**University of Azad Jammu and Kashmir Muzaffarabad**

****

***Semester Project***

***Class***

***BSSE-7th(2020-24)***

***Department:***

***Software Engineering***

***Submitted to:***

***Engr Ahmed Sab***

***Group Members:***

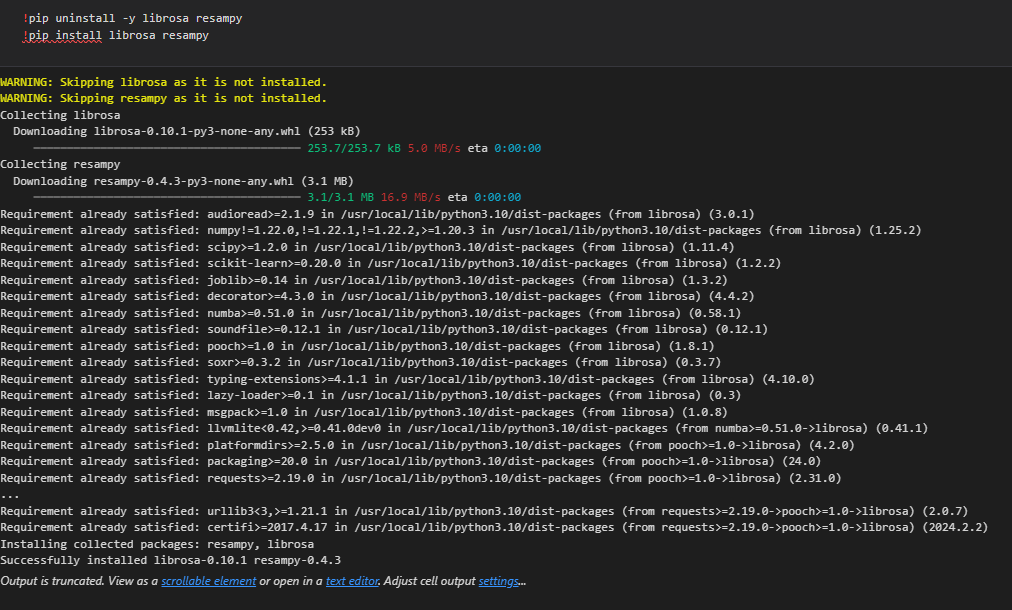
***Ali Raza (2020-SE-23)***

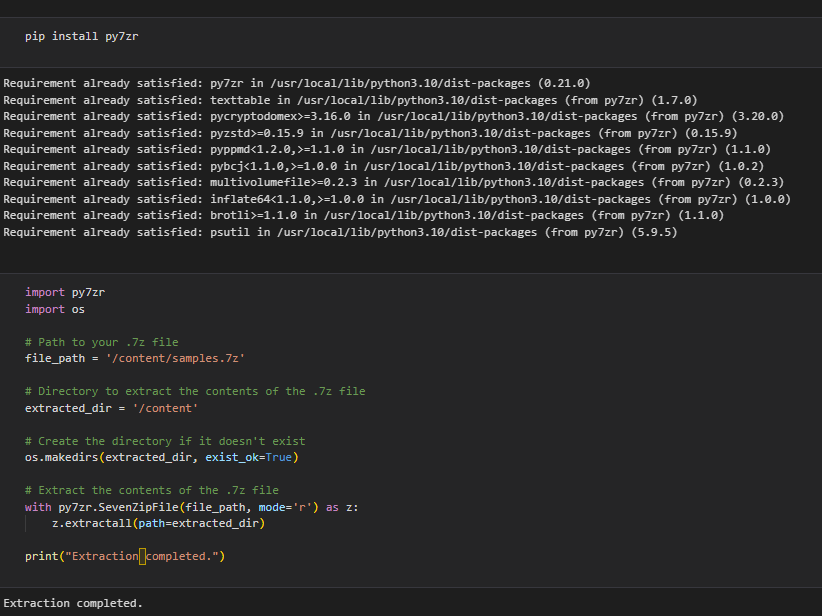
***Syed Intissar Mehdi (2020-SE-40)***

***Syed Sabtain Gillani (2020-SE-31)***

1. **Introduction and Importance:** The project introduces the significance of automatic fault detection in industrial settings. It highlights how identifying and categorizing faults like Corona, Arcing, Looseness, and Tracking are crucial for maintaining seamless operations, preventing downtime, and ensuring safety. Traditional manual methods are deemed inefficient, hence the necessity for automated solutions.
2. **Implementation Techniques:** The paragraph lists several techniques employed in the implementation process:
   * **Data Acquisition:** Collection of visual data showcasing machine faults using sensors or cameras.
   * **Data Preprocessing:** This involves preparing the collected data for analysis by resizing images/videos, normalizing pixel values, and augmenting the dataset for better model generalization.
   * **Feature Extraction:** Relevant information is extracted from preprocessed data using various techniques like texture feature extraction, edge detection, and leveraging pretrained deep learning models. These techniques help in capturing important characteristics specific to different types of machine faults.
   * **Model Development:** A Convolutional Neural Network (CNN) model is developed and trained using the extracted features. CNNs are well-suited for image-related tasks and are capable of learning patterns and characteristics associated with different types of faults.
