3: Digital audio signal - The way it is stored in memory



3. Preprocess the data

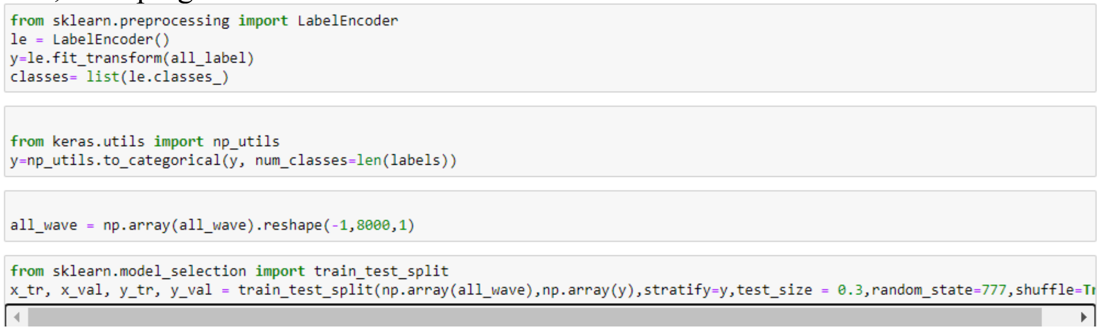
First, convert all the data to sampling rate 8000 Hz and remove the data which is less than one second. There are 10 classes in my project. Each class is a very simple vocabulary.



```
[7]: train_audio_path = "./train/train/audio/"

     all_wave = []
     all_label = []
     for label in labels:
         print(label)
         waves = [f for f in os.listdir(train_audio_path + '/'+ label) if f.endswith('.wav')]
         for wav in waves:
             samples, sample_rate = librosa.load(train_audio_path + '/' + label + '/' + wav, sr = 16000)
             samples = librosa.resample(samples, sample_rate, 8000)
             if(len(samples)== 8000) :
                 all_wave.append(samples)
                 all_label.append(label)
```

```
yes
no
up
down
left
right
on
off
stop
go
```

Then, convert it to integer by importing LabelEncoder library.
Last, since it is a muti-classification problem, converting the integer encoded labels to one-hot vector by importing np_utils library.
Last, reshaping the data from 2D to 3D in order to train the CNN model later.



```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
y=le.fit_transform(all_label)
classes= list(le.classes_)
```

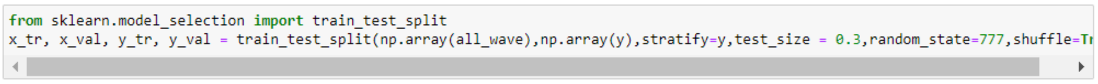
```
from keras.utils import np_utils
y=np_utils.to_categorical(y, num_classes=len(labels))
```

```
all_wave = np.array(all_wave).reshape(-1,8000,1)
```

```
from sklearn.model_selection import train_test_split
x_tr, x_val, y_tr, y_val = train_test_split(np.array(all_wave),np.array(y),stratify=y,test_size = 0.3,random_state=777,shuffle=Tr
```

4. Split into train and test set

Train the model on 70% of the data and validate on the remaining30% by importing train_test_split library.

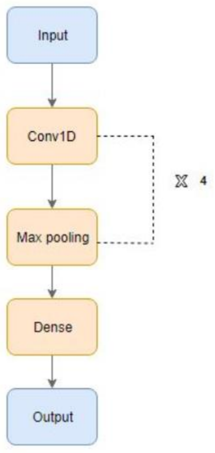


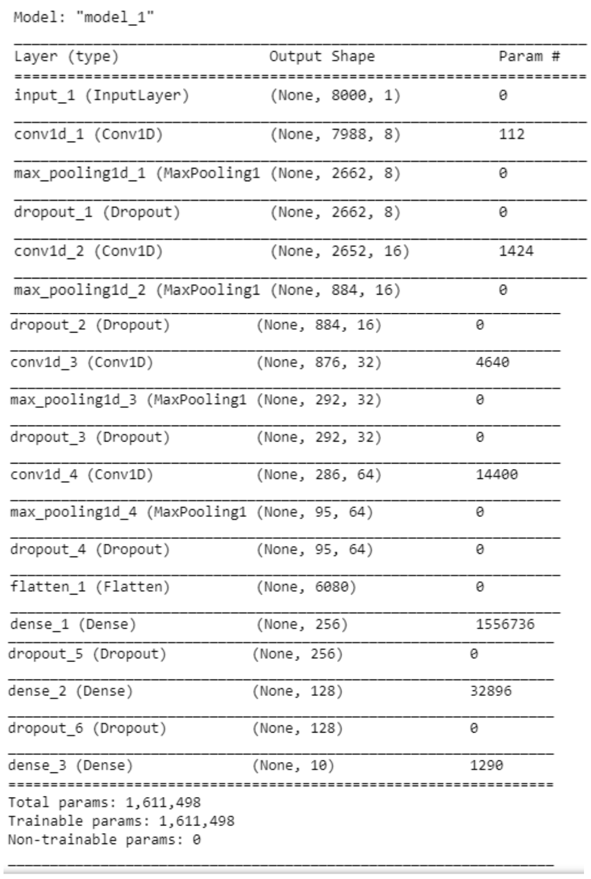
```
from sklearn.model_selection import train_test_split
x_tr, x_val, y_tr, y_val = train_test_split(np.array(all_wave),np.array(y),stratify=y,test_size = 0.3,random_state=777,shuffle=Tr
```

5. Implementing the machine learning models

Model Architecture:

A. CNN with the Relu activation function.





```
Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 8000, 1)           0
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 7988, 8)           112
_________________________________________________________________
max_pooling1d_1 (MaxPooling1 (None, 2662, 8)           0
_________________________________________________________________
dropout_1 (Dropout)          (None, 2662, 8)           0
_________________________________________________________________
conv1d_2 (Conv1D)            (None, 2652, 16)          1424
_________________________________________________________________
max_pooling1d_2 (MaxPooling1 (None, 884, 16)           0
_________________________________________________________________
dropout_2 (Dropout)          (None, 884, 16)           0
_________________________________________________________________
conv1d_3 (Conv1D)            (None, 876, 32)           4640
_________________________________________________________________
max_pooling1d_3 (MaxPooling1 (None, 292, 32)           0
_________________________________________________________________
dropout_3 (Dropout)          (None, 292, 32)           0
_________________________________________________________________
conv1d_4 (Conv1D)            (None, 286, 64)           14400
_________________________________________________________________
max_pooling1d_4 (MaxPooling1 (None, 95, 64)            0
_________________________________________________________________
dropout_4 (Dropout)          (None, 95, 64)            0
_________________________________________________________________
flatten_1 (Flatten)          (None, 6080)              0
_________________________________________________________________
dense_1 (Dense)              (None, 256)               1556736
_________________________________________________________________
dropout_5 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_2 (Dense)              (None, 128)               32896
_________________________________________________________________
dropout_6 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_3 (Dense)              (None, 10)                1290
=================================================================
Total params: 1,611,498
Trainable params: 1,611,498
Non-trainable params: 0
_________________________________________________________________
```

- Conv1D:

A 1-Dimensional convolution layer applies a convolution on two 1-dimensional data types (vectors, signals)

- Max Pooling:

Maximum Pooling is the process of splitting the data into several regions, taking the maximum value of a cell in those regions, and inserting them into an output matrix.

- Dropout Layer:

The Dropout layer, used to prevent overfitting in neural networks, randomly sets a fraction ‘rate’ of input units to 0 at each update during training time.

- Flatten Layer:

The Flatten layer converts multi-dimensional data into a single vector to be processed.

- Dense:

The Dense layer is a standard neural network layer.

B.CNN with sigmoid activation function.
C.Random Forest Tree with Entropy, Max depth(20/60)

6. Predict and see the accuracy of test set

IV. Results

Here are the results.

