Khaled Ahmed Mansour AI 20221320444

Speech Recognition Final Project Report

1) from google.colab import files

uploaded = files.upload()

Purpose: This code is used to upload files directly from your local machine to Google Colab.

from google.colab import files: Imports the files module from the google.colab package, which provides functions to upload and download files in Colab.

uploaded = files.upload(): Opens a file upload dialog in your browser, allowing you to select files from your local machine. The selected files are then uploaded to Colab, and uploaded is a dictionary where the keys are the file names and the values are the file data.

2) import zipfile

import os

Purpose: These lines import the necessary libraries for handling zip files and working with the file system.

import zipfile: Imports the zipfile module, which allows you to work with ZIP archives.

import os: Imports the os module, which provides functions to interact with the operating system, such as file and directory operations.

3) for file_name in uploaded.keys():

with zipfile.ZipFile(file_name, 'r') as zip_ref:

zip_ref.extractall('audio_dataset')

Purpose: This code iterates over the uploaded files (assuming there's one ZIP file) and extracts its contents into a specified directory.

for file_name in uploaded.keys():: Loops over the file names of the uploaded files. Since the uploaded dictionary contains file names as keys, this loop iterates through those keys.

with zipfile.ZipFile(file_name, 'r') as zip_ref:: Opens the uploaded ZIP file in read mode. The zipfile.ZipFile function creates a ZipFile object, and with ensures that the file is properly closed after extraction.

zip_ref.extractall('audio_dataset'): Extracts all the contents of the ZIP file into a directory named 'audio_dataset'. If the directory doesn't exist, it will be created automatically.

Summary

- **File Upload:** The first part of the code allows you to upload a ZIP file from your local machine to Google Colab.
- **Unzipping:** The second part unzips the uploaded file and extracts its contents into a directory named audio_dataset in the Colab environment.

4)import os

Purpose: The os module provides a way of using operating system-dependent functionality. It is commonly used for interacting with the file system (e.g., listing files in a directory, creating directories, etc.).
Usage: You might use os to navigate directories, handle file paths, or manage files and folders.

5)import numpy as np

Purpose: numpy is a powerful numerical computing library in Python. It is used for working with arrays, matrices, and performing a wide variety of mathematical operations on these data structures.
Usage: numpy is often used to handle and manipulate large datasets efficiently. It provides support for arrays and matrices, along with a large library of mathematical functions to operate on these arrays.

6)import librosa
import librosa.display

Purpose: librosa is a Python package for music and audio analysis. It provides the ability to load, process, and extract features from audio files.
Usage: librosa is used to load audio files, extract features (like Mel-frequency cepstral coefficients, or MFCCs), and perform various audio transformations. The librosa.display module is used for visualizing audio data.

7)import matplotlib.pyplot as plt

Purpose: matplotlib.pyplot is a state-based interface to matplotlib, a comprehensive library for creating static, animated, and interactive visualizations in Python.
Usage: plt is used to create and display plots, charts, and other visualizations. You might use it to visualize audio features, model performance, or other data.

8) import seaborn as sns

Purpose: seaborn is a data visualization library based on matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics.
Usage: seaborn is often used to create more sophisticated and visually appealing plots than matplotlib alone, especially for statistical data visualizations.

9) from sklearn.model_selection import train_test_split

Purpose: train_test_split is a utility function that splits datasets into random train and test subsets.
Usage: It is commonly used to separate your data into training and testing sets for model evaluation.

10) from sklearn.preprocessing import LabelEncoder

Purpose: LabelEncoder encodes categorical labels with values between 0 and the number of distinct classes minus 1.
Usage: LabelEncoder is used to convert categorical labels (e.g., 'cat', 'dog') into numerical form before feeding them into a machine learning model.

11) from tensorflow.keras.utils import to_categorical

Purpose: to_categorical converts a class vector (integers) to binary class matrix. This is particularly useful for categorical cross-entropy loss in neural networks.
Usage: It is used to convert the encoded labels (from LabelEncoder) into a one-hot encoded format, which is often required by neural networks for classification tasks.

12) from tensorflow.keras.models import Sequential

Purpose: Sequential is a linear stack of layers in Keras (part of TensorFlow). It's a convenient way to build a model layer-by-layer.
Usage: You use Sequential to define the architecture of your neural network model in a straightforward, sequential manner.

13) from tensorflow.keras.layers import Dense, Dropout

Purpose:

- **Dense**: Fully connected layers where each input node is connected to each output node. It is the standard layer in a neural network.
- **Dropout**: A regularization technique that randomly sets a fraction of input units to 0 at each update during training time to prevent overfitting.