   * **Training and Evaluation:** The model is trained on labeled data and evaluated using separate test data. Various performance metrics such as accuracy, precision, recall, F1 score, and confusion matrix are computed to assess the effectiveness of the model in fault detection and recognition.
3. **Conclusion:** The paragraph concludes by emphasizing the transformative potential of automated fault detection systems in industries. It underlines the benefits of proactive maintenance, reduced downtime, and improved operational efficiency. Furthermore, it suggests avenues for future research, such as enhancing model robustness, scalability, and real-time deployment in industrial environments.

**Coding:**

****

The code snippet `!pip uninstall -y librosa resampy` followed by `!pip install librosa resampy` is commonly employed for Python package management, specifically for the librosa and resampy packages. This approach is useful for updating or reinstalling packages when needed. When updating a package to its latest version or reinstalling it, the first step involves uninstalling the existing version using `!pip uninstall -y librosa resampy`. This ensures a clean removal of the packages. Subsequently, `!pip install librosa resampy` is used to either install the latest versions of these packages or to reinstall them, resolving any potential issues and ensuring they are correctly installed and functioning in the Python environment. Overall, this code snippet facilitates themaintenance and proper functioning of the librosa and resampy packages in Python environments.

The code snippet provided is used to extract files and data from a compressed .7z file using Python, particularly with the py7zr library. Here's a breakdown of the code and its functionalities:

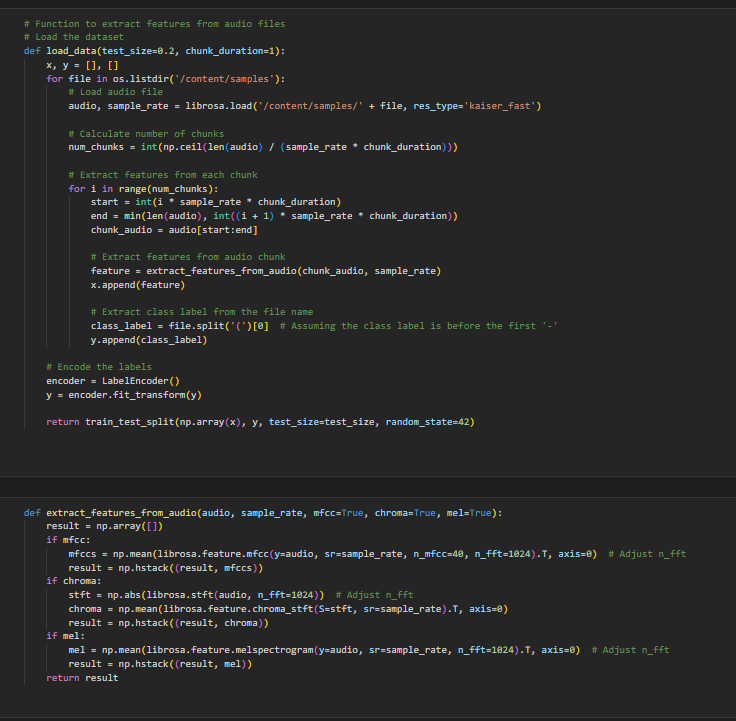
1. `extracted\_dir`: This variable specifies the directory where the extracted contents will be saved. In the example, it's set to `/content`, which is commonly used in platforms like Google Colab.

