EmotiSense: A Multi-Modal Emotion Detection Framework

Technical Report

EmotiSense: A Multi-Modal Emotion Detection Framework

Abstract

In the realm of human-computer interaction, accurately discerning and understanding human emotions plays a pivotal role. However, conventional emotion detection systems have predominantly relied on single modes of data analysis, such as text or facial recognition, resulting in limited accuracy and depth. The intricacy of human emotions, expressed through facial expressions, tone of voice, and language nuances, poses a significant challenge to these unimodal systems, often leading to misinterpretations and context loss. As digital interactions become increasingly prevalent, the necessity for advanced emotion detection systems that reflect human empathy and comprehension grows. This presents a compelling opportunity to explore a more comprehensive, multimodal approach to emotion detection, capitalizing on advancements in large language models (LLMs) and machine learning technologies.

This technical report delves into the paradigm shift towards multimodal emotion detection systems, driven by the need to capture the complexity of human emotions in digital interactions authentically. By integrating insights from facial expressions, vocal tones, and textual cues, our proposed approach aims to surpass the limitations of traditional unimodal systems. Leveraging advancements in LLMs and machine learning, our system seeks to enhance accuracy and contextual understanding, thereby facilitating more empathetic and nuanced human-computer interactions. The report offers insights into theoretical foundations, methodology, and implications, demonstrating the transformative potential of multimodal emotion detection systems in revolutionizing human-computer interaction paradigms.

1. Introduction

In the ever-evolving landscape of human-computer interaction, the accurate detection and interpretation of human emotions stand as paramount objectives. Traditional emotion detection systems, often limited by their reliance on single modalities, such as text or facial recognition, have faced notable challenges in capturing the nuanced complexities of human emotional expression. Recognizing this limitation, this project endeavors to pioneer a breakthrough in emotion detection by proposing a multimodal approach that integrates data from three primary sources: image, audio, and text. By harnessing the strengths of open-source Large Language Models (LLMs) like Bert, LLaVa, and Wav2vec, the aim is to construct a comprehensive model capable of discerning and interpreting human emotions with unprecedented accuracy and depth.

The integration of these modalities marks a significant departure from traditional unimodal systems, offering a holistic perspective on emotional expression that mirrors human cognition and perception. Through image processing, audio analysis, and textual interpretation, this project seeks to delve into the subtleties and intricacies of emotional states, enabling the system to perceive and respond to user emotions with a level of sophistication previously unseen. Beyond technological advancement, this endeavor holds the promise of fostering genuine human empathy in digital interactions, thereby unlocking new avenues in user experience, mental health support, customer service, and beyond. As we embark on this journey, the convergence of technology and empathy promises to redefine the landscape of human-computer interaction in profound and meaningful ways.

2. Methods

Data Preparation: The foundation of our methodology lies in meticulous data preparation, which involves collecting, cleaning, and structuring data from diverse sources. Through rigorous preprocessing techniques, we ensure the integrity and consistency of our datasets. This encompasses noise reduction in audio data, normalization of images, and tokenization of textual content. Preparing the data sets the stage for subsequent analysis and model development.

Pretraining and Feature Extraction: Our methodology emphasizes the utilization of pre-trained models such as Bert for textual analysis, Wav2vec for audio processing, and LLaVa for image analysis. These models serve as robust starting points, enabling us to extract meaningful features from each modality. Leveraging techniques like transfer learning, we fine-tune these models with our specific datasets, enhancing their ability to capture emotional cues effectively.

Model Training and Validation: The heart of our methodology lies in model training and validation, where machine learning algorithms are employed to train our emotion detection model. Through supervised learning techniques and labeled datasets, we iteratively refine the model's parameters to optimize performance. Validation ensures the reliability and accuracy of our model by rigorously testing its predictions against separate datasets.

Fine-Tuning and Optimization: In the pursuit of optimal performance, our methodology includes fine-tuning and optimization stages. This involves meticulous adjustment of model parameters, hyperparameter tuning, and exploration of different architectures to enhance predictive accuracy. By fine-tuning the model iteratively, we aim to achieve the highest level of precision in emotion detection across diverse modalities. Integrating LLM Outputs with Other Modalities: A critical aspect of our methodology involves developing a framework for seamlessly integrating the outputs of LLMs with image and audio data analysis. This ensures that insights from textual analysis complement those derived from visual and auditory cues, enriching the overall emotion prediction process. Through meticulous integration and alignment of data from different modalities, we strive to create a unified model that captures the richness and complexity of human emotions in digital interactions. Also, we propose the use of methods to efficiently fine-tune parameters, like LoRA, QLoRA, Adapter Modules and BitFit, each of which only fine-tune a small subset of parameters to a great effect and will help tune the models for emotion recognition.

Evaluation: In assessing the effectiveness of our multimodal emotion detection system, we employ a comprehensive evaluation framework that encompasses accuracy, reliability, real-time performance, scalability, and robustness. Precision, recall, and F1 score metrics serve as benchmarks for evaluating the system's accuracy and reliability in classifying emotions across various datasets. Real-time performance evaluation focuses on measuring system latency and computational efficiency, crucial for applications requiring immediate responses to user emotions. Scalability and robustness assessments ensure the system's ability to maintain performance consistency across diverse datasets and operational environments, reflecting its adaptability and reliability in real-world scenarios. By rigorously evaluating these aspects, we aim to demonstrate the efficacy and viability of our multimodal emotion detection system in enhancing human-computer interaction and fostering empathetic digital experiences.

3. Results and Discussions

Training Data Preprocessing:

The provided code demonstrates the preprocessing steps for the training data. Using the BERT tokenizer, the text data is tokenized, ensuring that it is properly formatted for input into the model. Additionally, the code defines a custom dataset class (`EmotionDataset`) to organize the tokenized encodings along with their corresponding labels.

Model Training:

The training arguments are configured, including parameters such as the number of training epochs, batch sizes, and logging settings. These settings are crucial for optimizing the training process and monitoring the model's performance. The `Trainer` class from the Transformers library is employed to orchestrate the training process using the specified model, training arguments, and datasets.

Evaluation Strategy:

The evaluation strategy is set to occur at the end of each epoch, as specified by `evaluation_strategy='epoch'`. This approach allows for comprehensive evaluation of the model's performance after each round of training, facilitating the detection of trends and potential overfitting.

Model Evaluation:

After training the model, it needs to be evaluated using the validation dataset (`val_dataset`). This step is crucial for assessing the model's generalization capabilities and identifying areas for improvement. Metrics such as accuracy, precision, recall, and F1-score can be computed to gauge the model's performance across different emotional categories.

Data Loading and Preprocessing:

The provided code snippet loads the dataset from a CSV file and preprocesses the text data by converting it to lowercase. This preprocessing step ensures consistency in text representation, which is essential for effective model training and evaluation.

Train-Test Split:

Before training the model, the dataset is typically split into training and testing subsets. However, the provided code does not explicitly show this step. It's essential to ensure that the model is evaluated on unseen data to obtain an unbiased estimate of its performance.

Future Directions:

To enhance the robustness and efficacy of the emotion detection framework, several avenues can be explored. This includes experimenting with different pre-trained language models, fine-tuning hyperparameters, incorporating additional modalities (e.g., gesture recognition), and augmenting the dataset to capture a broader spectrum of emotional expressions. Furthermore, conducting user studies to evaluate the framework's real-world usability and impact on human-computer interaction would provide valuable insights for refinement and optimization.

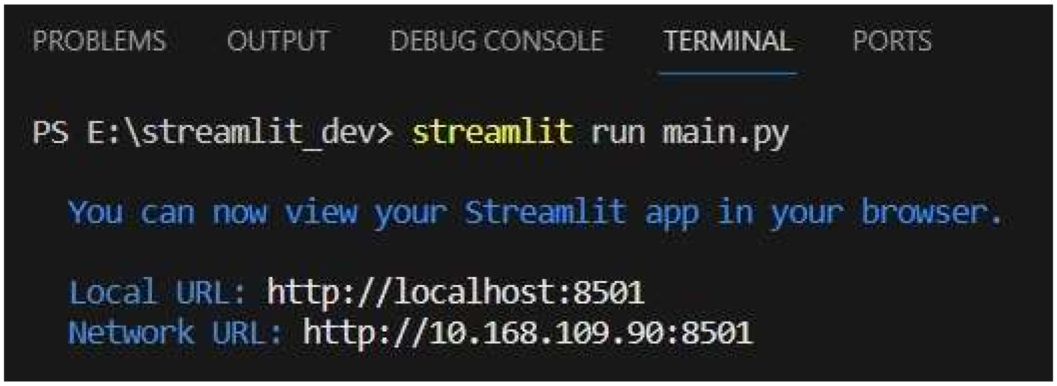
4. Conclusion

In conclusion, Our EmotiSense project represents a significant advancement in the field of emotion detection systems, addressing the limitations of traditional unimodal approaches by harnessing the power of multimodal data integration. By incorporating insights from facial expressions, vocal tones, and textual cues, our framework offers a holistic understanding of human emotions, paving the way for more empathetic and nuanced human-computer interactions. Through the convergence of cutting-edge technologies such as large language models (LLMs) and machine learning, EmotiSense not only enhances accuracy and contextual understanding but also fosters genuine human empathy in digital interactions, thereby redefining the landscape of human-computer interaction paradigms.

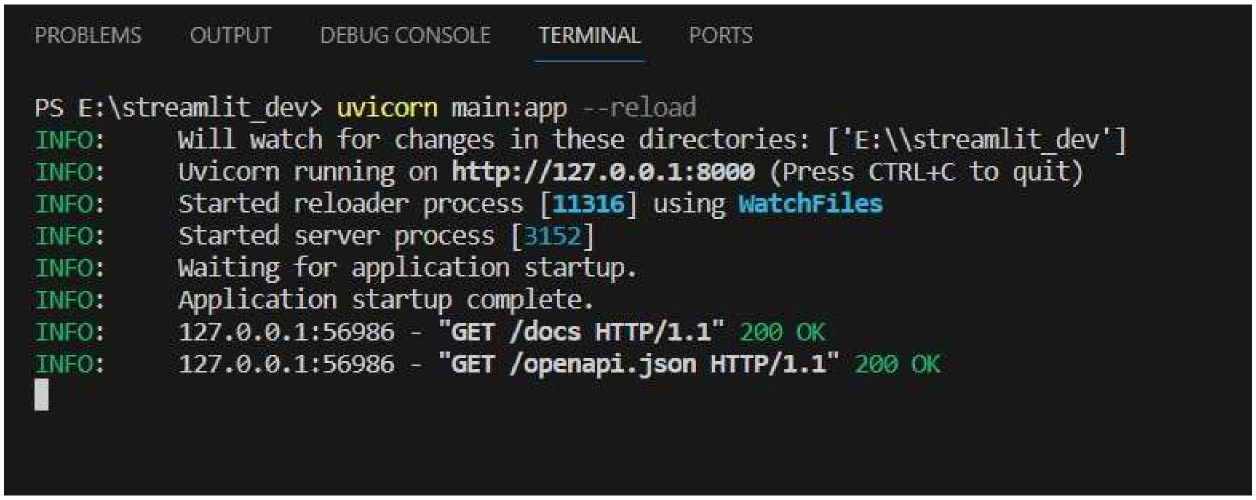
Moving forward, the transformative potential of EmotiSense extends beyond technological innovation, offering promising implications for various domains including user experience design, mental health support, customer service, and beyond. As we continue to refine and optimize the framework, future research endeavors may explore avenues for enhancing model robustness, scalability, and real-time performance. Additionally, user-centric evaluations and deployment in real-world scenarios will be crucial for validating the framework's efficacy and ensuring its seamless integration into diverse applications. Ultimately, EmotiSense stands as a testament to the synergy between technology and empathy, heralding a new era of emotionally intelligent computing that empowers users and fosters deeper human connections in the digital age.

