Preprocessing

We will read the audio file using a package called wavio which is a python module that defines two functions wavio.write() and wavio.read(). We have used wavio.read() to read each audio file that returned sampling rate and data in numpy array format. The data is stored as a two-dimensional array with the

//Reading the audio file: shape being (None,2) where 'None' represents the arbitrary number of rows and '2' represents the number of columns. The first column is considered as Speaker 1 data and the second column is considered as speaker 2 data.

//Code snippet….

//Down sampling

The sampling rate was found to be 44100. As the interval size is 20milli seconds, the down sampling is computed as follows:

Down sampling = (Sampling rate*20)/1000

So, we got the down sampling value to be 882.

//Code snippet…..

We have the intervals as (0ms-20ms), (10ms-30ms),……such that the overlapped interval size is 10milli seconds.

We have implemented this as follows:

//Code snippet….

//Training Data

Every interval is represented as (i,j). The initial value of i=0 and j=882.The range(0,882) represents 20ms data of each speaker. For each iteration, i and j are incremented by (down sampling value)/2= 882/2=441. The process repeats until j<=number of rows for both the speakers.

//Code snippet…

Now we have the training data of the speaker 1 in p1 numpy array and that of speaker 2 in p2 numpy array.

//Code snippet

//Reading the csv files

Now we will read the csv files of speaker 1 and speaker 2 individually using pd.read_csv().

The unit of the time columns tmin and tmax in the csv file is seconds. We are converting the unit to 10 milli seconds by multiplying it with 100 because our overlapped interval is 10milli seconds.

Now we have taken an empty list t1= [ ].

Let us consider the speaker 1 data of the first audio file which has (tmin, tmax) values to be(0seconds , 6.083seconds) which is equivalent to (0,608.3 10milli seconds) and the values are rounded to the nearest integer.

If this range 0-608.3 10milliseconds is non-speech, which holds the label "0", then the empty list t1 is appended with 608 zeros which is calculated as t1=t1+608*[0]

Now moving to the second row in the csv file, let us say we have tmin and tmax values to be 608 and 1100, with the label to be "1", now we will calculate t1 as: t1=608*[0]+((1100-608)*[1]).

This process repeats till the end of the file.

//Code snippet…

Similarly, we will repeat the above process for other speaker 2 data as well.

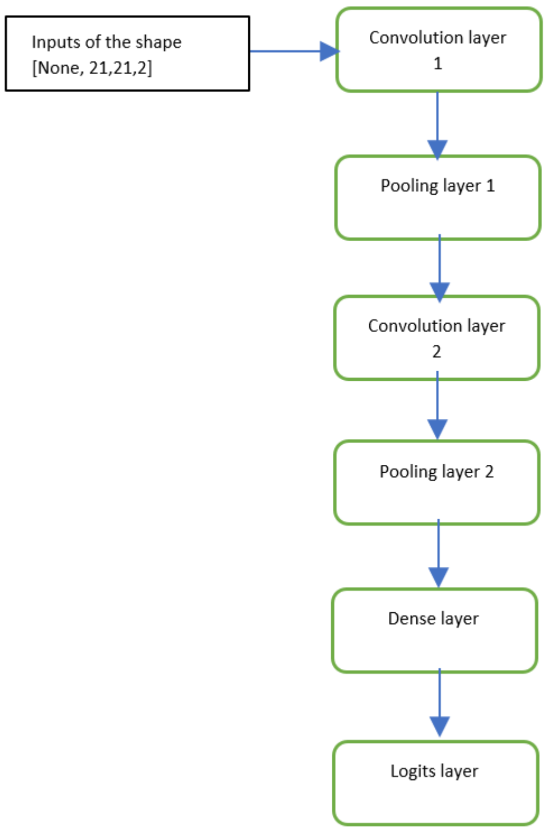
//Code snippet…

Now we have a list of speech and non-speech labels over a 10ms interval as [0ms-10ms], [10ms-20ms] and ... Now we will need to convert this list into 20ms intervals with 10ms overlap as [0ms- 20ms], [10ms -30ms],….To evaluate the interval to 20ms with 10ms overlap, we have taken the minimum value between the 0ms to 10ms and 10ms to 20ms.

Then we build a convolution neural network which has a learning rate of 0.001 with Adam optimizer and batch_size=100.

CNN ARCHITECTURE

A Convolution neural network is composed of a stack of convolutional modules that perform feature extraction. Each module consists of a convolutional layer followed by a pooling layer. The last convolutional module is followed by one or more dense layers that performs the classification. The final dense layer in a CNN contains a single node for each target class in the model with a softmax activation function that generates a value between 0 – 1, for which, we can interpret the softmax values for a given data. We can then predict in which target class (Speech or Non-speech) will our data falls into.



Number of layers – six layers of which there are two convolution layers, two max pooling layers, one dense layer and an output layer.

We used softmax activation function in our output layer which is a generalization of the logistic function that "squashes" a K-dimensional vector {z} of arbitrary real values to a K-dimensional vector of real values in the range [0, 1] that add up to 1.