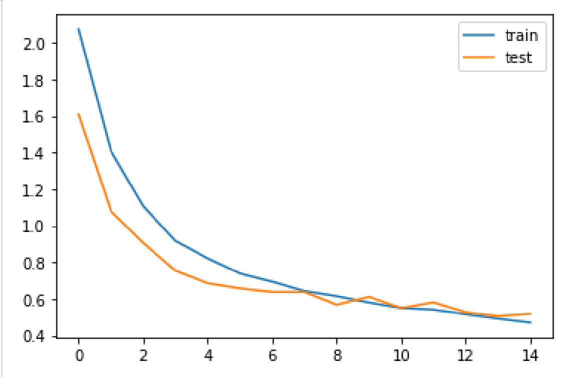
- CNN:

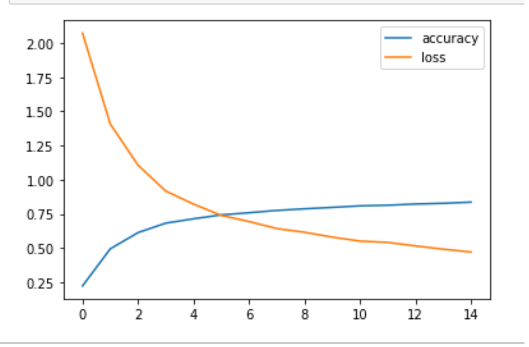
  After training for 15 epochs and with batch size32, we evaluate the model.

After 15 epochs with Relu activation function, the accuracy comes to 0.8286 and the loss is 0.4703 which performs very well.

```
y: 0.8281
Epoch 15/15
14918/14918 [==============================] - 70s 5ms/step - loss: 0.4703 - accuracy: 0.8353 - val_loss: 0.5174 - val_accurac
y: 0.8286
```

The following is the loss of the training set and testing set and the accuracy and loss of the data.

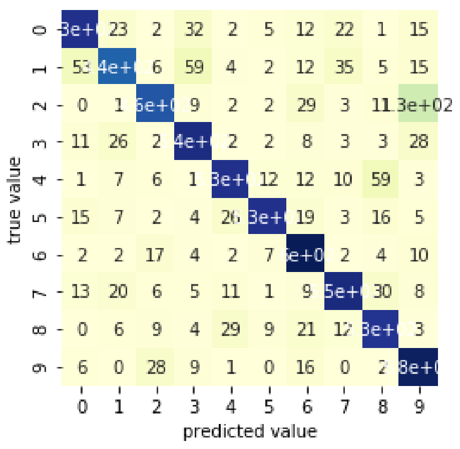




confusion matrix and heatmap:

```
Text(91.68, 0.5, 'true value')
```

```
              precision    recall  f1-score   support

           0       0.84      0.82      0.83       646
           1       0.83      0.70      0.76       630
           2       0.86      0.71      0.77       650
           3       0.81      0.86      0.84       629
           4       0.87      0.83      0.85       643
           5       0.93      0.85      0.89       631
           6       0.81      0.92      0.86       647
           7       0.86      0.84      0.85       652
           8       0.80      0.85      0.82       619
           9       0.73      0.90      0.81       647

    accuracy                           0.83      6394
   macro avg       0.83      0.83      0.83      6394
weighted avg       0.83      0.83      0.83      6394

[[532  23   2  32   2   5  12  22   1  15]
 [ 53 439   6  59   4   2  12  35   5  15]
 [  0   1 460   9   2   2  29   3  11 133]
 [ 11  26   2 544   2   2   8   3   3  28]
 [  1   7   6   1 532  12  12  10  59   3]
 [ 15   7   2   4  26 534  19   3  16   5]
 [  2   2  17   4   2   7 597   2   4  10]
 [ 13  20   6   5  11   1   9 549  30   8]
 [  0   6   9   4  29   9  21  12 526   3]
 [  6   0  28   9   1   0  16   0   2 585]]
```