Appendices

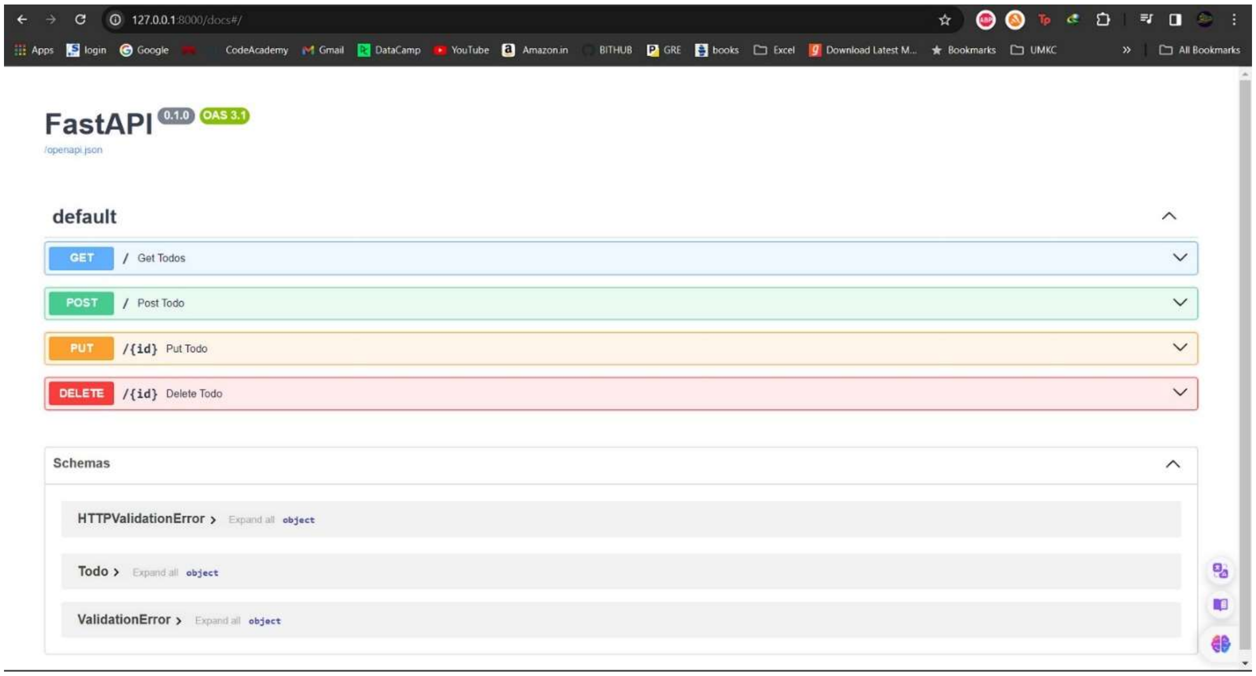
FRONT END SCREENSHOT'S:



```
PS E:\streamlit_dev> streamlit run main.py

  You can now view your Streamlit app in your browser.

  Local URL: http://localhost:8501
  Network URL: http://10.168.109.90:8501
```



```
PS E:\streamlit_dev> uvicorn main:app --reload
INFO:     Will watch for changes in these directories: ['E:\\streamlit_dev']
INFO:     Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
INFO:     Started reloader process [11316] using WatchFiles
INFO:     Started server process [3152]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
INFO:     127.0.0.1:56986 - "GET /docs HTTP/1.1" 200 OK
INFO:     127.0.0.1:56986 - "GET /openapi.json HTTP/1.1" 200 OK
```



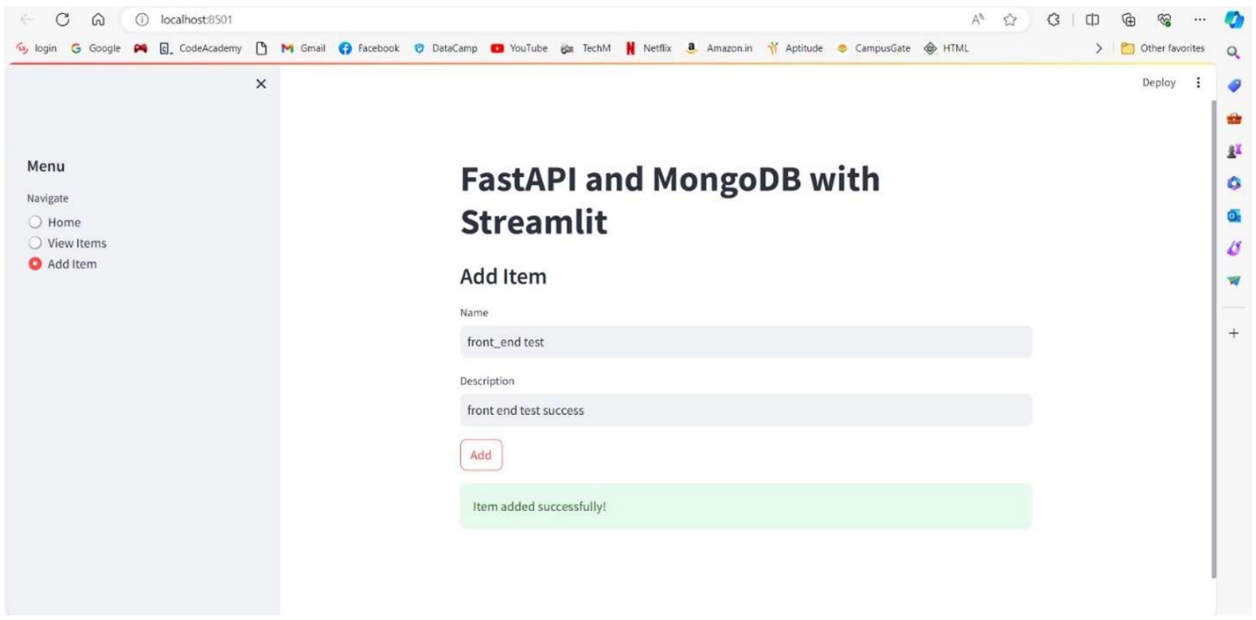
FastAPI 0.1.0 OAS 3.1
/openapi.json

default

GET / Get Todos
POST / Post Todo
PUT /{id} Put Todo
DELETE /{id} Delete Todo

Schemas

HTTPValidationError > Expand all object
Todo > Expand all object
ValidationError > Expand all object



Menu
Navigate
Home
View Items
Add Item [✓]

FastAPI and MongoDB with Streamlit

Add Item

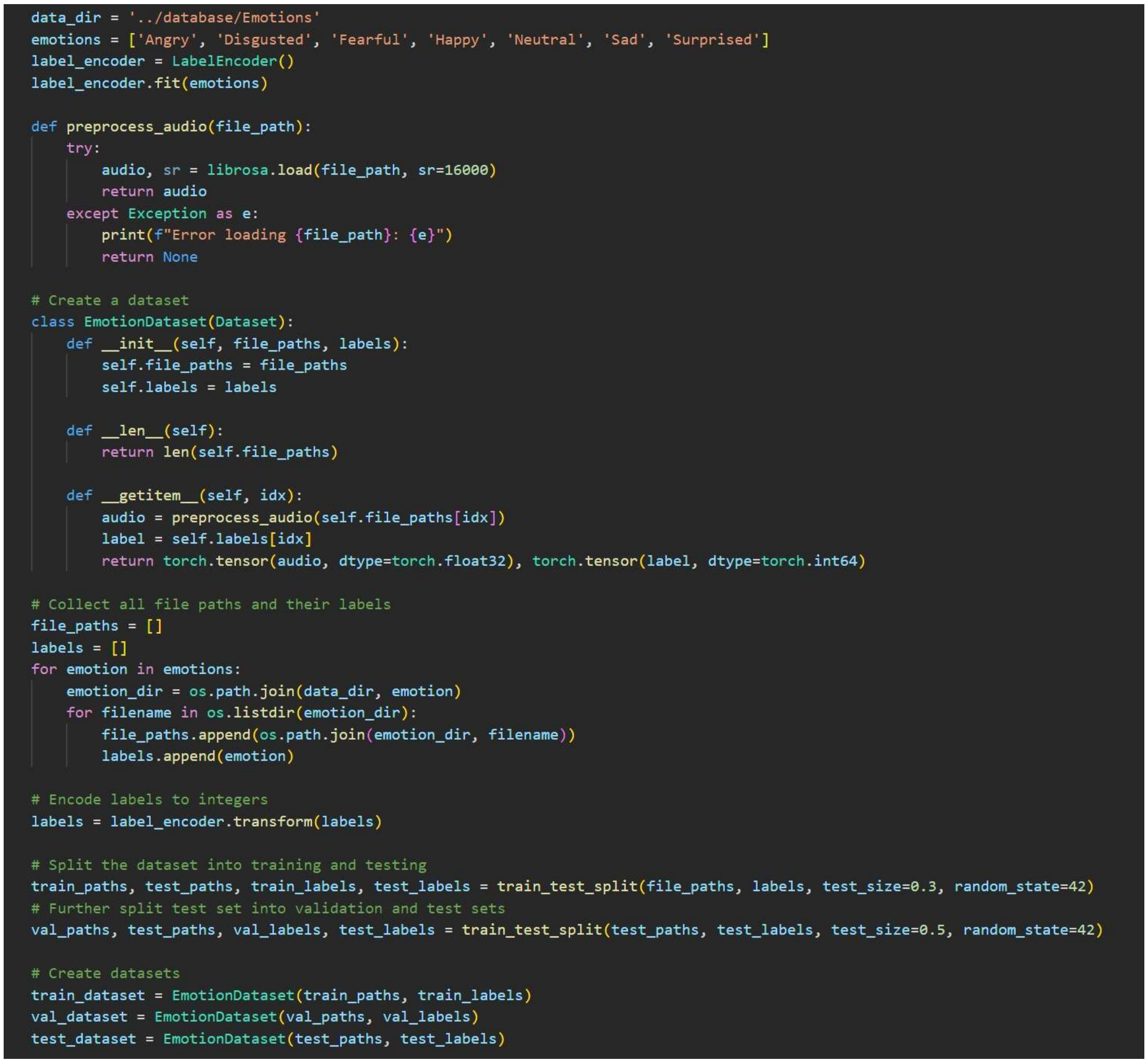
Name: front_end test

Description: front end test success

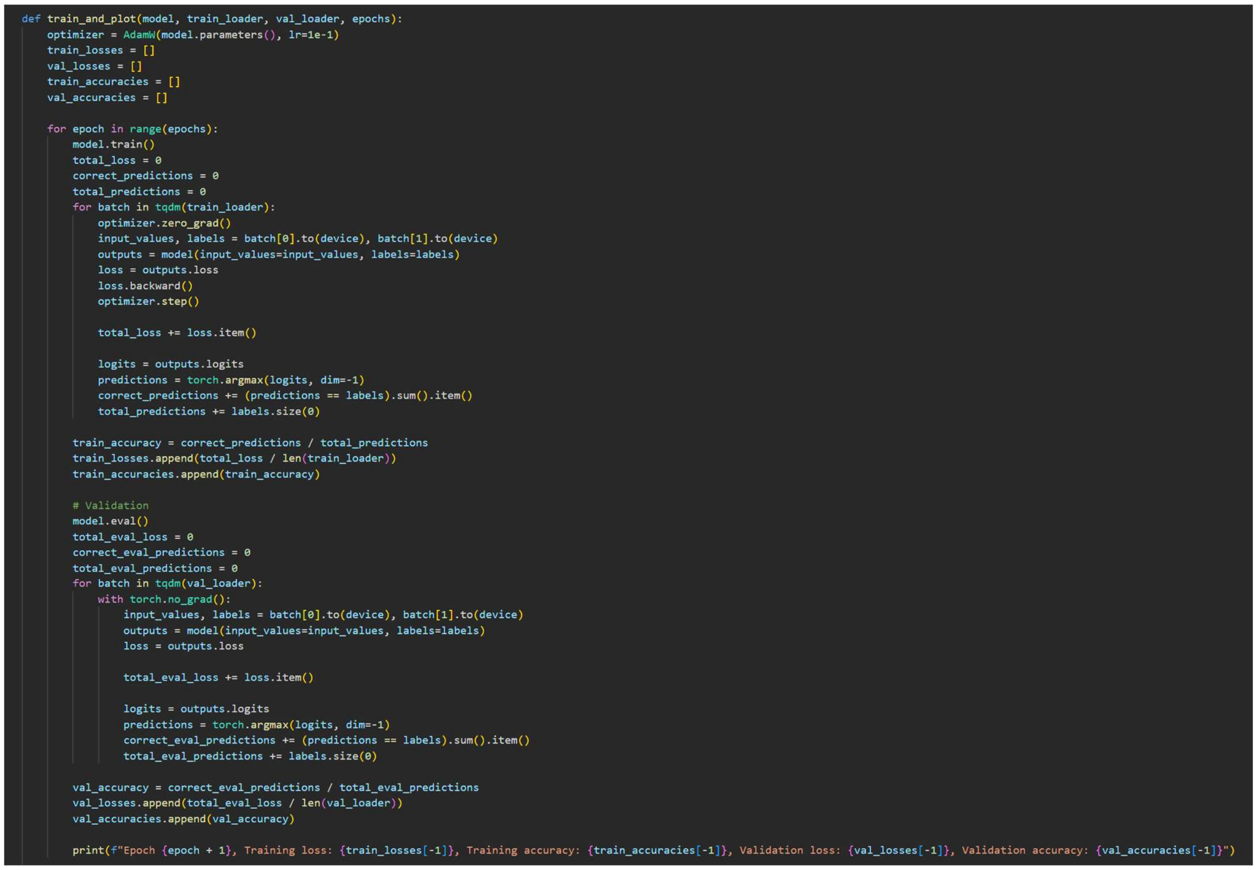
Add

Item added successfully!

