In [2]:
```
import json
import torch
from transformers import BertTokenizer
import os

# Function to load JSONL (JSON lines) file
def load_json_file(filepath):
    with open(filepath, 'r', encoding='utf-8') as f:
        return [json.loads(line.strip()) for line in f]

# Paths to data files
base_path = r'/Users/shetty/Downloads/DM3'
train_file = os.path.join(base_path, '/Users/shetty/Downloads/train.js
test_file = os.path.join(base_path, '/Users/shetty/Downloads/test.json
val_file = os.path.join(base_path, '/Users/shetty/Downloads/validation

# Load datasets
train_data = load_json_file(train_file)
test_data = load_json_file(test_file)
val_data = load_json_file(val_file)

# Print the structure of the first item in train_data
print("Structure of the first item in train_data:")
print(json.dumps(train_data[0], indent=2))

# Define label classes
all_labels = ['anger', 'anticipation', 'disgust', 'fear', 'joy',
              'love', 'optimism', 'pessimism', 'sadness', 'surprise',

# Convert labels to one-hot encoded lists
def convert_labels_to_one_hot(data, label_classes):
    """
    Add a 'labels' key to each data item with one-hot-encoded labels.
    """
    print("Keys in a data item:", list(data[0].keys()))
    for item in data:
        item['labels'] = [float(item[label]) for label in label_classe
    return data

# Apply one-hot encoding to the datasets
train_data = convert_labels_to_one_hot(train_data, all_labels)
val_data = convert_labels_to_one_hot(val_data, all_labels)
test_data = convert_labels_to_one_hot(test_data, all_labels)

# Initialize the BERT tokenizer
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')

# Tokenize and encode the texts
```

```
def encode_texts(data, tokenizer, max_length=128):
    """
    Tokenizes and encodes text data using the tokenizer.
    """
    # Assuming 'tweet' is the key for text content. Adjust this if nec
    texts = [item['tweet'] for item in data]
    return tokenizer(texts, padding=True, truncation=True, max_length=

try:
    # Encode texts for each dataset
    train_encodings = encode_texts(train_data, tokenizer)
    test_encodings = encode_texts(test_data, tokenizer)
    val_encodings = encode_texts(val_data, tokenizer)
except KeyError as e:
    print(f"KeyError: {e}. The key for text content might be incorrect
    print("Available keys in train_data:", list(train_data[0].keys()))

# Convert labels to tensors
train_labels = torch.tensor([item['labels'] for item in train_data], d
test_labels = torch.tensor([item['labels'] for item in test_data], dty
val_labels = torch.tensor([item['labels'] for item in val_data], dtype

# Print shapes to verify the labels
print("Train labels shape:", train_labels.shape)
print("Test labels shape:", test_labels.shape)
print("Validation labels shape:", val_labels.shape)

# Print encoding shapes if successful
if 'train_encodings' in locals():
    print("Train encodings shape:", train_encodings['input_ids'].shape
    print("Test encodings shape:", test_encodings['input_ids'].shape)
    print("Validation encodings shape:", val_encodings['input_ids'].sh
```

```
Structure of the first item in train_data:
{
  "ID": "2017-En-21378",
  "Tweet": "@ibishotelsuk @ibisHotels_FR @ibishotelbr @ibishotel @ibi
shotelBDG @ibishotel1 to give me my keys back. They aren't for my hou
se! #shocking",
  "anger": true,
  "anticipation": false,
  "disgust": true,
  "fear": true,
  "joy": false,
  "love": false,
  "optimism": false,
  "pessimism": false,
  "sadness": false,
  "surprise": true,
  "trust": false
}
```

```
Keys in a data item: ['ID', 'Tweet', 'anger', 'anticipation', 'disgus
t', 'fear', 'joy', 'love', 'optimism', 'pessimism', 'sadness', 'surpr
ise', 'trust']
Keys in a data item: ['ID', 'Tweet', 'anger', 'anticipation', 'disgus
t', 'fear', 'joy', 'love', 'optimism', 'pessimism', 'sadness', 'surpr
ise', 'trust']
Keys in a data item: ['ID', 'Tweet', 'anger', 'anticipation', 'disgus
t', 'fear', 'joy', 'love', 'optimism', 'pessimism', 'sadness', 'surpr
ise', 'trust']
KeyError: 'tweet'. The key for text content might be incorrect.