Convolutional Layer #1: In this layer, we applied 32 5x5 filters by extracting 5x5-pixel sub-regions with ReLU activation function. The kernel size is [5,5].

Pooling Layer #1: In this layer, we perform the max pooling with a 2x2 filter and stride of 2 which specifies that the pooled regions do not overlap.

Convolutional Layer #2: In this layer, we apply 64 5x5 filters with ReLU activation function. The kernel size is [5,5].

Pooling Layer #2: In this layer, again we perform the max pooling with a 2x2 filter and stride of 2.

Dense Layer #1: 1024 neurons with dropout regularization rate of 0.4 (probability of 0.4 that any given element will be dropped during training)

Logits Layer: In this layer we have 2 neurons, one for each target class 0 for non-speech and 1 for speech.

We have used the following functions in our model to create each type of layer:

conv2d(): This functions constructs a two-dimensional convolutional layer by taking number of filters, filter kernel size, padding, and activation function as arguments.

max_pooling2d() – This functions constructs a two-dimensional pooling layer using the max-pooling algorithm by taking pooling filter size and stride as arguments.

dense() – This function constructs a dense layer by taking the number of neurons and activation function as arguments.

tf.nn_sparse_softmax_cross_entropy_with_logits():

We have used tf.nn_sparse_softmax_cross_entropy_with_logits() as our cost function which measures the probability error in discrete classification tasks in which the classes are mutually exclusive.

Our Neural Network functions as follows:

Step 1:

By using the conv2D we will create a convolution layer 1 to which we will pass our input of the shape [None, 21,21,2]. Then we will apply 32 5*5 convolution filters to the data. For each sub-region, the layer performs a set of mathematical operations to produce a single value in the output feature map which then applies ReLU activation function to the output.

Step 2:

The generated output is then passed as an input to the max pooling layer 1 in which we down-sample our data that is extracted by the convolutional layers to reduce the dimensionality of the feature map in order to decrease processing time. We used max pooling algorithm that extracts sub-regions of the feature map(2x2-pixel) to keep the maximum value and discards all other values.

Step 3:

The output that is simulated by the pooling layer 1 is then passed as an input to the convolution layer 2 which again applies 32 5*5 convolution filters to the data for which, for each sub-region, the layer performs a set of mathematical operations to produce a single value in the output feature map which then applies ReLU activation function to the output.

Step 4:

This output is again passed as an input to the pooling layer 2 in which we again down-sample our data that is extracted by the convolutional layers to reduce the dimensionality of the feature map in order to decrease processing time. We used max pooling algorithm that extracts sub-regions of the feature map(2x2-pixel) to keep the maximum value and discards all other values.

Step 5:

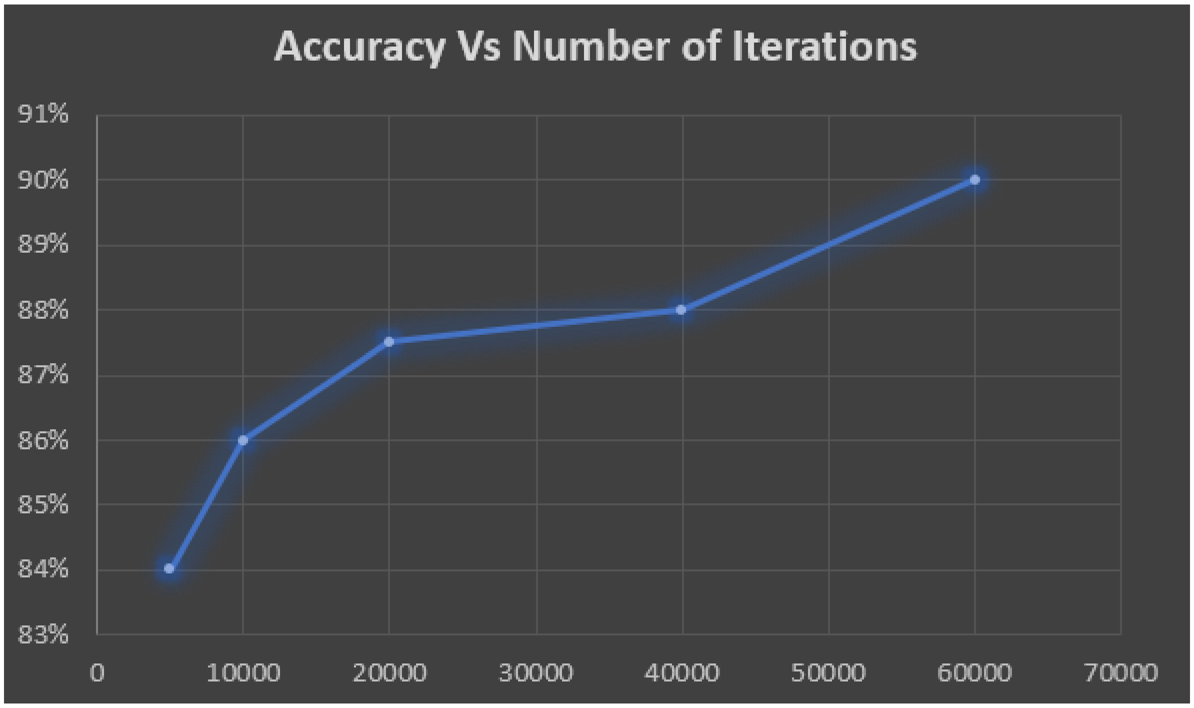
This output is then passed as an input to the dense layer which performs classification on the features extracted by the convolutional layers and down-sampled by the pooling layers. Every node in the layer is connected to every node in the preceding layer. In this layer, there is a regularization rate of 0.4 that will drop any given element during the training.

Step 6:

The logits layer simulates the final output which categorizes two classes 0 for non-speech and 1 for speech.

LEARNING CURVE, ACCURACY AND LOSS

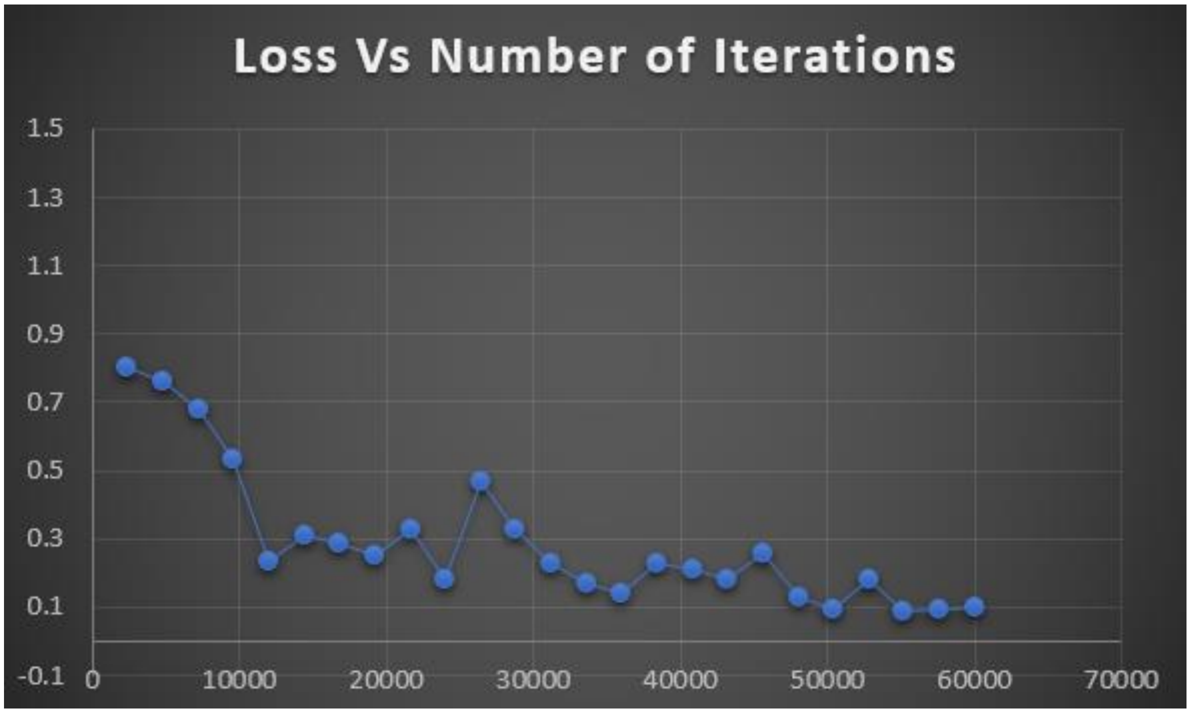
Initially, we passed our data using 5000 iterations for which the accuracy was found to be around 84%. Then we increased the number of iterations to 60,000 for which the accuracy was found to be around 90% that evaluates the quality of fit relative to the data.



From our accuracy vs number of iterations scatter plot graph, we can clearly see that the more number of iterations, the better will be the accuracy.

Learning rate:

Learning rate is a parameter that is chosen by the programmer. A high learning rate means that bigger steps are taken in the weight updates and thus it may take less time for model to converge to an optimal set of weights.



From our Loss vs number of iterations scatter plot graph, we can clearly see that the loss function and the number of iterations are inversely proportional to each other. In other words, as the number of iterations increases, there is a decrease in the loss values.

RESULTS

WHAT WE LEARNT?

We learnt the following things:

1. To detect to which target class the data belongs to by measuring the evaluation metrics with an accuracy of 90% that results in a high performance neural network.

2. To normalize the values to get accurate data that doesn't go out of range and for easy computations.

3. To decrease the learning rate during training.

4. To use an optimizer called Adam optimizer that minimizes the cost function that is more stable.