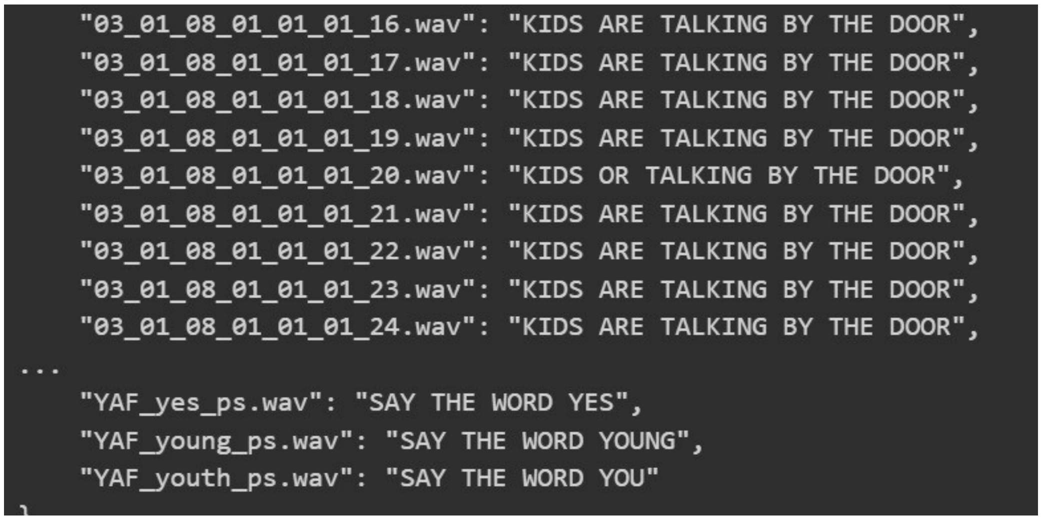
ALGORITHM SCREENSHOT’S :



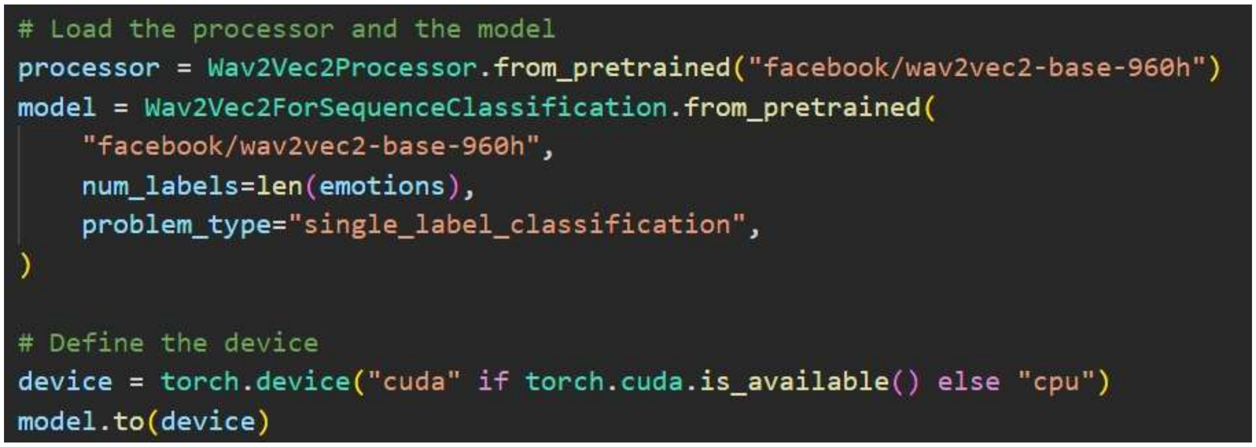
```python
data_dir = '../database/Emotions'
emotions = ['Angry', 'Disgusted', 'Fearful', 'Happy', 'Neutral', 'Sad', 'Surprised']
label_encoder = LabelEncoder()
label_encoder.fit(emotions)

def preprocess_audio(file_path):
    try:
        audio, sr = librosa.load(file_path, sr=16000)
        return audio
    except Exception as e:
        print(f"Error loading {file_path}: {e}")
        return None

# Create a dataset
class EmotionDataset(Dataset):
    def __init__(self, file_paths, labels):
        self.file_paths = file_paths
        self.labels = labels

    def __len__(self):
        return len(self.file_paths)

    def __getitem__(self, idx):
        audio = preprocess_audio(self.file_paths[idx])
        label = self.labels[idx]
        return torch.tensor(audio, dtype=torch.float32), torch.tensor(label, dtype=torch.int64)

# Collect all file paths and their labels
file_paths = []
labels = []
for emotion in emotions:
    emotion_dir = os.path.join(data_dir, emotion)
    for filename in os.listdir(emotion_dir):
        file_paths.append(os.path.join(emotion_dir, filename))
        labels.append(emotion)

# Encode labels to integers
labels = label_encoder.transform(labels)

# Split the dataset into training and testing
train_paths, test_paths, train_labels, test_labels = train_test_split(file_paths, labels, test_size=0.3, random_state=42)
# Further split test set into validation and test sets
val_paths, test_paths, val_labels, test_labels = train_test_split(test_paths, test_labels, test_size=0.5, random_state=42)

# Create datasets
train_dataset = EmotionDataset(train_paths, train_labels)
val_dataset = EmotionDataset(val_paths, val_labels)
test_dataset = EmotionDataset(test_paths, test_labels)
```



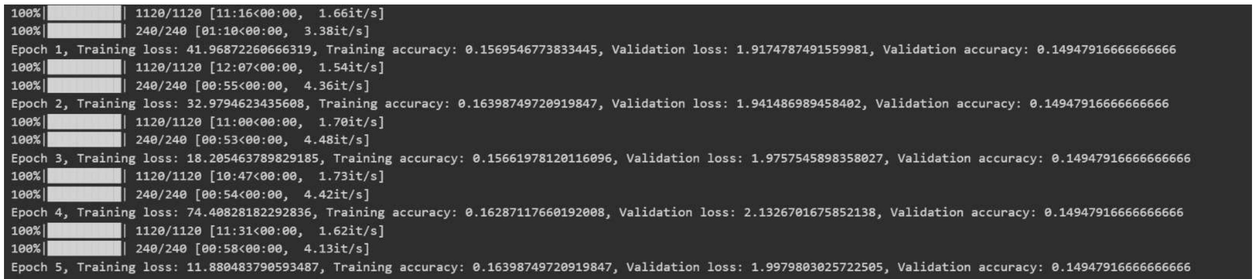
```python
def train_and_plot(model, train_loader, val_loader, epochs):
    optimizer = AdamW(model.parameters(), lr=1e-1)
    train_losses = []
    val_losses = []
    train_accuracies = []
    val_accuracies = []

    for epoch in range(epochs):
        model.train()
        total_loss = 0
        correct_predictions = 0
        total_predictions = 0
        for batch in tqdm(train_loader):
            optimizer.zero_grad()
            input_values, labels = batch[0].to(device), batch[1].to(device)
            outputs = model(input_values=input_values, labels=labels)
            loss = outputs.loss
            loss.backward()
            optimizer.step()

            total_loss += loss.item()

            logits = outputs.logits
            predictions = torch.argmax(logits, dim=-1)
            correct_predictions += (predictions == labels).sum().item()
            total_predictions += labels.size(0)

        train_accuracy = correct_predictions / total_predictions
        train_losses.append(total_loss / len(train_loader))
        train_accuracies.append(train_accuracy)

        # Validation
        model.eval()
        total_eval_loss = 0
        correct_eval_predictions = 0
        total_eval_predictions = 0
        for batch in tqdm(val_loader):
            with torch.no_grad():
                input_values, labels = batch[0].to(device), batch[1].to(device)
                outputs = model(input_values=input_values, labels=labels)
                loss = outputs.loss

                total_eval_loss += loss.item()

                logits = outputs.logits
                predictions = torch.argmax(logits, dim=-1)
                correct_eval_predictions += (predictions == labels).sum().item()
                total_eval_predictions += labels.size(0)

        val_accuracy = correct_eval_predictions / total_eval_predictions
        val_losses.append(total_eval_loss / len(val_loader))
        val_accuracies.append(val_accuracy)

        print(f"Epoch {epoch + 1}, Training loss: {train_losses[-1]}, Training accuracy: {train_accuracies[-1]}, Validation loss: {val_losses[-1]}, Validation accuracy: {val_accuracies[-1]}")
```



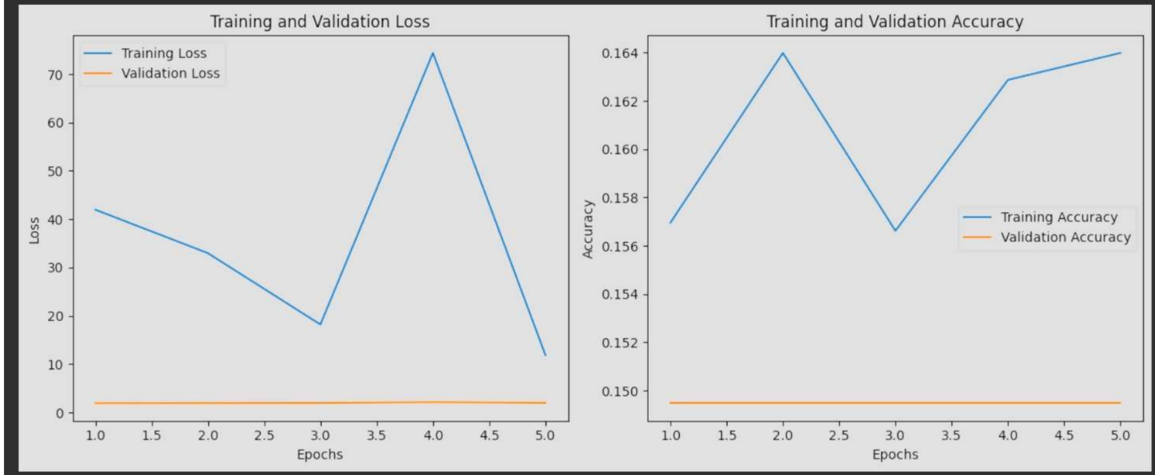
```
    "03_01_08_01_01_01_16.wav": "KIDS ARE TALKING BY THE DOOR",
    "03_01_08_01_01_01_17.wav": "KIDS ARE TALKING BY THE DOOR",
    "03_01_08_01_01_01_18.wav": "KIDS ARE TALKING BY THE DOOR",
    "03_01_08_01_01_01_19.wav": "KIDS ARE TALKING BY THE DOOR",
    "03_01_08_01_01_01_20.wav": "KIDS OR TALKING BY THE DOOR",
    "03_01_08_01_01_01_21.wav": "KIDS ARE TALKING BY THE DOOR",
    "03_01_08_01_01_01_22.wav": "KIDS ARE TALKING BY THE DOOR",
    "03_01_08_01_01_01_23.wav": "KIDS ARE TALKING BY THE DOOR",
    "03_01_08_01_01_01_24.wav": "KIDS ARE TALKING BY THE DOOR",
...
    "YAF_yes_ps.wav": "SAY THE WORD YES",
    "YAF_young_ps.wav": "SAY THE WORD YOUNG",
    "YAF_youth_ps.wav": "SAY THE WORD YOU"
}
```

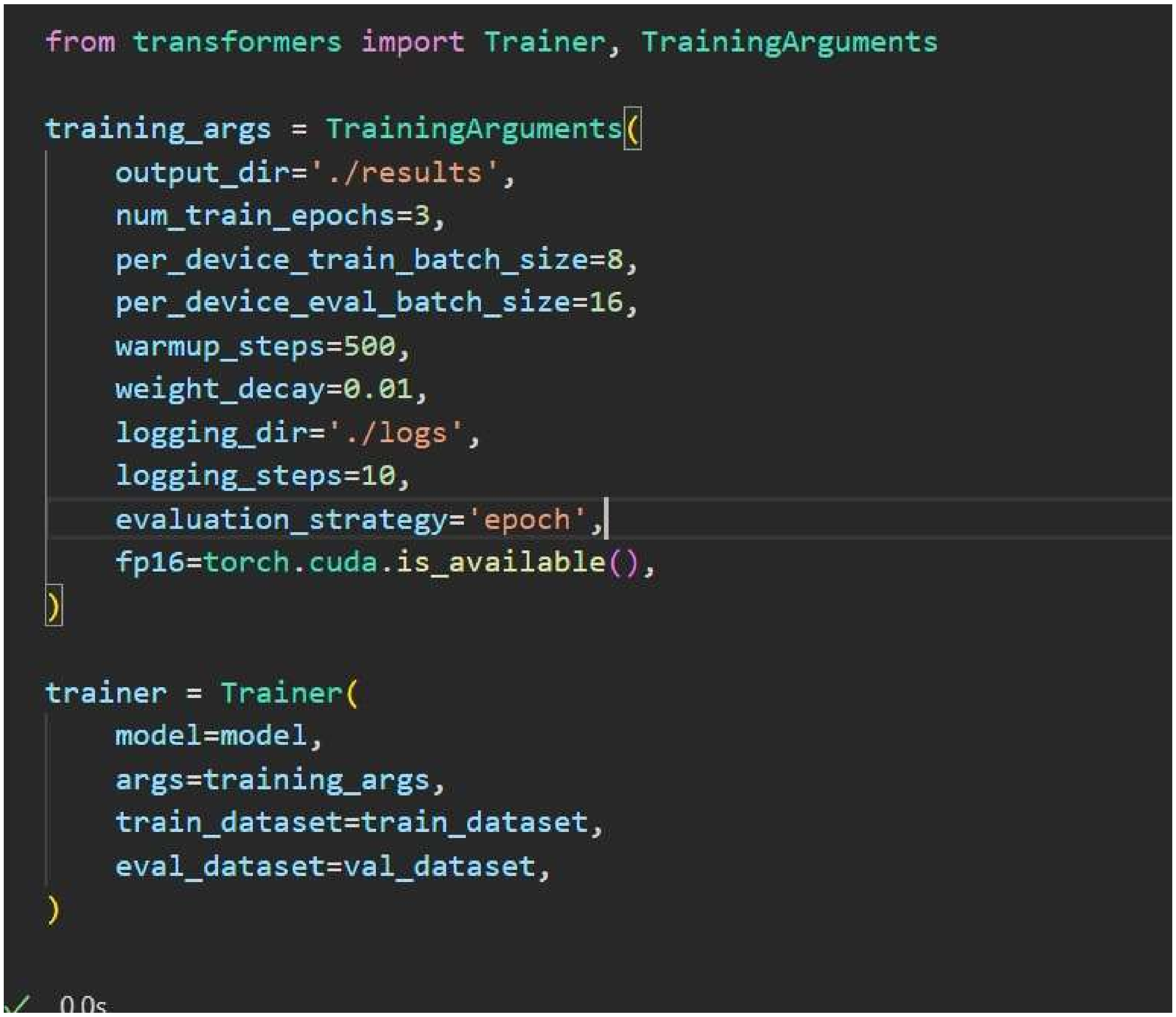


```python
# Load the processor and the model
processor = Wav2Vec2Processor.from_pretrained("facebook/wav2vec2-base-960h")
model = Wav2Vec2ForSequenceClassification.from_pretrained(
    "facebook/wav2vec2-base-960h",
    num_labels=len(emotions),
    problem_type="single_label_classification",
)

# Define the device
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)
```

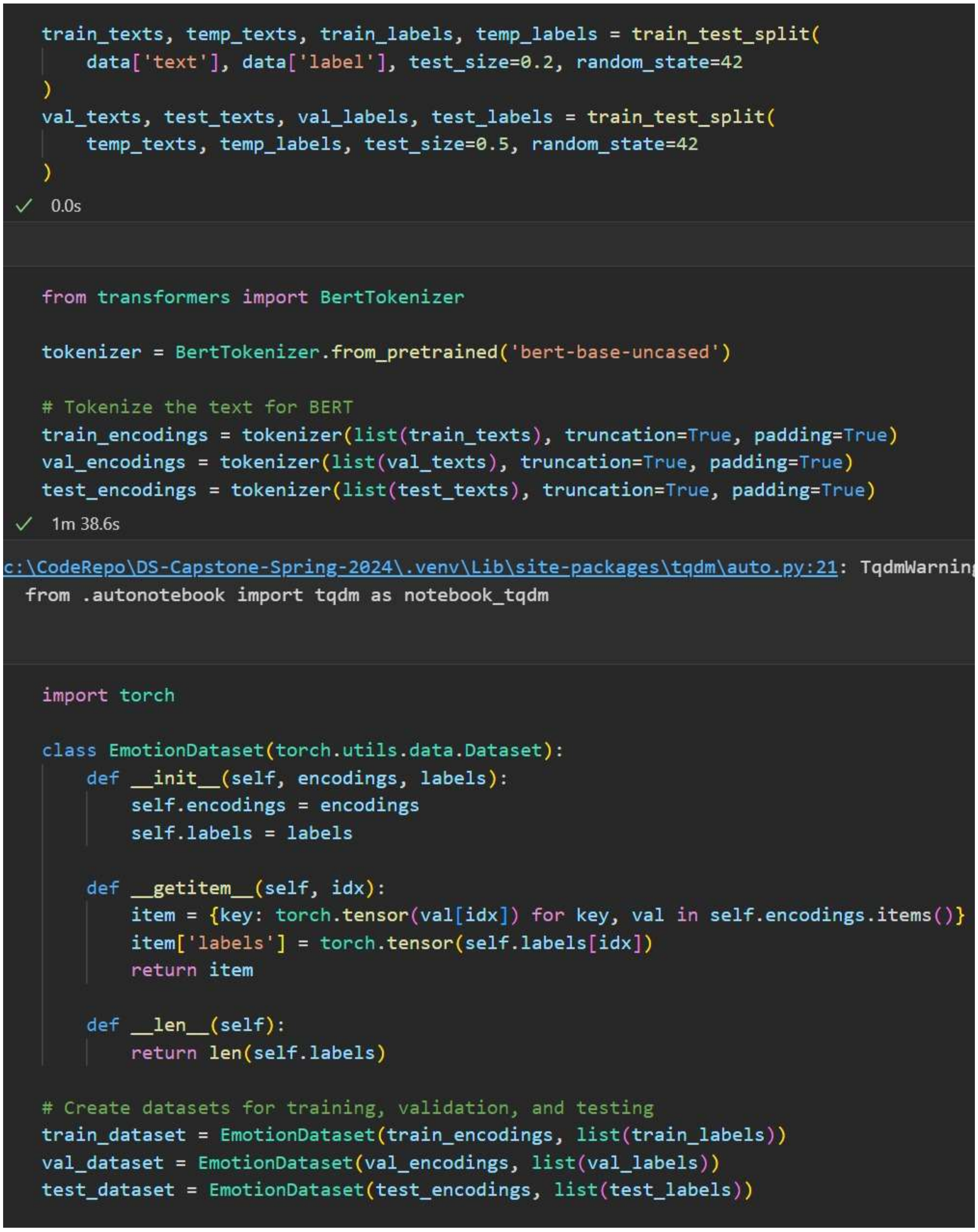


```
100%|██████████| 1120/1120 [11:16<00:00,  1.66it/s]
100%|██████████| 240/240 [01:10<00:00,  3.38it/s]
Epoch 1, Training loss: 41.96872260666319, Training accuracy: 0.1569546773833445, Validation loss: 1.9174787491559981, Validation accuracy: 0.14947916666666666
100%|██████████| 1120/1120 [12:07<00:00,  1.54it/s]
100%|██████████| 240/240 [00:55<00:00,  4.36it/s]
Epoch 2, Training loss: 32.9794623435608, Training accuracy: 0.16398749720919847, Validation loss: 1.941486989458402, Validation accuracy: 0.14947916666666666
100%|██████████| 1120/1120 [11:00<00:00,  1.70it/s]
100%|██████████| 240/240 [00:53<00:00,  4.48it/s]
Epoch 3, Training loss: 18.205463789829185, Training accuracy: 0.15661978120116096, Validation loss: 1.9757545898358027, Validation accuracy: 0.14947916666666666
100%|██████████| 1120/1120 [10:47<00:00,  1.73it/s]
100%|██████████| 240/240 [00:54<00:00,  4.42it/s]
Epoch 4, Training loss: 74.40828182292836, Training accuracy: 0.16287117660192008, Validation loss: 2.1326701675852138, Validation accuracy: 0.14947916666666666
100%|██████████| 1120/1120 [11:31<00:00,  1.62it/s]
100%|██████████| 240/240 [00:58<00:00,  4.13it/s]
Epoch 5, Training loss: 11.880483790593487, Training accuracy: 0.16398749720919847, Validation loss: 1.9979803025722505, Validation accuracy: 0.14947916666666666
```





```
from transformers import Trainer, TrainingArguments

training_args = TrainingArguments(
    output_dir='./results',
    num_train_epochs=3,
    per_device_train_batch_size=8,
    per_device_eval_batch_size=16,
    warmup_steps=500,
    weight_decay=0.01,
    logging_dir='./logs',
    logging_steps=10,
    evaluation_strategy='epoch',
    fp16=torch.cuda.is_available(),
)

trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=val_dataset,
)
```



```
train_texts, temp_texts, train_labels, temp_labels = train_test_split(
    data['text'], data['label'], test_size=0.2, random_state=42
)
val_texts, test_texts, val_labels, test_labels = train_test_split(
    temp_texts, temp_labels, test_size=0.5, random_state=42
)
```