Confusion matrix C is such that c(i,j) is equal to the number of observations known to be in group i but predicted to be in group j.
As we can see the relation between the predicted value and true value. Which the dark blue line means the prediction is good.
And the last part of the code, we first randomly play a wave file, then print out the prediction of our model.

the prediction performs well.

```python
def predict(audio):
    prob=model.predict(audio.reshape(1,8000,1))
    index=np.argmax(prob[0])
    return classes[index]
```

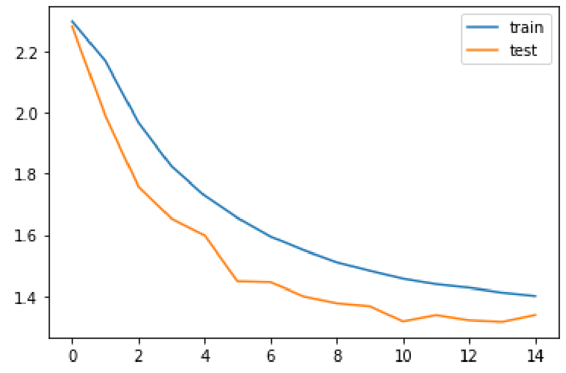
```python
import random
index=random.randint(0,len(x_val)-1)
samples=x_val[index].ravel()
print("Audio:",classes[np.argmax(y_val[index])])
ipd.Audio(samples, rate=8000)
```

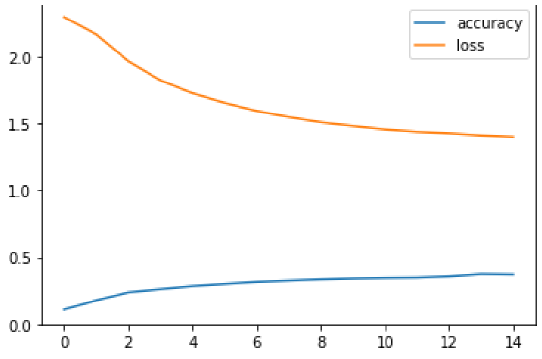
```
Audio: go
```

```python
print("Text:",predict(samples))
```

```
Text: go
```

- CNN with sigmoid function: accuracy:0.3918
  Below is loss of the training set and testing set
  And the and the accuracy and loss of the data.



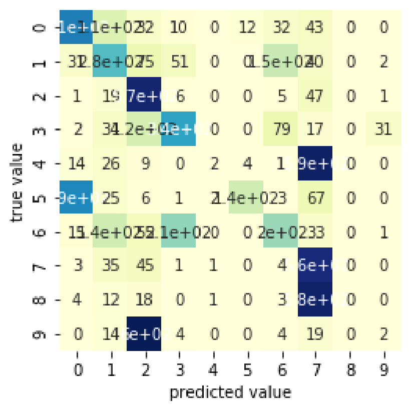


```
14918/14918 [==============================] - 99s 7ms/step - loss: 1.4112 - accuracy: 0.3723 - val_loss: 1.3149 - val_accurac
y: 0.4090
Epoch 15/15
14918/14918 [==============================] - 88s 6ms/step - loss: 1.4002 - accuracy: 0.3692 - val_loss: 1.3376 - val_accurac
y: 0.3918
```

confusion matrix and heatmap:

```
              precision    recall  f1-score   support

           0       0.47      0.63      0.54       646
           1       0.41      0.45      0.43       630
           2       0.37      0.88      0.52       650
           3       0.55      0.55      0.55       629
           4       0.33      0.00      0.01       643
           5       0.89      0.22      0.35       631
           6       0.42      0.31      0.35       647
           7       0.28      0.86      0.43       652
           8       0.00      0.00      0.00       619
           9       0.05      0.00      0.01       647

    accuracy                           0.39      6394
   macro avg       0.38      0.39      0.32      6394
weighted avg       0.38      0.39      0.32      6394

[[407 110  32  10   0  12  32  43   0   0]
 [ 31 282  75  51   0   0 149  40   0   2]
 [  1  19 571   6   0   0   5  47   0   1]
 [  2  34 123 343   0   0  79  17   0  31]
 [ 14  26   9   0   2   4   1 587   0   0]
 [391  25   6   1   2 136   3  67   0   0]
 [ 15 136  55 208   0   0 199  33   0   1]
 [  3  35  45   1   1   0   4 563   0   0]
 [  4  12  18   0   1   0   3 581   0   0]
 [  0  14 604   4   0   0   4  19   0   2]]
```