Usage: Dense layers are used to create fully connected layers in the network, while Dropout layers are used to prevent overfitting by randomly deactivating certain neurons during training.

14)from sklearn.metrics import confusion_matrix, classification_report

Purpose:

- **confusion_matrix**: Computes a confusion matrix to evaluate the accuracy of a classification.
- **classification_report**: Generates a report showing the main classification metrics (precision, recall, f1-score, and support).

Usage: These are used to evaluate the performance of your classification model, providing insights into how well the model performs across different classes.

15) audio_dir = 'audio_dataset/' # Path where your ZIP file is extracted

Purpose: This line sets the variable audio_dir to the path where your audio dataset is located. After unzipping the ZIP file in a previous step, the extracted files are placed in the 'audio_dataset/' directory.

Usage: This path will be used to access the audio files in your dataset.

16) def visualize_audio(file_name):
audio, sample_rate = librosa.load(file_name)
plt.figure(figsize=(14, 5))

Purpose: This line defines a function called visualize_audio that takes a single argument file_name (the path to the audio file) and visualizes its waveform and spectrogram.

librosa.load(file_name): Loads the audio file specified by file_name. It returns two values:

- audio: A NumPy array containing the audio time series.
- sample_rate: The sampling rate of the audio file (number of samples per second).

plt.figure(figsize=(14, 5)): Initializes a new figure with a specified size (14 inches wide by 5 inches tall) for the visualizations.

17) # Plot Waveform
plt.subplot(1, 2, 1)
librosa.display.waveshow(audio, sr=sample_rate)
plt.title("Waveform")

Purpose: This block plots the waveform of the audio file.

plt.subplot(1, 2, 1): Creates a subplot (1 row, 2 columns, 1st plot) in the figure for displaying the waveform.

librosa.display.waveshow(audio, sr=sample_rate): Plots the waveform of the audio signal using the sampling rate to scale the time axis.

plt.title("Waveform"): Sets the title of this subplot to "Waveform".

18) # Plot Spectrogram
plt.subplot(1, 2, 2)
spectrogram = librosa.feature.melspectrogram(y=audio, sr=sample_rate)
librosa.display.specshow(librosa.power_to_db(spectrogram, ref=np.max), sr=sample_rate)
plt.title("Spectrogram")

Purpose: This block plots the spectrogram of the audio file.
plt.subplot(1, 2, 2): Creates a subplot (1 row, 2 columns, 2nd plot) in the figure for displaying the spectrogram.
librosa.feature.melspectrogram(y=audio, sr=sample_rate): Computes the Mel-scaled spectrogram of the audio signal. A spectrogram represents the intensity of frequencies over time.
librosa.display.specshow(librosa.power_to_db(spectrogram, ref=np.max), sr=sample_rate): Displays the spectrogram in decibels using the power of the spectrogram, with ref=np.max normalizing the spectrogram to the maximum value.
plt.title("Spectrogram"): Sets the title of this subplot to "Spectrogram".

19) plt.show()

- **Purpose:** Displays the figure containing the waveform and spectrogram subplots.

20) visualize_audio('audio_dataset/cats_dogs/cat_1.wav')

Purpose: This line calls the visualize_audio function with a specific audio file ('audio_dataset/cats_dogs/cat_1.wav') to visualize its waveform and spectrogram.
Usage: This example assumes that the file 'cat_1.wav' exists in the directory 'audio_dataset/cats_dogs/'. When run, it will display the waveform and spectrogram of this audio file.

Summary
This code is designed to load an audio file, visualize its waveform (the signal in the time domain), and display its spectrogram (a representation of the signal in the frequency domain over time). The visualize_audio function allows you to analyze the structure of audio data, which is useful for understanding how the sound varies over time and its frequency content. The example usage provided demonstrates how to apply this visualization function to a specific file in your dataset.

21) def extract_features(file_name):
    audio, sample_rate = librosa.load(file_name, res_type='kaiser_fast')
    mfccs = librosa.feature.mfcc(y=audio, sr=sample_rate, n_mfcc=40)
    return np.mean(mfccs.T, axis=0)

Purpose: This function loads an audio file, extracts the Mel-frequency cepstral coefficients (MFCCs) from it, and returns the mean of these coefficients as a feature vector.

librosa.load(file_name, res_type='kaiser_fast'): Loads the audio file using the kaiser_fast resampling method, which is faster while maintaining good quality.

librosa.feature.mfcc(y=audio, sr=sample_rate, n_mfcc=40): Extracts 40 MFCCs from the audio signal. MFCCs are a common feature used in audio processing and machine learning tasks.

np.mean(mfccs.T, axis=0): Computes the mean of the MFCCs across time (the transpose of MFCCs ensures that we average over the time dimension).

22) features, labels = [], []

Purpose: These lists will store the extracted features and corresponding labels for each audio file processed.

23) print(os.listdir(audio_dir)[0])
# Get the first item in the directory (which is a folder)
first_item = os.listdir(audio_dir)[0]
first_item_path = os.path.join(audio_dir, first_item)

Purpose: This code snippet retrieves the first item in the audio_dir directory (which is assumed to be a folder containing the audio files) and constructs the path to this folder.

os.listdir(audio_dir)[0]: Lists the contents of audio_dir and selects the first item.

os.path.join(audio_dir, first_item): Combines audio_dir and first_item to create a full path to the first folder.

```
24) if os.path.isdir(first_item_path):
    # List the files inside this folder
    for file_name in os.listdir(first_item_path):
        if file_name.endswith('.wav'):
            print("Processing:", file_name)
            file_path = os.path.join(first_item_path, file_name)
            label = 'cat' if 'cat' in file_name.lower() else 'dog'
            features.append(extract_features(file_path))
            labels.append(label)
```

Purpose: This block checks if the first_item is indeed a directory and then processes each .wav file within it.

os.path.isdir(first_item_path): Checks if first_item_path is a directory.

os.listdir(first_item_path): Lists all files in the directory.

file_name.endswith('.wav'): Filters to only process files with a .wav extension.

file_path = os.path.join(first_item_path, file_name): Constructs the full path to each .wav file.

label = 'cat' if 'cat' in file_name.lower() else 'dog': Determines the label (cat or dog) based on the filename.

features.append(extract_features(file_path)): Extracts features from the audio file and appends them to the features list.

labels.append(label): Appends the corresponding label to the labels list.

25) X = np.array(features)
y = np.array(labels)

Purpose: Converts the lists of features and labels into NumPy arrays, which are easier to work with in machine learning pipelines.

26) le = LabelEncoder()
y = to_categorical(le.fit_transform(y))

Purpose: Encodes the string labels ('cat', 'dog') into numerical values (e.g., 0 and 1) and then converts these into a one-hot encoded format.
le.fit_transform(y): Fits the LabelEncoder on y and transforms the labels into numerical format.
to_categorical(): Converts the numerical labels into a one-hot encoded format, which is a standard input format for neural networks in classification tasks.

27) X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

- **Purpose:** Splits the dataset into training and testing sets to evaluate the model's performance.
- **train_test_split(X, y, test_size=0.2, random_state=42):** Splits X and y into training and testing sets, with 20% of the data reserved for testing. The random_state=42 ensures reproducibility of the split.

Summary
This code processes an audio dataset by extracting MFCC features from each .wav file, assigns a label based on the filename (cat or dog), encodes the labels, and splits the data into training and testing sets. The extracted features will be used as input for a machine learning model, and the encoded labels serve as the target variable for classification.

28) model = Sequential()

Purpose: This line initializes a sequential model in Keras, which means that layers will be added to the model one after another in a linear stack.

29) model.add(Dense(256, input_shape=(X_train.shape[1],), activation='relu'))

Purpose: Adds the first dense (fully connected) layer with 256 units (neurons).
input_shape=(X_train.shape[1],): Specifies the shape of the input data for this layer. X_train.shape[1] corresponds to the number of features in the input data.
activation='relu': Uses the ReLU (Rectified Linear Unit) activation function, which is commonly used in neural networks for introducing non-linearity.

30) model.add(Dropout(0.5))
Purpose: Adds a dropout layer to the model with a dropout rate of 50% (0.5). This layer randomly sets 50% of the inputs to zero during training to prevent overfitting by reducing the reliance of the model on any specific neurons.

31) model.add(Dense(128, activation='relu'))

Purpose: Adds another dense layer with 128 units and ReLU activation function. This layer continues to learn more complex patterns from the previous layer's output.

32) model.add(Dropout(0.5))

Purpose: Adds another dropout layer with a dropout rate of 50%. This further helps in regularizing the model and preventing overfitting.

33) model.add(Dense(64, activation='relu'))

Purpose: Adds a third dense layer with 64 units and ReLU activation function. This layer is designed to learn more complex features in a progressively smaller feature space.

34) model.add(Dropout(0.5))

Purpose: Adds a third dropout layer with a dropout rate of 50%. Consistent dropout layers after each dense layer help ensure that the model generalizes well.

35)model.add(Dense(y_train.shape[1], activation='softmax'))

Purpose: Adds the output layer with units equal to the number of classes (as given by y_train.shape[1]), which is 2 in this case (cat and dog).

activation='softmax': The softmax activation function is used to convert the output of the network into probability distributions over the classes.

36) model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

Purpose: Configures the model for training.
loss='categorical_crossentropy': Specifies the loss function to use, which is categorical crossentropy for multi-class classification problems.
optimizer='adam': Specifies the optimizer to use, which is Adam, a popular optimization algorithm that adapts the learning rate during training.
metrics=['accuracy']: Specifies the metric to evaluate the model, which is accuracy in this case.

37) history = model.fit(X_train, y_train, epochs=50, batch_size=32, validation_data=(X_test, y_test), verbose=1)

Purpose: Trains the model on the training data.
X_train, y_train: The input data and corresponding labels for training.
epochs=50: Specifies the number of times the model will iterate over the entire training dataset.
batch_size=32: Specifies the number of samples that will be propagated through the network at once during training.
validation_data=(X_test, y_test): Specifies the validation data that will be used to evaluate the model after each epoch. This helps in monitoring the model's performance on unseen data during training.
verbose=1: Prints the progress of training to the console.

Summary
This code builds a multi-layer perceptron (MLP) model with three hidden layers. Each layer has a certain number of neurons (256, 128, 64) with ReLU activation and dropout for regularization. The output layer uses softmax activation to perform classification between the classes (cats and dogs). The model is compiled using the Adam optimizer and trained using the categorical cross-entropy loss function, with accuracy as the evaluation metric. The model is trained for 50 epochs with a batch size of 32, and validation is done on a separate test set.

38) score = model.evaluate(X_test, y_test, verbose=0)
print(f'Test accuracy: {score[1]*100:.2f}%')

Purpose: This code evaluates the trained model on the test data.
model.evaluate(X_test, y_test, verbose=0): Computes the loss and accuracy of the model on the test set (X_test, y_test). The verbose=0 argument suppresses the output.
score[1]*100: Retrieves the accuracy from the evaluation result (score[1] corresponds to accuracy) and multiplies it by 100 to convert it to a percentage.
print(f'Test accuracy: {score[1]*100:.2f}%'): Prints the test accuracy as a percentage, formatted to two decimal places.

39) y_pred = model.predict(X_test)
y_pred_classes = np.argmax(y_pred, axis=1)
y_true = np.argmax(y_test, axis=1)

Purpose: This code generates predictions on the test set and converts them into class labels.

y_pred = model.predict(X_test): Uses the model to predict the class probabilities for each instance in the test set.

y_pred_classes = np.argmax(y_pred, axis=1): Converts the predicted probabilities to class labels by selecting the class with the highest probability for each instance. np.argmax returns the indices of the maximum values along the specified axis (axis=1 means across rows).

y_true = np.argmax(y_test, axis=1): Converts the one-hot encoded true labels (y_test) back into class labels by selecting the index of the maximum value (which corresponds to the class label).

40) cm = confusion_matrix(y_true, y_pred_classes)

Purpose: This generates a confusion matrix to visualize the performance of the classification model.

confusion_matrix(y_true, y_pred_classes): Creates a confusion matrix, which shows the number of correct and incorrect predictions for each class. Rows represent the true classes, and columns represent the predicted classes.

```
41)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
xticklabels=le.classes_, yticklabels=le.classes_)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

Purpose: This code visualizes the confusion matrix using a heatmap.

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=le.classes_, yticklabels=le.classes_): Plots the confusion matrix cm as a heatmap.

- annot=True: Annotates each cell with the number of instances.
- fmt='d': Formats the annotations as integers (decimal format).
- cmap='Blues': Specifies the color map for the heatmap.
- xticklabels=le.classes_, yticklabels=le.classes_: Sets the class labels for the x and y axes.

plt.xlabel('Predicted'), plt.ylabel('Actual'): Labels the x and y axes as 'Predicted' and 'Actual', respectively.

plt.title('Confusion Matrix'): Adds a title to the plot.

plt.show(): Displays the plot.

42) print(classification_report(y_true, y_pred_classes, target_names=le.classes_))

Purpose: This code generates a detailed classification report that includes precision, recall, F1-score, and support for each class.

classification_report(y_true, y_pred_classes, target_names=le.classes_):

- y_true: The true class labels.
- y_pred_classes: The predicted class labels.
- target_names=le.classes_: Maps the class labels to their string names ('cat' and 'dog').

Summary

- **Evaluation:** The model is evaluated on the test set, and the accuracy is printed.
- **Prediction:** Predictions are made on the test data, and these are compared with the actual labels.
- **Confusion Matrix:** The performance of the model is visualized using a confusion matrix heatmap, which shows how often the model correctly and incorrectly classified each class.
- **Classification Report:** A detailed classification report is printed, which provides precision, recall, and F1-score for each class, giving a deeper understanding of the model's performance.