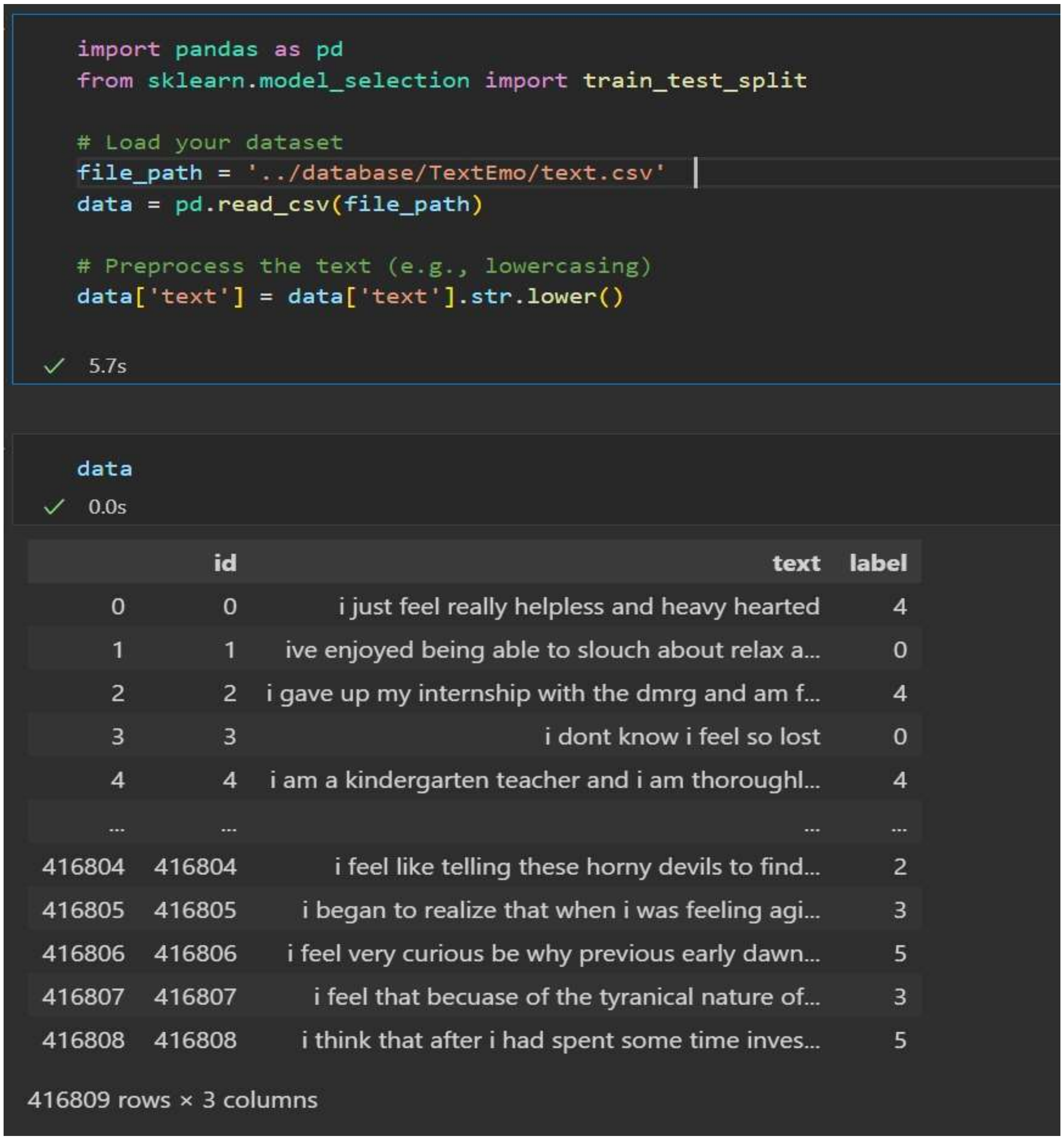
✓ 0.0s

```
from transformers import BertTokenizer

tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')

# Tokenize the text for BERT
train_encodings = tokenizer(list(train_texts), truncation=True, padding=True)
val_encodings = tokenizer(list(val_texts), truncation=True, padding=True)
test_encodings = tokenizer(list(test_texts), truncation=True, padding=True)
```

✓ 1m 38.6s

```
c:\CodeRepo\DS-Capstone-Spring-2024\.venv\Lib\site-packages\tqdm\auto.py:21: TqdmWarnin
  from .autonotebook import tqdm as notebook_tqdm
```

```
import torch

class EmotionDataset(torch.utils.data.Dataset):
    def __init__(self, encodings, labels):
        self.encodings = encodings
        self.labels = labels

    def __getitem__(self, idx):
        item = {key: torch.tensor(val[idx]) for key, val in self.encodings.items()}
        item['labels'] = torch.tensor(self.labels[idx])
        return item

    def __len__(self):
        return len(self.labels)

# Create datasets for training, validation, and testing
train_dataset = EmotionDataset(train_encodings, list(train_labels))
val_dataset = EmotionDataset(val_encodings, list(val_labels))
test_dataset = EmotionDataset(test_encodings, list(test_labels))
```



```
import pandas as pd
from sklearn.model_selection import train_test_split

# Load your dataset
file_path = '../database/TextEmo/text.csv'
data = pd.read_csv(file_path)

# Preprocess the text (e.g., lowercasing)
data['text'] = data['text'].str.lower()
```

✓ 5.7s

```
data
```

✓ 0.0s

```
            id                                              text  label
0            0             i just feel really helpless and heavy hearted      4
1            1      ive enjoyed being able to slouch about relax a...      0
2            2      i gave up my internship with the dmrg and am f...      4
3            3                               i dont know i feel so lost      0
4            4      i am a kindergarten teacher and i am thoroughl...      4
...        ...                                                  ...    ...
416804  416804          i feel like telling these horny devils to find...      2
416805  416805        i began to realize that when i was feeling agi...      3
416806  416806      i feel very curious be why previous early dawn...      5
416807  416807         i feel that becuase of the tyranical nature of...      3
416808  416808        i think that after i had spent some time inves...      5

416809 rows × 3 columns
```