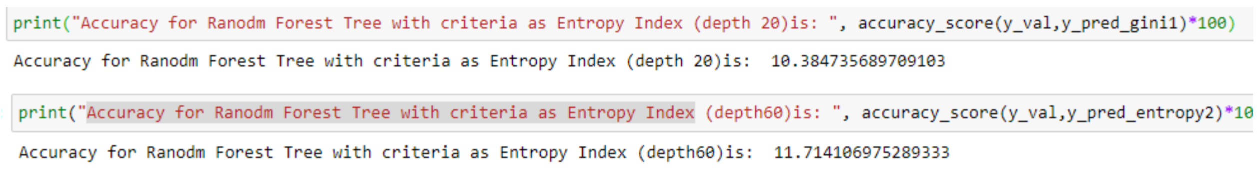
```
Text(91.68, 0.5, 'true value')
```



The output of sigmoid saturates for a large positive or large negative number. Thus, the gradient at these regions is almost zero. During backpropagation, this local gradient is multiplied with the gradient of this gates' output. Thus, if the local gradient is very small, it'll kill the gradient and the network will

not learn. This problem of vanishing gradient is solved by ReLU. That's the reason why the accuracy is so low when we use sigmoid activation function.
Moreover, sigmoid outputs are not zero-centered, it can indirectly introduce undesirable zig-zagging dynamics in the gradient updates for the weights. Then leads to the bad performance.

- Random Forest:
  Since Random Forest classifier is not suitable for the audio signal to text model, the accuracy turns out to be bad (10%).
  Decision could only work with data with table format, which CNN can be implemented in Images, audio and text.
  Since there is not a strong, qualitatively important relationship among the features in the sense of the data being an image, or audio. These structures are typically not well-approximated by many rectangular partitions. The data live in a time series, or are a series of images, the random forest will have a very hard time recognizing that.



```
print("Accuracy for Ranodm Forest Tree with criteria as Entropy Index (depth 20)is: ", accuracy_score(y_val,y_pred_gini1)*100)

Accuracy for Ranodm Forest Tree with criteria as Entropy Index (depth 20)is:  10.384735689709103

print("Accuracy for Ranodm Forest Tree with criteria as Entropy Index (depth60)is: ", accuracy_score(y_val,y_pred_entropy2)*10

Accuracy for Ranodm Forest Tree with criteria as Entropy Index (depth60)is:  11.714106975289333
```

V. Improvement

After listening to other student's presentation, I can try to extract the features of each sound. So that I can store the data in a table. Then, use random forest model to train the table format data rather than the raw sound.

VI. References

https://www.analyticsvidhya.com/blog/2019/07/learn-build-first-speech-to-text-model-python/
https://www.kaggle.com/c/tensorflow-speech-recognition-challenge/data?select=train.7z
https://github.com/aravindpai/Speech-Recognition/blob/master/Speech%20Recognition.ipynb
https://github.com/seth814/Audio-Classification
https://github.com/drscotthawley/panotti
https://en.wikipedia.org/wiki/Convolutional_neural_network
https://towardsdatascience.com/simple-introduction-to-convolutional-neural-networks-cdf8d3077bac
Spatial Pyramid Pooling in Deep Convolutional
Networks for Visual Recognition
https://medium.com/analytics-vidhya/a-guide-to-neural-network-layers-with-applications-in-keras-40ccb7ebb57a

VII. Codes and Data link:

https://drive.google.com/drive/folders/1c_4b08CqJglBu-Snf_Co2rLXSHC_UKVD?usp=sharing