Available keys in train_data: ['ID', 'Tweet', 'anger', 'anticipatio
n', 'disgust', 'fear', 'joy', 'love', 'optimism', 'pessimism', 'sadne
ss', 'surprise', 'trust', 'labels']
Train labels shape: torch.Size([3000, 11])
Test labels shape: torch.Size([1500, 11])
Validation labels shape: torch.Size([400, 11])
Train encodings shape: torch.Size([500, 56])
Test encodings shape: torch.Size([100, 51])
Validation encodings shape: torch.Size([100, 58])
```

In [3]:

```python
import json
import torch
from torch.utils.data import TensorDataset, DataLoader
from transformers import BertTokenizer
import os

# Function to load JSONL (JSON lines) file
def load_json_file(filepath):
    with open(filepath, 'r', encoding='utf-8') as f:
        return [json.loads(line.strip()) for line in f]

# Paths to data files
base_path = r'/Users/shetty/Downloads/DM3'
train_file = os.path.join(base_path, '/Users/shetty/Downloads/train.js
test_file = os.path.join(base_path, '/Users/shetty/Downloads/test.json
val_file = os.path.join(base_path, '/Users/shetty/Downloads/validation

# Load datasets
train_data = load_json_file(train_file)
test_data = load_json_file(test_file)
val_data = load_json_file(val_file)

# Print the structure of the first item in train_data
print("Structure of the first item in train_data:")
print(json.dumps(train_data[0], indent=2))

# Define label classes
all_labels = ['anger', 'anticipation', 'disgust', 'fear', 'joy',
              'love', 'optimism', 'pessimism', 'sadness', 'surprise',
```

```
# Function to convert labels to one-hot encoded lists
def convert_labels_to_one_hot(data, label_classes):
    """
    Adds a 'labels' key to each data item with one-hot-encoded labels.
    """
    print("Keys in a data item:", list(data[0].keys()))
    for item in data:
        item['labels'] = [float(item[label]) for label in label_classe
    return data

# Apply one-hot encoding to the datasets
train_data = convert_labels_to_one_hot(train_data, all_labels)
val_data = convert_labels_to_one_hot(val_data, all_labels)
test_data = convert_labels_to_one_hot(test_data, all_labels)

# Initialize the BERT tokenizer
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')

# Function to tokenize and encode the texts
def encode_texts(data, text_key='Tweet', max_length=128):
    """
    Tokenizes and encodes text data using the tokenizer.
    """
    if text_key not in data[0]:
        raise KeyError(f"'{text_key}' not found in data. Available key
    texts = [item[text_key] for item in data]
    return tokenizer(texts, padding=True, truncation=True, max_length=

# Encode texts for each dataset
try:
    train_encodings = encode_texts(train_data)
    test_encodings = encode_texts(test_data)
    val_encodings = encode_texts(val_data)
    print("Text encoding completed successfully.")
except KeyError as e:
    print(f"Error during text encoding: {e}")
    raise

# Convert labels to tensors
train_labels = torch.tensor([item['labels'] for item in train_data], d
test_labels = torch.tensor([item['labels'] for item in test_data], dty
val_labels = torch.tensor([item['labels'] for item in val_data], dtype

# Print shapes to verify the labels and encodings
print("Train labels shape:", train_labels.shape)
print("Test labels shape:", test_labels.shape)
print("Validation labels shape:", val_labels.shape)
print("Train encodings shape:", train_encodings['input_ids'].shape)
print("Test encodings shape:", test_encodings['input_ids'].shape)
```

```
print("Validation encodings shape:", val_encodings['input_ids'].shape)

# Function to create DataLoader
def create_dataloader(encodings, labels, batch_size=16):
    """
    Creates a DataLoader from encodings and labels.
    """
    input_ids = encodings['input_ids']
    attention_mask = encodings['attention_mask']
    dataset = TensorDataset(input_ids, attention_mask, labels)
    return DataLoader(dataset, batch_size=batch_size, shuffle=True)

# Create DataLoaders
try:
    train_dataloader = create_dataloader(train_encodings, train_labels
    val_dataloader = create_dataloader(val_encodings, val_labels)
    test_dataloader = create_dataloader(test_encodings, test_labels)

    # Print DataLoader sizes
    print("Train DataLoader size:", len(train_dataloader))
    print("Validation DataLoader size:", len(val_dataloader))
    print("Test DataLoader size:", len(test_dataloader))
except Exception as e:
    print(f"Error creating DataLoaders: {e}")
    print("Shape of train_encodings:", {k: v.shape for k, v in train_e
    print("Shape of train_labels:", train_labels.shape)
```

```
Structure of the first item in train_data:
{
  "ID": "2017-En-21378",
  "Tweet": "@ibishotelsuk @ibisHotels_FR @ibishotelbr @ibishotel @ibi
shotelBDG @ibishotel1 to give me my keys back. They aren't for my hou
se! #shocking",
  "anger": true,
  "anticipation": false,
  "disgust": true,
  "fear": true,
  "joy": false,
  "love": false,
  "optimism": false,
  "pessimism": false,
  "sadness": false,
  "surprise": true,
  "trust": false
}
Keys in a data item: ['ID', 'Tweet', 'anger', 'anticipation', 'disgus
t', 'fear', 'joy', 'love', 'optimism', 'pessimism', 'sadness', 'surpr
ise', 'trust']
Keys in a data item: ['ID', 'Tweet', 'anger', 'anticipation', 'disgus
t', 'fear', 'joy', 'love', 'optimism', 'pessimism', 'sadness', 'surpr
ise', 'trust']
```

```
ise', 'trust']
Keys in a data item: ['ID', 'Tweet', 'anger', 'anticipation', 'disgus
t', 'fear', 'joy', 'love', 'optimism', 'pessimism', 'sadness', 'surpr
ise', 'trust']
Text encoding completed successfully.
Train labels shape: torch.Size([3000, 11])

Test labels shape: torch.Size([1500, 11])
Validation labels shape: torch.Size([400, 11])
Train encodings shape: torch.Size([3000, 71])
Test encodings shape: torch.Size([1500, 59])
Validation encodings shape: torch.Size([400, 58])
Train DataLoader size: 188
Validation DataLoader size: 25
Test DataLoader size: 94
```

In [4]:

```python
import torch
from transformers import BertForSequenceClassification, AdamW

# Define the number of labels for multi-label classification
num_labels = len(all_labels)  # Should be 11 based on your previous co

# Initialize the BERT model for multi-label classification
def initialize_model(model_name='bert-base-uncased', num_labels=num_la
    """
    Initializes the BERT model for multi-label classification.
    Args:
        model_name (str): The name of the pretrained model to use.
        num_labels (int): The number of output labels.
    Returns:
        model (nn.Module): The initialized model.
    """
    model = BertForSequenceClassification.from_pretrained(
        model_name,
        num_labels=num_labels,
        problem_type="multi_label_classification"
    )
    print(f"Model initialized with {num_labels} output labels.")
    return model

# Initialize the optimizer
def initialize_optimizer(model, learning_rate=2e-5):
    """
    Initializes the AdamW optimizer for the model.
    Args:
        model (nn.Module): The model to optimize.
        learning_rate (float): The learning rate for the optimizer.
    Returns:
        optimizer: The initialized optimizer.
    """
    optimizer = AdamW(model.parameters(), lr=learning_rate)
```

```
    print(f"Optimizer initialized with learning rate {learning_rate}."
    return optimizer

# Set up the device (GPU or CPU)
def setup_device():
    """
    Determines whether to use a GPU or CPU for training.
    Returns:
        device (torch.device): The device to use.
    """
    device = torch.device("cuda" if torch.cuda.is_available() else "cp
    print(f"Model will run on device: {device}")
    return device

# Initialize the model, optimizer, and device
model = initialize_model()
optimizer = initialize_optimizer(model)
device = setup_device()

# Move the model to the selected device
model.to(device)
print("Model successfully moved to the selected device.")
```

```
Some weights of the model checkpoint at bert-base-uncased were not us
ed when initializing BertForSequenceClassification: ['cls.prediction
s.transform.LayerNorm.weight', 'cls.seq_relationship.weight', 'cls.se
q_relationship.bias', 'cls.predictions.decoder.weight', 'cls.predicti
ons.transform.dense.bias', 'cls.predictions.transform.dense.weight',
'cls.predictions.bias', 'cls.predictions.transform.LayerNorm.bias']
- This IS expected if you are initializing BertForSequenceClassificat
ion from the checkpoint of a model trained on another task or with an
other architecture (e.g. initializing a BertForSequenceClassification
model from a BertForPreTraining model).
- This IS NOT expected if you are initializing BertForSequenceClassif
ication from the checkpoint of a model that you expect to be exactly
identical (initializing a BertForSequenceClassification model from a
BertForSequenceClassification model).
Some weights of BertForSequenceClassification were not initialized fr
om the model checkpoint at bert-base-uncased and are newly initialize
d: ['classifier.bias', 'classifier.weight']
You should probably TRAIN this model on a down-stream task to be able
to use it for predictions and inference.

Model initialized with 11 output labels.
Optimizer initialized with learning rate 2e-05.
Model will run on device: cpu
Model successfully moved to the selected device.

/Users/shetty/anaconda3/anaconda3/lib/python3.11/site-packages/transf
ormers/optimization.py:407: FutureWarning: This implementation of Ada
mW is deprecated and will be removed in a future version. Use the PyT
```

```
orch implementation torch.optim.AdamW instead, or set `no_deprecation
_warning=True` to disable this warning
  warnings.warn(
```

In [ ]:

In [ ]:

In [1]:
```
import json
import torch
from torch.utils.data import DataLoader, TensorDataset
from transformers import BertTokenizer, BertForSequenceClassification
from torch.optim import AdamW
from torch.cuda.amp import autocast, GradScaler
from sklearn.metrics import classification_report
from tqdm import tqdm
import matplotlib.pyplot as plt

# Function to load JSON file
def load_json_file(filepath, max_samples=None):
    with open(filepath, 'r', encoding='utf-8') as f:
        data = [json.loads(line.strip()) for line in f]
    if max_samples:
        return data[:max_samples]
    return data

# File paths (update as needed)
train_path = '/Users/shetty/Downloads/train.json'
test_path = '/Users/shetty/Downloads/test.json'
