Listing 6.4 Setting up a Pytorch `Dataset` class

```
import torch
from torch.utils.data import Dataset

class SpamDataset(Dataset):
    def __init__(self, csv_file, tokenizer, max_length=None,
                 pad_token_id=50256):
        self.data = pd.read_csv(csv_file)
                                                    <-- Pretokenizes texts
        self.encoded_texts = [
            tokenizer.encode(text) for text in self.data["Text"]
        ]

        if max_length is None:
            self.max_length = self._longest_encoded_length()
        else:
            self.max_length = max_length
                                                    <-- Truncates sequences if they are longer than max_length
            self.encoded_texts = [
                encoded_text[:self.max_length]
                for encoded_text in self.encoded_texts
            ]

                                                    <-- Pads sequences to the longest sequence
        self.encoded_texts = [
            encoded_text + [pad_token_id] *
            (self.max_length - len(encoded_text))
            for encoded_text in self.encoded_texts
        ]


    def __getitem__(self, index):
        encoded = self.encoded_texts[index]
        label = self.data.iloc[index]["Label"]
        return (
            torch.tensor(encoded, dtype=torch.long),
            torch.tensor(label, dtype=torch.long)
        )

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

    def _longest_encoded_length(self):
        max_length = 0
        for encoded_text in self.encoded_texts:
            encoded_length = len(encoded_text)
            if encoded_length > max_length:
                max_length = encoded_length
        return max_length
```

The `SpamDataset` class loads data from the CSV files we created earlier, tokenizes the text using the GPT-2 tokenizer from `tiktoken`, and allows us to *pad* or *truncate* the sequences to a uniform length determined by either the longest sequence or a predefined maximum length. This ensures each input tensor is of the same size, which is necessary to create the batches in the training data loader we implement next:

```
train_dataset = SpamDataset(
    csv_file="train.csv",
    max_length=None,
    tokenizer=tokenizer
)
```

The longest sequence length is stored in the dataset's `max_length` attribute. If you are curious to see the number of tokens in the longest sequence, you can use the following code:

```
print(train_dataset.max_length)
```

The code outputs `120`, showing that the longest sequence contains no more than 120 tokens, a common length for text messages. The model can handle sequences of up to 1,024 tokens, given its context length limit. If your dataset includes longer texts, you can pass `max_length=1024` when creating the training dataset in the preceding code to ensure that the data does not exceed the model's supported input (context) length.

Next, we pad the validation and test sets to match the length of the longest training sequence. Importantly, any validation and test set samples exceeding the length of the longest training example are truncated using `encoded_text[:self.max_length]` in the `SpamDataset` code we defined earlier. This truncation is optional; you can set `max_length=None` for both validation and test sets, provided there are no sequences exceeding 1,024 tokens in these sets:

```
val_dataset = SpamDataset(
    csv_file="validation.csv",
    max_length=train_dataset.max_length,
    tokenizer=tokenizer
)
test_dataset = SpamDataset(
    csv_file="test.csv",
    max_length=train_dataset.max_length,
    tokenizer=tokenizer
)
```

Exercise 6.1 Increasing the context length

Pad the inputs to the maximum number of tokens the model supports and observe how it affects the predictive performance.

Using the datasets as inputs, we can instantiate the data loaders similarly to when we were working with text data. However, in this case, the targets represent class labels rather than the next tokens in the text. For instance, if we choose a batch size of 8, each batch will consist of eight training examples of length 120 and the corresponding class label of each example, as illustrated in figure 6.7.

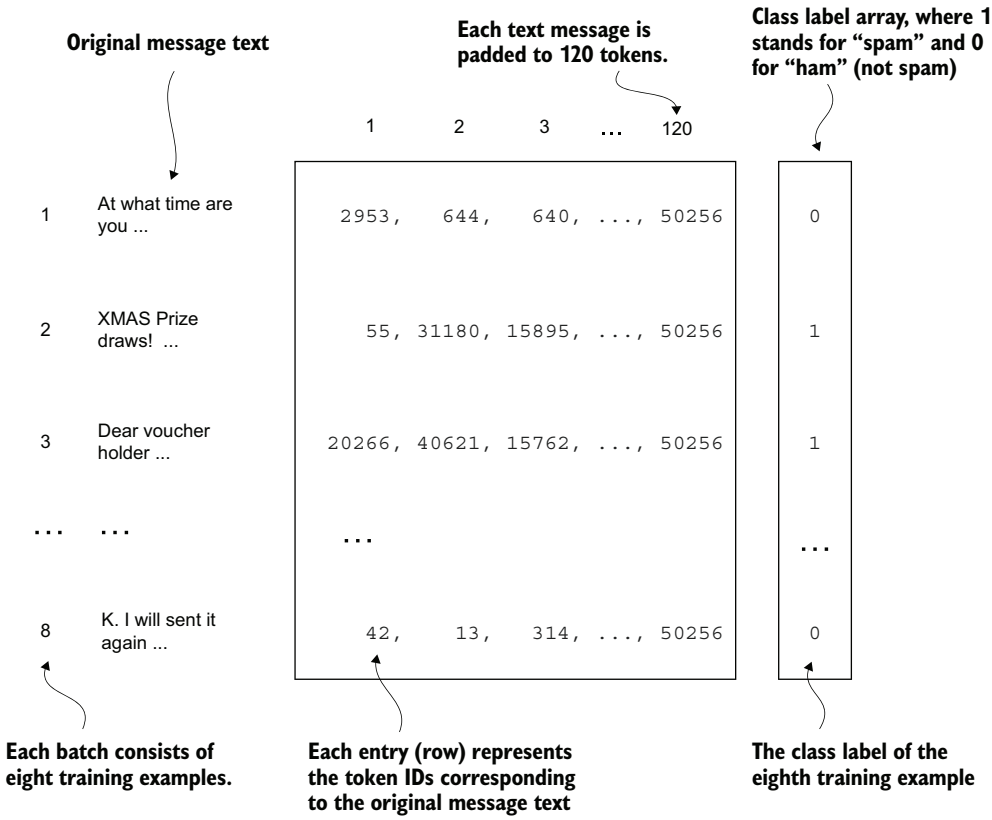

Figure 6.7 A single training batch consisting of eight text messages represented as token IDs. Each text message consists of 120 token IDs. A class label array stores the eight class labels corresponding to the text messages, which can be either `0` ("not spam") or `1` ("spam").