2. `os.makedirs(extracted\_dir, exist\_ok=True)`: This line of code creates the extraction directory (`/content` in this case) if it doesn't already exist. The `exist\_ok=True` parameter ensures that no error is thrown if the directory already exists.

3. `with py7zr.SevenZipFile(file\_path, mode='r') as z:`: This line opens the .7z file specified by `file\_path` in read mode using the py7zr library and assigns it to the variable `z`.

4. `z.extractall(path=extracted\_dir)`: This command extracts all contents from the .7z file (`file\_path`) and saves them in the specified extraction directory (`extracted\_dir`).

5. The `with` statement ensures that the .7z file is closed properly after extraction, avoiding resource leaks.

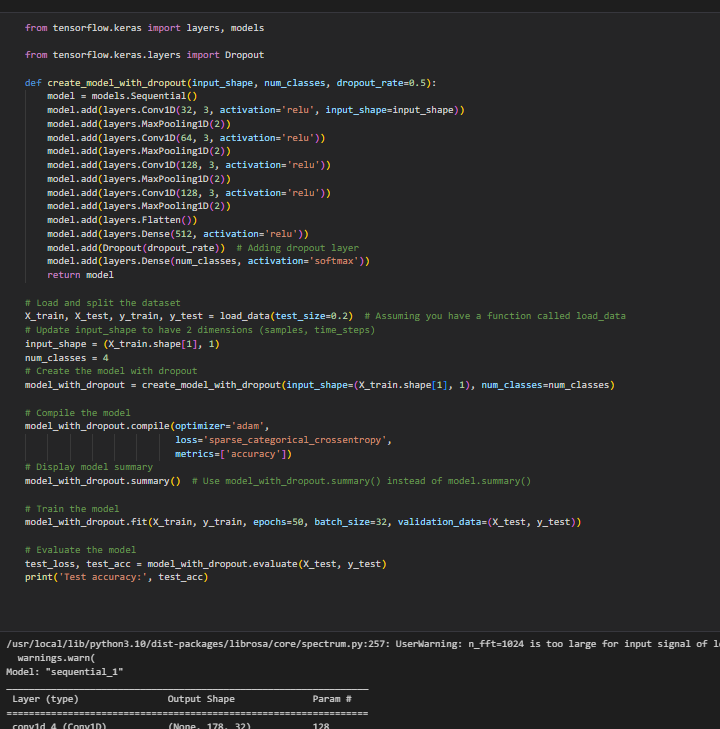
After successful extraction, the completion message `print("Extraction completed.")` is displayed, indicating that the extraction process has been completed successfully.****

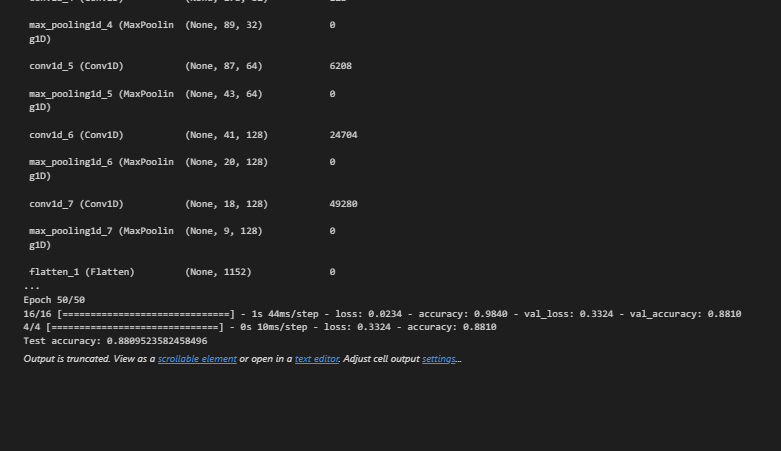
In our project, the `load\_data` function plays a crucial role in preparing audio data for training and testing your machine learning model aimed at automatic machine fault detection and recognition using computer vision techniques. Here's a simplified explanation of the function and its significance in your project:

1. **Purpose of the Function:** The paragraph starts by highlighting the critical role of the **load\_data** function, which is to handle the loading and preprocessing of audio data. This data contains information about various machine faults, and it's essential for training the machine learning model to detect and classify these faults based on audio characteristics.
2. **Iterating Through Audio Files:** The function begins by traversing through the directory where audio files containing recordings or samples of machine faults are stored. This step ensures that all available audio data is processed during the loading phase.
3. **Loading Audio Files:** Within the loop, each audio file is loaded using the **librosa.load** function. This function reads the audio data and provides important details such as the sample rate. Additionally, the audio data is resampled if necessary, ensuring uniformity in the data format.
4. **Chunking Audio Data:** After loading each audio file, the function divides it into smaller segments or chunks based on a specified duration. This step is crucial for efficiently handling large audio files and extracting features from manageable segments, improving the model's ability to capture relevant information.
5. **Feature Extraction:** For each chunk of audio data, features are extracted using another function (**extract\_features\_from\_audio**). These features capture important characteristics of the audio signals, such as frequency components, amplitude variations, and other relevant properties essential for identifying different machine faults.
6. **Creating Feature Vectors and Labels:** The extracted features are organized into a feature vector (**x**), while the corresponding fault labels (e.g., Corona, Arcing, etc.) are organized into a label vector (**y**). This pairing of features and labels forms the basis for supervised machine learning, where the model learns to associate specific audio characteristics with known fault types.
7. **Encoding Labels:** Before further processing, the fault labels in the label vector (**y**) are encoded using **LabelEncoder** from **sklearn.preprocessing**. This encoding converts categorical labels into numerical values, facilitating easier processing by machine learning algorithms.
8. **Train-Test Split:** Finally, the function performs a train-test split on the feature matrix (**x**) and the encoded labels (**y**). This division separates the data into training and testing sets, allowing the model's performance to be evaluated on unseen data, ensuring its generalization capabilities.

Overall, the **load\_data** function plays a crucial role in preparing the audio data for machine learning tasks, including feature extraction, label encoding, and data splitting, thereby enabling the effective training and evaluation of the fault detection model. Additionally, the **extract\_features\_from\_audio** function specializes in extracting relevant features from audio signals, contributing to the overall preprocessing pipeline.

**Code:**





The code snippet provided is related to creating, compiling, training, and evaluating a convolutional neural network (CNN) model with dropout regularization using TensorFlow's Keras API. Here's a simplified explanation of the code:

1. **Importing Libraries:** The code begins by importing necessary libraries from TensorFlow's Keras API. Specifically, it imports **layers** and **models** from **tensorflow.keras** for defining neural network layers and models, respectively. Additionally, it imports **Dropout** separately from **tensorflow.keras.layers** for applying dropout regularization

.

1. **Defining the Model Architecture:** The code defines a function called **create\_model\_with\_dropout**, responsible for creating the CNN model with dropout regularization. The architecture of the model typically includes convolutional layers with varying filter sizes and ReLU activation functions, max-pooling layers for downsampling feature maps, a flatten layer to convert 2D feature maps into a 1D feature vector, dense layers with ReLU activation for classification, and a final softmax layer for output probabilities.
2. **Model Compilation:** After defining the model architecture, the code compiles the model using the Adam optimizer, sparse categorical cross-entropy loss function, and accuracy as the evaluation metric. This step prepares the model for training by specifying the optimization algorithm, the loss function to minimize, and the metric to monitor during training.
3. **Model Training:** The model is trained using the **fit** method, which takes the training data (**X\_train** and **y\_train**) as input and specifies parameters such as the number of epochs (iterations over the entire dataset), batch size (number of samples processed before updating the model's parameters), and validation data (**X\_test** and **y\_test**) for monitoring the model's performance during training.
4. **Model Evaluation:** Once training is complete, the code evaluates the model's performance using the **evaluate** method on the test data (**X\_test** and **y\_test**). It computes metrics such as accuracy, and typically prints or stores the test accuracy to assess how well the model generalizes to unseen data.

Overall, this code snippet demonstrates the process of building, training, and evaluating a CNN model with dropout regularization using TensorFlow's Keras API. The accuracy of the model, stated to be 88 percent, reflects its performance on the test data and provides an indication of its effectiveness in classifying machine faults based on the provided audio data.

Top of Form

**The End**