val_path = '/Users/shetty/Downloads/validation.json'

# Load and preprocess the datasets
max_samples = 500  # Limit the training dataset size
train_data = load_json_file(train_path, max_samples)
test_data = load_json_file(test_path, max_samples // 5)
val_data = load_json_file(val_path, max_samples // 5)

print(f"Loaded {len(train_data)} train samples, {len(val_data)} valida

# Define label classes
all_labels = ['anger', 'anticipation', 'disgust', 'fear', 'joy',
              'love', 'optimism', 'pessimism', 'sadness', 'surprise',

# Convert labels to one-hot encoded lists
def convert_labels_to_list(data, label_classes):
    for item in data:
        item['labels'] = [float(item.get(label, 0)) for label in label
    return data
```

```
train_data = convert_labels_to_list(train_data, all_labels)
val_data = convert_labels_to_list(val_data, all_labels)
test_data = convert_labels_to_list(test_data, all_labels)

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

# Tokenize and encode the texts
def encode_texts(data):
    texts = [item['Tweet'] for item in data]  # Assuming 'Tweet' conta
    return tokenizer(texts, padding=True, truncation=True, max_length=

# Encode data
train_encodings = encode_texts(train_data)
val_encodings = encode_texts(val_data)
test_encodings = encode_texts(test_data)

# Convert labels to tensors
train_labels = torch.tensor([item['labels'] for item in train_data], d
val_labels = torch.tensor([item['labels'] for item in val_data], dtype
test_labels = torch.tensor([item['labels'] for item in test_data], dty

# Create DataLoader
def create_dataloader(encodings, labels, batch_size=16):
    dataset = TensorDataset(encodings['input_ids'], encodings['attenti
    return DataLoader(dataset, batch_size=batch_size, shuffle=True)

train_dataloader = create_dataloader(train_encodings, train_labels)
val_dataloader = create_dataloader(val_encodings, val_labels)
test_dataloader = create_dataloader(test_encodings, test_labels)

# Initialize the model
num_labels = len(all_labels)
model = BertForSequenceClassification.from_pretrained(
    'bert-base-uncased',
    num_labels=num_labels,
    problem_type="multi_label_classification"
)

# Set up optimizer and scaler for mixed precision training
optimizer = AdamW(model.parameters(), lr=2e-5)
scaler = GradScaler()

# Set device
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model.to(device)

# Training loop
num_epochs = 5
```

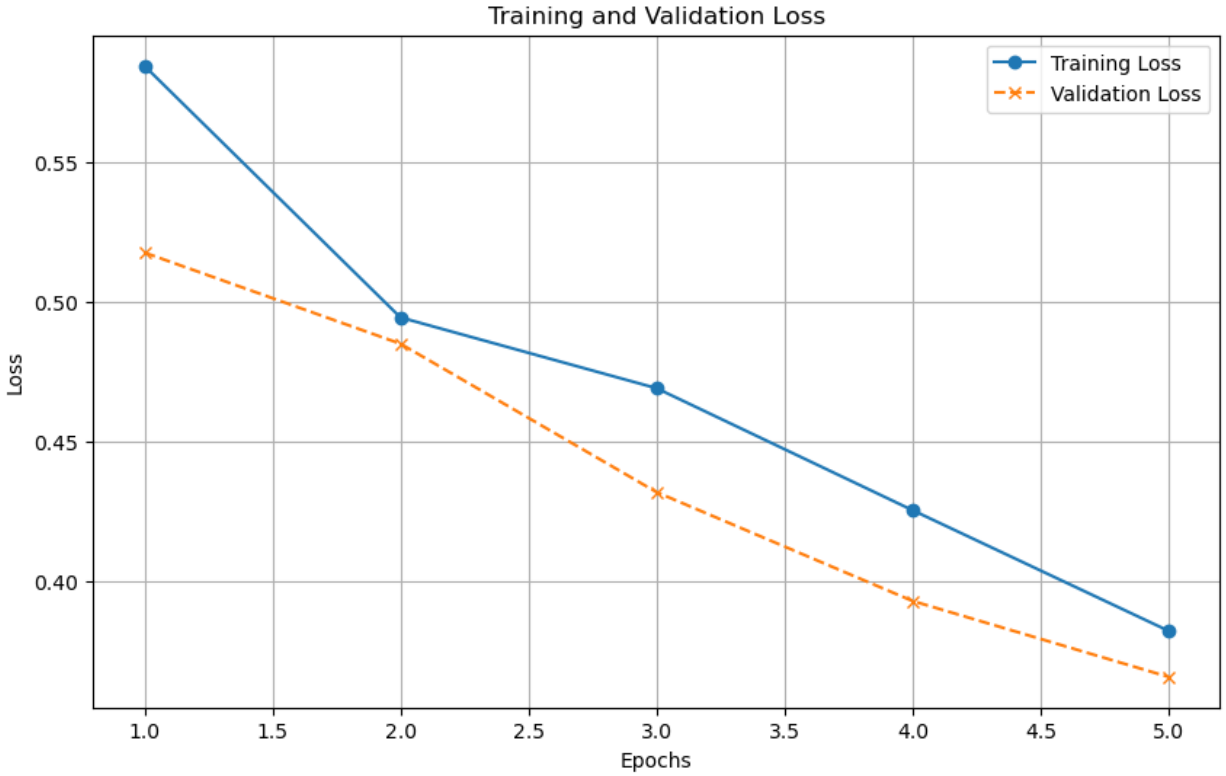
```
train_losses = []
val_losses = []

for epoch in range(num_epochs):
    model.train()
    total_train_loss = 0
    progress_bar = tqdm(train_dataloader, desc=f'Epoch {epoch+1}/{num_

    for batch in progress_bar:
        input_ids, attention_mask, labels = [b.to(device) for b in bat
        optimizer.zero_grad()

        with autocast():  # Mixed precision training
            outputs = model(input_ids, attention_mask=attention_mask,
            loss = outputs.loss

        scaler.scale(loss).backward()
        scaler.step(optimizer)
