Annapoornima S

225229101

Lab8. Audio corpus creation and binary classification using DNN

Creating dataset

In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

Read Audio

In [2]:

```
import librosa
```

In [3]:

```
age_path = r"rhyming/age"
cage_path = r"rhymin/cage"
```

In [4]:

```
import os
```

In [5]:

```
audio_data = []
```

In [6]:

```
def read_audio(path):
    audio_data = []
    for filename in os.listdir(path):
        file_path = os.path.join(path, filename)
        if os.path.isfile(file_path) and filename.endswith('.mp3'):
            audio, _ = librosa.load(file_path, sr=None)
            audio_data.append(audio)
    return audio_data
```

In [7]:

```
def extract_features(audio_data):
    stft_features = []
    for audio in audio_data:
        stft = librosa.stft(audio)
        stft_features.append(stft)
    return stft_features
```

In [8]:

```
def dataset(cage_path, age_path):
    cage_audio_data = read_audio(cage_path)
    age_audio_data = read_audio(age_path)

    cage_stft_features = extract_features(cage_audio_data)
    age_stft_features = extract_features(age_audio_data)

    max_freq_bins = max(stft.shape[0] for stft in cage_stft_features + age_stft_features
    max_time_frames = max(stft.shape[1] for stft in cage_stft_features + age_stft_featur

    def pad_or_truncate(stft):
        pad_width = ((0, max_freq_bins - stft.shape[0]), (0, max_time_frames - stft.shap
        return np.pad(stft, pad_width, mode='constant')[:, :max_time_frames]

    cage_stft_features = [pad_or_truncate(stft) for stft in cage_stft_features]
    age_stft_features = [pad_or_truncate(stft) for stft in age_stft_features]

    cage_labels = np.zeros(len(cage_stft_features), dtype=int)
    age_labels = np.ones(len(age_stft_features), dtype=int)

    X = np.array(cage_stft_features + age_stft_features)
    y = np.concatenate([cage_labels, age_labels])
    return X, y
```

In [9]:

```
X, y = dataset(cage_path, age_path)
```

In [10]:

```
print(X)
```

```
  [ 0.0000000e+00-0.00000000e+00j  2.1426619e-08+2.19027552e-08j
    5.1744919e-06+1.87791568e-06j ...  0.0000000e+00+0.00000000e+00j
    0.0000000e+00+0.00000000e+00j  0.0000000e+00+0.00000000e+00j]
  ...
  [ 0.0000000e+00+0.00000000e+00j -1.3515309e-12+4.12847751e-12j
    2.1963738e-10-1.14392162e-10j ...  0.0000000e+00+0.00000000e+00j
    0.0000000e+00+0.00000000e+00j  0.0000000e+00+0.00000000e+00j]
  [ 0.0000000e+00+0.00000000e+00j -3.8321750e-12+2.78168526e-12j
    1.5117942e-10+7.20930468e-11j ...  0.0000000e+00+0.00000000e+00j
    0.0000000e+00+0.00000000e+00j  0.0000000e+00+0.00000000e+00j]
  [ 0.0000000e+00+0.00000000e+00j -4.8684719e-12+0.00000000e+00j
    1.5599608e-11+0.00000000e+00j ...  0.0000000e+00+0.00000000e+00j
    0.0000000e+00+0.00000000e+00j  0.0000000e+00+0.00000000e+00j]]

 [[-5.4620506e-07+0.00000000e+00j -4.8377628e-06+0.00000000e+00j
    5.5268778e-05+0.00000000e+00j ...  0.0000000e+00+0.00000000e+00j
    0.0000000e+00+0.00000000e+00j  0.0000000e+00+0.00000000e+00j]
  [-4.9705937e-07-2.09103391e-07j  1.6542983e-05-9.33953288e-06j
   -5.6196630e-05+7.59871327e-05j ...  0.0000000e+00+0.00000000e+00j
    0.0000000e+00+0.00000000e+00j  0.0000000e+00+0.00000000e+00j]
```

In [11]:

```
print(y)
```

```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1
 1 1 1]
```

Split the dataset

In [12]:

```
from sklearn.model_selection import train_test_split
```

In [13]:

```
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.25,random_state=42,stra
```

In [14]:

```
X_train.shape,y_train.shape,X_test.shape,y_test.shape
```

Out[14]:

```
((30, 1025, 247), (30,), (10, 1025, 247), (10,))
```

Train a Neural Network Model

In [15]:

```
from keras.layers import Dense,InputLayer,Flatten
from keras.models import Sequential
```

In [16]:

```
model=Sequential()
model.add(InputLayer(input_shape=(X_train.shape[1], X_train.shape[2])))
model.add(Flatten())
model.add(Dense(128,activation='relu'))
model.add(Dense(64,activation='relu'))
model.add(Dense(1,activation='sigmoid'))
model.compile(optimizer='adam',loss='binary_crossentropy',metrics=['accuracy'])
```

In [17]:

```
model.fit(X_train, y_train, epochs=100, batch_size=32)
```

```
Epoch 58/100
1/1 [==============================] - 0s 274ms/step - loss: 3.1755 -
accuracy: 0.6667
Epoch 59/100
1/1 [==============================] - 0s 286ms/step - loss: 2.4713 -
accuracy: 0.6667
Epoch 60/100
1/1 [==============================] - 0s 270ms/step - loss: 1.8978 -
accuracy: 0.6667
Epoch 61/100
1/1 [==============================] - 0s 278ms/step - loss: 1.5467 -
accuracy: 0.6667
Epoch 62/100
1/1 [==============================] - 0s 267ms/step - loss: 2.2080 -
accuracy: 0.6667
Epoch 63/100
1/1 [==============================] - 0s 270ms/step - loss: 2.3310 -
accuracy: 0.6667
Epoch 64/100
1/1 [==============================] - 0s 286ms/step - loss: 1.8682 -
```

In [18]:

```
model.evaluate(X_test,y_test)
```

```
WARNING:tensorflow:You are casting an input of type complex64 to an incom
patible dtype float32.  This will discard the imaginary part and may not
be what you intended.
1/1 [==============================] - 0s 165ms/step - loss: 201.4521 - a
ccuracy: 0.0000e+00
```

Out[18]:

```
[201.45211791992188, 0.0]
```

Different Neural Network Models

In [19]:

```
def create_model(layers, nodes):
    model = Sequential()
    model.add(InputLayer(input_shape=(X_train.shape[1], X_train.shape[2])))
    model.add(Flatten())
    for _ in range(layers):
        model.add(Dense(nodes, activation='relu'))
    model.add(Dense(1, activation='sigmoid'))
    return model
```

In [20]:

```
def train_and_evaluate_model(model, X_train, y_train, X_test, y_test):
    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
    start_time = time.time()
    model.fit(X_train, y_train, epochs=50, batch_size=32, verbose=0)
    training_time = time.time() - start_time
    train_accuracy = model.evaluate(X_train, y_train, verbose=0)[1]
    test_accuracy = model.evaluate(X_test, y_test, verbose=0)[1]
    return model.count_params(), train_accuracy, test_accuracy, training_time
```

In [21]:

```
nodes_list = [8, 16, 32, 64, 128]
layers_list = [2, 3, 4]
```

In [22]:

```
import time
```

In [23]:

```
num_params_list = []
train_accuracy_list = []
test_accuracy_list = []
training_time_list = []
```

In [24]:

```
for nodes in nodes_list:
    for layers in layers_list:
        model = create_model(layers, nodes)
        num_params, train_accuracy, test_accuracy, training_time = train_and_evaluate_mc
        num_params_list.append(num_params)
        train_accuracy_list.append(train_accuracy)
        test_accuracy_list.append(test_accuracy)
        training_time_list.append(training_time)
```

```
y not be what you intended.
WARNING:tensorflow:You are casting an input of type complex64 to an in
compatible dtype float32.  This will discard the imaginary part and ma
y not be what you intended.
WARNING:tensorflow:You are casting an input of type complex64 to an in
compatible dtype float32.  This will discard the imaginary part and ma
y not be what you intended.
WARNING:tensorflow:You are casting an input of type complex64 to an in
compatible dtype float32.  This will discard the imaginary part and ma
y not be what you intended.
WARNING:tensorflow:You are casting an input of type complex64 to an in
compatible dtype float32.  This will discard the imaginary part and ma
y not be what you intended.
WARNING:tensorflow:You are casting an input of type complex64 to an in
compatible dtype float32.  This will discard the imaginary part and ma
y not be what you intended.
WARNING:tensorflow:You are casting an input of type complex64 to an in
compatible dtype float32.  This will discard the imaginary part and ma
y not be what you intended.
WARNING:tensorflow:You are casting an input of type complex64 to an in
```

In [25]:

```
print("Training Accuracies:",train_accuracy_list)
```

```
Training Accuracies: [0.6666666865348816, 0.6666666865348816, 0.666666686
5348816, 0.6666666865348816, 0.6666666865348816, 0.6666666865348816, 0.66
66666865348816, 0.6666666865348816, 0.6666666865348816, 0.666666686534881
6, 0.6666666865348816, 0.6666666865348816, 0.6666666865348816, 0.66666668
65348816, 0.6666666865348816]
```

In [26]:

```
print("Parameters list:",num_params_list)
```

```
Parameters list: [2025489, 2025561, 2025633, 4051105, 4051377, 4051649, 8
102721, 8103777, 8104833, 16207489, 16211649, 16215809, 32423169, 3243968
1, 32456193]
```

In [27]:

```
print("Training Time:",training_time_list)
```

```
Training Time: [2.748487710952759, 2.860250949859619, 2.8435146808624268,
3.5246965885162354, 3.7564148902893066, 3.7965033054351807, 5.29689049720
7642, 5.35703182220459, 5.652527093887329, 9.283875942230225, 10.03985214
2333984, 8.220107793807983, 15.40755033493042, 15.872951984405518, 18.614
256858825684]
```

In [28]:

```
print("Test Accuracies:",test_accuracy_list)
```

```
Test Accuracies: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0]
```

In [29]:

```
best_test_accuracy_index = test_accuracy_list.index(max(test_accuracy_list))
best_nodes = nodes_list[best_test_accuracy_index // len(layers_list)]
best_layers = layers_list[best_test_accuracy_index % len(layers_list)]
```

In [30]:

```
best_test_accuracy_index
```

Out[30]:

```
0
```

In [31]:

```
best_nodes
```

Out[31]:

```
8
```

In [32]:

```
best_layers
```

Out[32]:

```
2
```

In [37]:

```
accuracy_per_unit_time_list = [test_accuracy_list[i] / training_time_list[i] for i in ra
```

In [41]:

```
accuracy_per_unit_time_list
```

Out[41]:

```
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.
0]
```

In [43]:

```
def find_best_model(nodes_list, layers_list, num_params_list, train_accuracy_list, test_
    best_test_accuracy_index = test_accuracy_list.index(max(test_accuracy_list))
    best_nodes = nodes_list[best_test_accuracy_index // len(layers_list)]
    best_layers = layers_list[best_test_accuracy_index % len(layers_list)]
    return best_nodes, best_layers
best_nodes, best_layers = find_best_model(nodes_list, layers_list, num_params_list, trai
print(f"The best model has {best_layers} layers and {best_nodes} nodes.")
```

```
The best model has 2 layers and 8 nodes.
```

In [ ]: