Audio classification using MLP and CNN architectures

Multimedia final project

Group members:

Mohammad Reza Babaei Mosleh

Abtin Hassan Nezhad

Soheil Abdollahi

Morteza Salimi

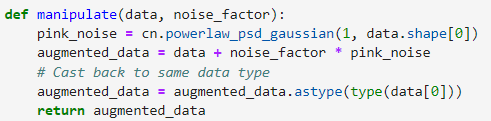
In this project we implemented deep neural networks for audio classification problem using two different methods:

1. MLP (Muli Layer Perceptron) with MFCC features
2. CNN (Convolutional Neural Networks) with spectrum features

# Dataset

Our dataset is not one of the well-known open-source datasets and is gathered by our team.

For the first step we recorded about 800 audios with different formats and converted them to .wav format. Next, we doubled our dataset using data augmentation methods with adding noise to our original data. After creating our base dataset, we did some verifications

 (Function to add noise)

# Pre-Processing

At the second step we did some pre-processing

A – cutting all audios to 3 seconds:

our audios are more than 3 second length, we clipped all of our data to 3 second as our entries should be in fixed size

B – down sampling

We down sampled our fixed size data to pick 16000 sample per second and 48000 sample over all

C – train, test, validation split:

we split our data into 70% for training, 20% for test and 10% for validation

D – creating .CSV file

then we created 2 .csv file for our data set for later uses in data loading. The first one is raw .CSV and the second is in One-Hot encoded format

# Preparing Data Loader For MLP

In this step we prepared a data loader for MLP model to load our data and labels with respect to our pre made .CSV files. to process data and we extracted the MFCC features and added them to our data loader to feed to model and visualizing our features.

 (Code for data loader)

Then we loaded our data using 8 batches for more training speed (our training works with GPU)

# Model Architecture For MLP

then we wrote our training function an specified our model Architecture in this project we trained 4 different models with different accuracy for later use (models are included in this repository). for example, we show one of the models:

in our training we used L2 regularization for preventing from overfitting also we used early stopping with patience value of 15 epoch for this purpose.

In this example model we used an architecture like below

Dense layer (2790 \* 512) followed by batch normalization, dropout (0.5) and ReLU activation function

Dense layer (512 \* 128) followed by batch normalization, dropout (0.5) and ReLU activation function

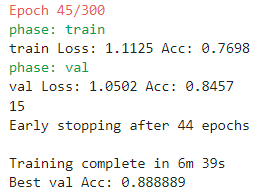
Dense layer (128 \* 64) followed by batch normalization, dropout (0.5) and ReLU activation function

Dense layer (64 \* 16) followed by batch normalization, dropout (0.5) and ReLU activation function

Dense layer (16 \* 4) followed by SoftMax of 4 neurons

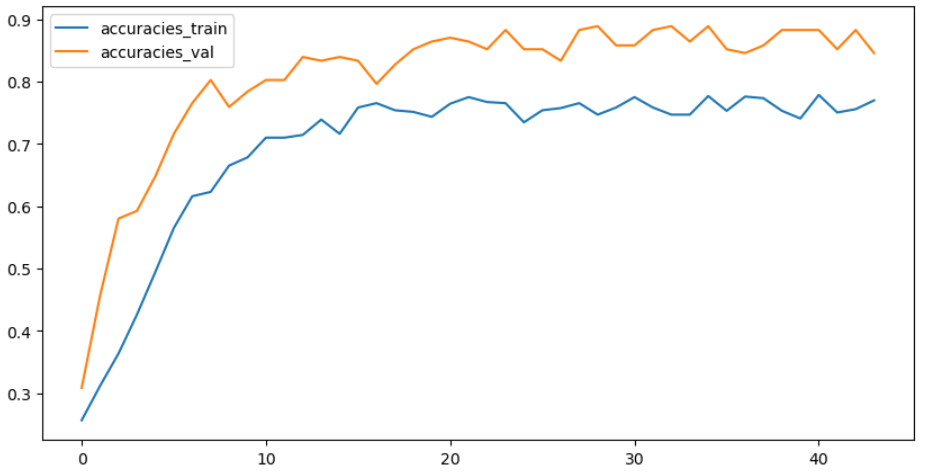
# Results For MLP

In our last epoch the training accuracy followed by validation is shown below:

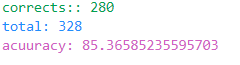


It is worth mentioning that our training stopped by early stopping after 45 epochs

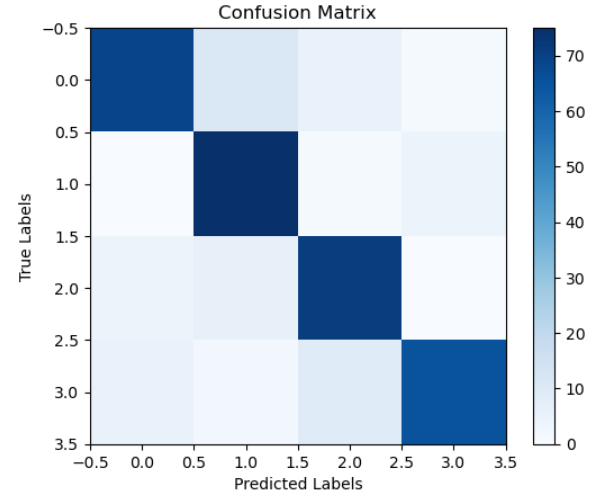
The graph of training process is like below:



After making sure that the model is not overfitted we test it on test data:



Confusion matrix:



# Preparing Data Loader For CNN

Like the previous part we prepared a data loader for process our data and add it to our data loader with proper format and by proper format we mean spectrum of audios:



This process will return us features of size (128, 94).

# Model Architecture For CNN

then we wrote our training function an specified our model Architecture in this project (model are included in this repository):

in our training we used L2 regularization for preventing from overfitting also we used early stopping with patience value of 15 epoch for this purpose.

Our layers are listed below:

Conv2d(in\_channels=1, out\_channels=32, kernel\_size=5, padding="same")

MaxPool2d(kernel\_size=2)

BatchNorm2d(num\_features=32)

ReLU()

Dropout(p=0.1)

Conv2d(in\_channels=32, out\_channels=32, kernel\_size=3, padding="same")

MaxPool2d(kernel\_size=2)

BatchNorm2d(num\_features=32)

ReLU()

Dropout(p=0.1)

Conv2d(in\_channels=32, out\_channels=64, kernel\_size=3, padding="same")

MaxPool2d(kernel\_size=2)

BatchNorm2d(num\_features=64)

ReLU()

Dropout(p=0.1)

Conv2d(in\_channels=64, out\_channels=64, kernel\_size=3, padding="same")

MaxPool2d(kernel\_size=2)

BatchNorm2d(num\_features=64)

ReLU()

Dropout(p=0.3)

Flatten()

Dense (in\_features=fc\_dim\_in, out\_features=128)

BatchNorm2d(num\_features=128)

ReLU()

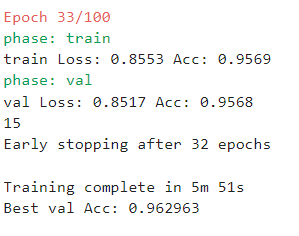
Dropout(p=0.5)

Dense (in\_features=128, out\_features=4

Softmax(dim = 1)

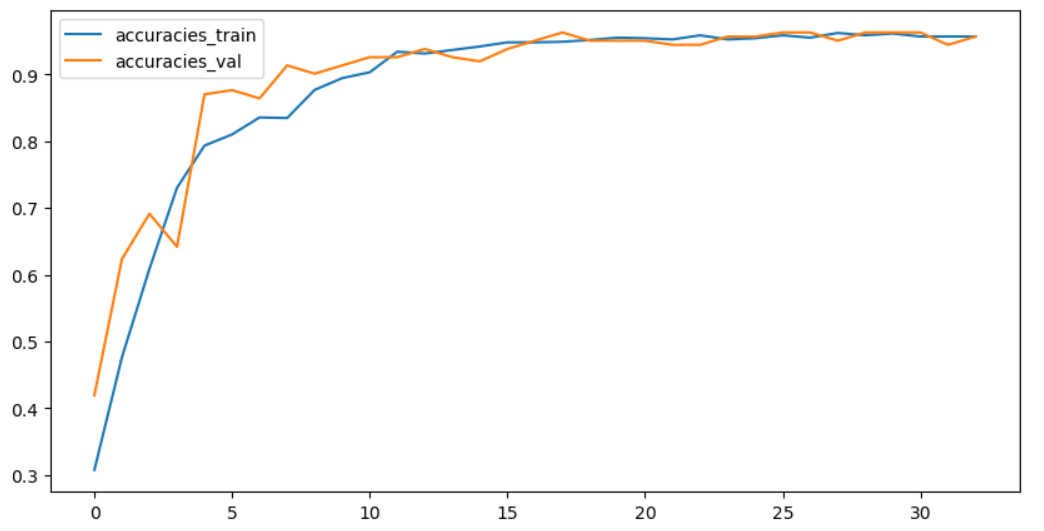
# Results

Our training accuracy after 33 epochs is:

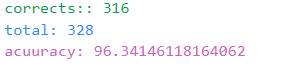


It is worth mentioning that we were planned for 300 epochs of training but we had early stopped after 15 epochs of being unimproved in validation accuracy on epoch 34

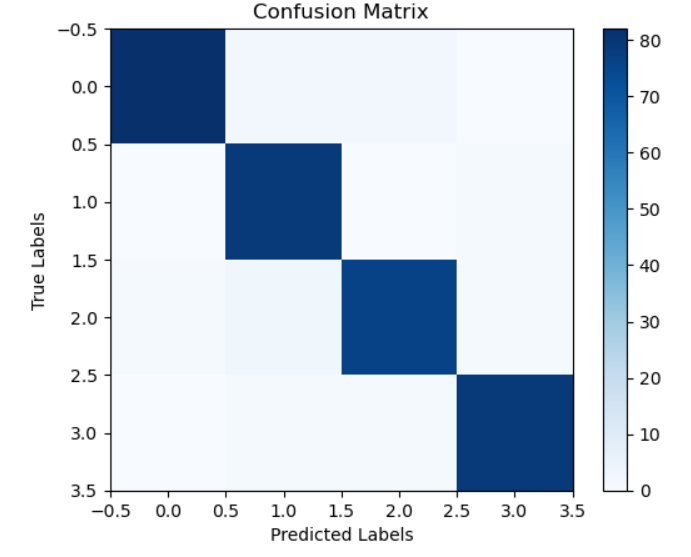
The graph of training process is:



After ensuring that model is not overfitted we tested our data of test dataset and here are the results:

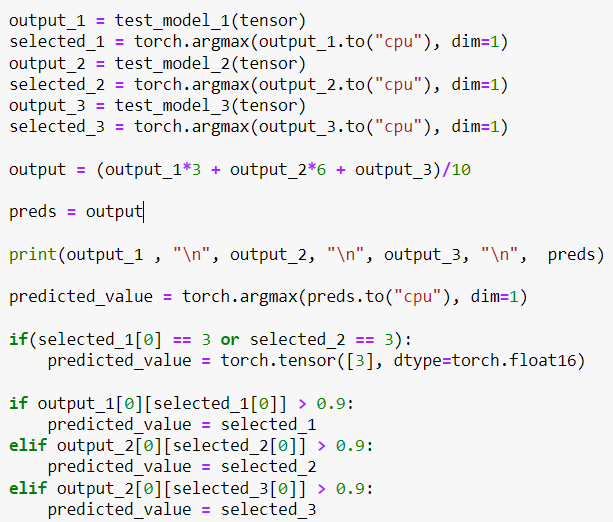


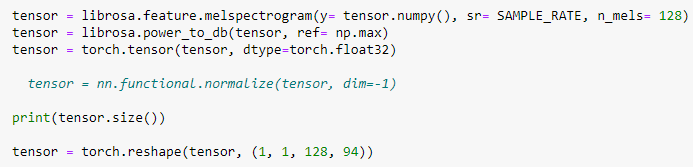
Confusion matrix is:



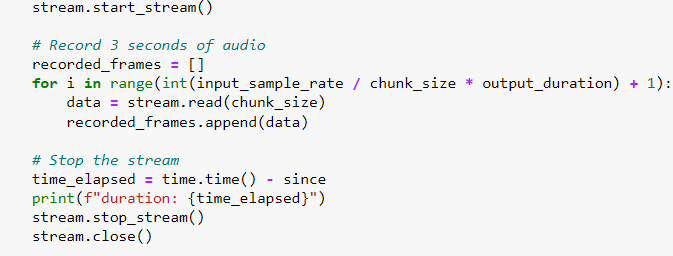
# Real-Time Processing Of MLP & CNN

Because our MLP models was not accurate enough to do the classification task individually we used **Ensemble Learning** method by combining models’ outputsconsidering different weights for every model here is how we combined our models:

(Code for ensemble learning with MLP)

(Code for processing of CNN input audio)

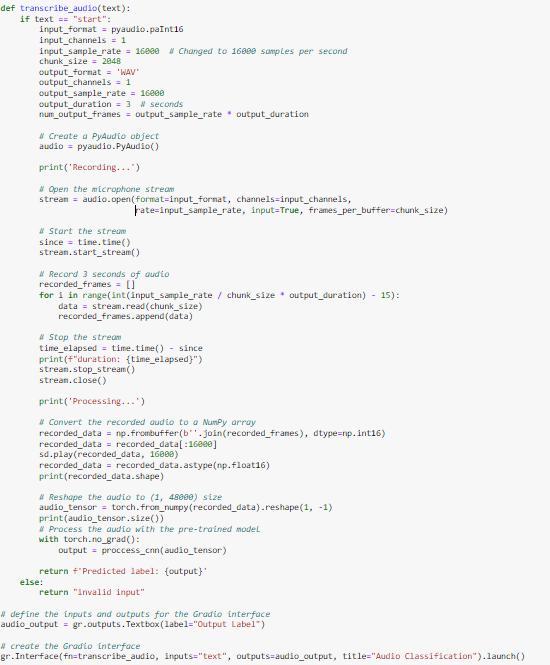
Then we used our system microphone to get audio in 1 second periods. Then we cropped audio to 1 second and padded it to 48000 samples (this reduces accuracy but also reduce the delay between 2 different commands).

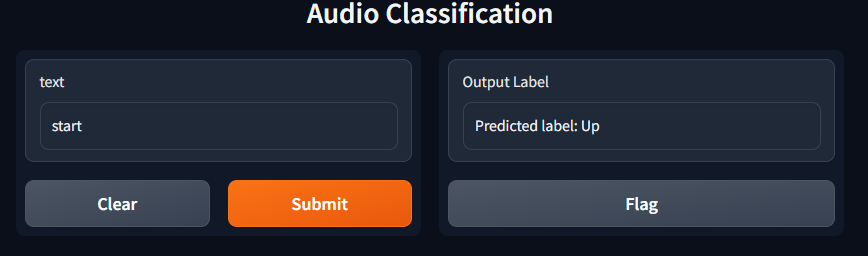


It is notable that our processor in this method has better accuracy but still not good enough for real-time processing in game so we used CNN model for our game.

# Gradio

The Gradio library is used to run code with the help of a user interface. In calling this function, we choose the input type (text, audio, image, etc.) and the output type. We set the input as text and the output as a text box. When the user enters "start" as input, the microphone is activated and ready to receive audio. After receiving and processing the audio, it displays the output label in the text box.

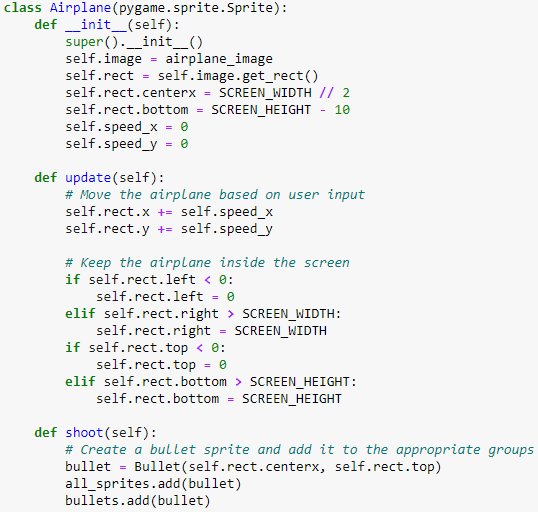
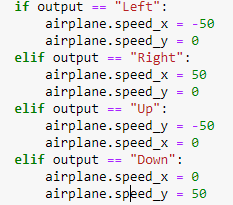




# Game

Game:  
This game is programmed using the Pygame library, and constants such as dimensions and scores have been defined. Additionally, we define classes and methods for the airplane, object(bomb), bullet, and explosion. In the airplane class, the init methods are for constructing the airplane, the update method is for its position based on speed, and the shoot method is for firing bullet s. The bomb object class also has an init method for constructing it and an update method for changing its position and explosion. The explosion and bullet classes are also similar to the airplane class.  
The spawn function adds bombs to the screen. The endgame function displays a message and results when the game ends. After creating the above, the game starts.  
As previously mentioned, sound is received and processed in real-time according to the given label, and the speed of the airplane is adjusted. The update method of the airplane is called each time, and the airplane goes to its new position.  
Furthermore, we use the explosion class for the collision of a bullet with a bomb. Every 5 seconds, a bomb is added to the screen.  
The number of lives and the sharpness of the score are displayed in the corner of the screen. When the number of lives reaches zero, we exit the game and call the endgame function 07:22 PM

Above explanations could be seen in below code:

THE END