        scaler.update()

        total_train_loss += loss.item()
        progress_bar.set_postfix({'train_loss': f'{loss.item():.4f}'})

    avg_train_loss = total_train_loss / len(train_dataloader)
    train_losses.append(avg_train_loss)

    # Validation phase
    model.eval()
    total_val_loss = 0
    with torch.no_grad():
        for batch in tqdm(val_dataloader, desc=f'Epoch {epoch+1}/{num_
            input_ids, attention_mask, labels = [b.to(device) for b in
            with autocast():
                outputs = model(input_ids, attention_mask=attention_ma
            total_val_loss += outputs.loss.item()

    avg_val_loss = total_val_loss / len(val_dataloader)
    val_losses.append(avg_val_loss)

    print(f'Epoch {epoch+1}/{num_epochs}: Train Loss = {avg_train_loss
    print('-' * 50)

print("Training completed!")

# Save the model
torch.save(model.state_dict(), 'bert_multi_label_model.pth')
print("Model saved!")

# Plot learning curves
def plot_learning_curves(train_losses, val_losses):
```

```
    epochs = range(1, len(train_losses) + 1)
    plt.figure(figsize=(10, 6))
    plt.plot(epochs, train_losses, label="Training Loss", marker='o')
    plt.plot(epochs, val_losses, label="Validation Loss", linestyle='-
    plt.title("Training and Validation Loss")
    plt.xlabel("Epochs")
    plt.ylabel("Loss")
    plt.legend()
    plt.grid(True)
    plt.show()

plot_learning_curves(train_losses, val_losses)

from sklearn.metrics import classification_report

# Evaluate the model using sklearn.metrics
def evaluate_model(model, dataloader, label_names):
    model.eval()
    all_preds, all_labels = [], []
    with torch.no_grad():
        for batch in dataloader:
            input_ids, attention_mask, labels = [b.to(device) for b in
            outputs = model(input_ids, attention_mask=attention_mask)
            logits = outputs.logits
            preds = torch.sigmoid(logits).cpu().numpy() > 0.5  # Multi
            all_preds.extend(preds)
            all_labels.extend(labels.cpu().numpy())

    # Generate the classification report
    print(classification_report(all_labels, all_preds, target_names=la

# Evaluate on the test set
evaluate_model(model, test_dataloader, all_labels)
```

```
Loaded 500 train samples, 100 validation samples, 100 test samples

Some weights of the model checkpoint at bert-base-uncased were not us
ed when initializing BertForSequenceClassification: ['cls.prediction
s.transform.LayerNorm.weight', 'cls.seq_relationship.weight', 'cls.se
q_relationship.bias', 'cls.predictions.decoder.weight', 'cls.predicti
ons.transform.dense.bias', 'cls.predictions.transform.dense.weight',
'cls.predictions.bias', 'cls.predictions.transform.LayerNorm.bias']
- This IS expected if you are initializing BertForSequenceClassificat
ion from the checkpoint of a model trained on another task or with an
other architecture (e.g. initializing a BertForSequenceClassification
model from a BertForPreTraining model).
- This IS NOT expected if you are initializing BertForSequenceClassif
ication from the checkpoint of a model that you expect to be exactly
identical (initializing a BertForSequenceClassification model from a
BertForSequenceClassification model).
Some weights of BertForSequenceClassification were not initialized fr
```

```
om the model checkpoint at bert-base-uncased and are newly initialize
d: ['classifier.bias', 'classifier.weight']
You should probably TRAIN this model on a down-stream task to be able
to use it for predictions and inference.
/Users/shetty/anaconda3/anaconda3/lib/python3.11/site-packages/torch/
cuda/amp/grad_scaler.py:120: UserWarning: torch.cuda.amp.GradScaler i
s enabled, but CUDA is not available.  Disabling.
  warnings.warn("torch.cuda.amp.GradScaler is enabled, but CUDA is no
t available.  Disabling.")
Epoch 1/5 [Train]:   0%|                                          | 0/32 [00:0
0<?, ?it/s]/Users/shetty/anaconda3/anaconda3/lib/python3.11/site-pack
ages/torch/amp/autocast_mode.py:204: UserWarning: User provided devic
e_type of 'cuda', but CUDA is not available. Disabling
  warnings.warn('User provided device_type of \'cuda\', but CUDA is n
ot available. Disabling')
Epoch 1/5 [Train]: 100%|██████| 32/32 [00:39<00:00,  1.23s/it, train_l
oss=0.4575]
Epoch 1/5 [Val]: 100%|████████████████████████████| 7/7 [00:02<00:00,
2.99it/s]

Epoch 1/5: Train Loss = 0.5846, Validation Loss = 0.5177
--------------------------------------------------

Epoch 2/5 [Train]: 100%|██████| 32/32 [00:37<00:00,  1.19s/it, train_l
oss=0.5575]
Epoch 2/5 [Val]: 100%|████████████████████████████| 7/7 [00:02<00:00,
2.89it/s]

Epoch 2/5: Train Loss = 0.4945, Validation Loss = 0.4851
--------------------------------------------------

Epoch 3/5 [Train]: 100%|██████| 32/32 [00:37<00:00,  1.18s/it, train_l
oss=0.4446]
Epoch 3/5 [Val]: 100%|████████████████████████████| 7/7 [00:02<00:00,
2.91it/s]

Epoch 3/5: Train Loss = 0.4692, Validation Loss = 0.4320
--------------------------------------------------

Epoch 4/5 [Train]: 100%|██████| 32/32 [00:38<00:00,  1.19s/it, train_l
oss=0.4061]
Epoch 4/5 [Val]: 100%|████████████████████████████| 7/7 [00:02<00:00,
2.98it/s]

Epoch 4/5: Train Loss = 0.4256, Validation Loss = 0.3931
--------------------------------------------------

Epoch 5/5 [Train]: 100%|██████| 32/32 [00:38<00:00,  1.19s/it, train_l
oss=0.4067]
Epoch 5/5 [Val]: 100%|████████████████████████████| 7/7 [00:02<00:00,
2.85it/s]
```

```
Epoch 5/5: Train Loss = 0.3824, Validation Loss = 0.3659
--------------------------------------------------
Training completed!
Model saved!
```



	precision	recall	f1-score	support
anger	0.65	0.82	0.73	34
anticipation	0.00	0.00	0.00	15
disgust	0.70	0.91	0.79	34
fear	0.00	0.00	0.00	21
joy	0.83	0.76	0.79	45
love	0.00	0.00	0.00	14
optimism	0.73	0.62	0.67	39
pessimism	0.00	0.00	0.00	13
sadness	0.33	0.32	0.33	22
surprise	0.00	0.00	0.00	6
trust	0.00	0.00	0.00	5
micro avg	0.68	0.50	0.58	248
macro avg	0.30	0.31	0.30	248
weighted avg	0.48	0.50	0.49	248
samples avg	0.57	0.49	0.51	248

In [ ]: