The code in the following listing creates the training, validation, and test set data loaders that load the text messages and labels in batches of size 8.

Listing 6.5 Creating PyTorch data loaders

```
from torch.utils.data import DataLoader

num_workers = 0                    <-- This setting ensures compatibility
batch_size = 8                         with most computers.
torch.manual_seed(123)

train_loader = DataLoader(
    dataset=train_dataset,
    batch_size=batch_size,
    shuffle=True,
    num_workers=num_workers,
    drop_last=True,
)
val_loader = DataLoader(
    dataset=val_dataset,
    batch_size=batch_size,
    num_workers=num_workers,
    drop_last=False,
)
test_loader = DataLoader(
    dataset=test_dataset,
    batch_size=batch_size,
    num_workers=num_workers,
    drop_last=False,
)
```

To ensure that the data loaders are working and are, indeed, returning batches of the expected size, we iterate over the training loader and then print the tensor dimensions of the last batch:

```
for input_batch, target_batch in train_loader:
    pass
print("Input batch dimensions:", input_batch.shape)
print("Label batch dimensions", target_batch.shape)
```

The output is

```
Input batch dimensions: torch.Size([8, 120])
Label batch dimensions torch.Size([8])
```

As we can see, the input batches consist of eight training examples with 120 tokens each, as expected. The label tensor stores the class labels corresponding to the eight training examples.

Lastly, to get an idea of the dataset size, let's print the total number of batches in each dataset:

```
print(f"{len(train_loader)} training batches")
print(f"{len(val_loader)} validation batches")
print(f"{len(test_loader)} test batches")
```

The number of batches in each dataset are

```
130 training batches
19 validation batches
38 test batches
```

Now that we've prepared the data, we need to prepare the model for fine-tuning.

6.4 Initializing a model with pretrained weights

We must prepare the model for classification fine-tuning to identify spam messages. We start by initializing our pretrained model, as highlighted in figure 6.8.

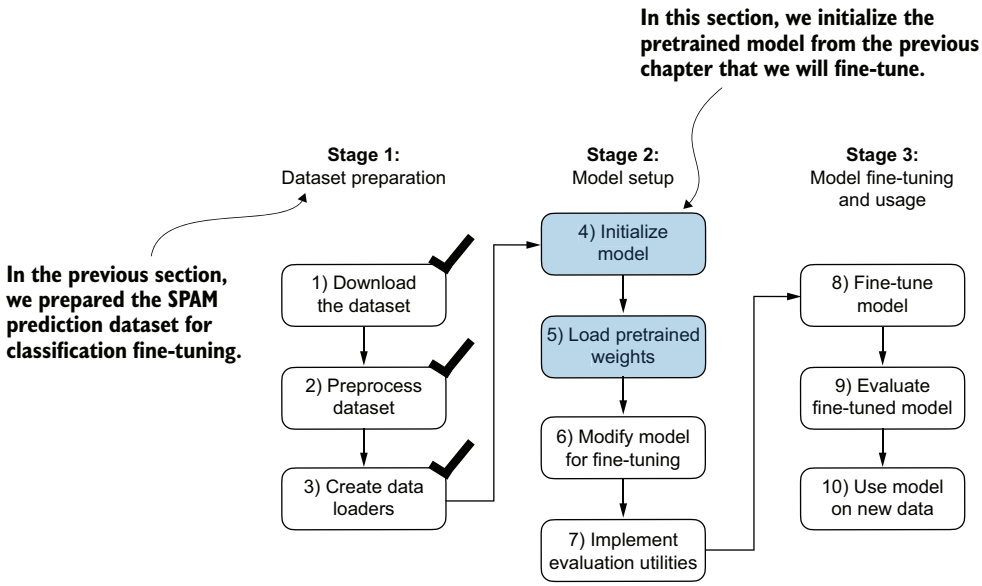

Figure 6.8 The three-stage process for classification fine-tuning the LLM. Having completed stage 1, preparing the dataset, we now must initialize the LLM, which we will then fine-tune to classify spam messages.

To begin the model preparation process, we employ the same configurations we used to pretrain unlabeled data:

```
CHOOSE_MODEL = "gpt2-small (124M)"
INPUT_PROMPT = "Every effort moves"
```

```
BASE_CONFIG = {
    "vocab_size": 50257,          # Vocabulary size
    "context_length": 1024,       # Context length
    "drop_rate": 0.0,             # Dropout rate
    "qkv_bias": True              # Query-key-value bias
}
model_configs = {
    "gpt2-small (124M)": {"emb_dim": 768, "n_layers": 12, "n_heads": 12},
    "gpt2-medium (355M)": {"emb_dim": 1024, "n_layers": 24, "n_heads": 16},
    "gpt2-large (774M)": {"emb_dim": 1280, "n_layers": 36, "n_heads": 20},
    "gpt2-xl (1558M)": {"emb_dim": 1600, "n_layers": 48, "n_heads": 25},
}
BASE_CONFIG.update(model_configs[CHOOSE_MODEL])
```

Next, we import the `download_and_load_gpt2` function from the `gpt_download.py` file and reuse the `GPTModel` class and `load_weights_into_gpt` function from pretraining (see chapter 5) to load the downloaded weights into the GPT model.

Listing 6.6 Loading a pretrained GPT model

```
from gpt_download import download_and_load_gpt2
from chapter05 import GPTModel, load_weights_into_gpt

model_size = CHOOSE_MODEL.split(" ")[-1].lstrip("(").rstrip(")")
settings, params = download_and_load_gpt2(
    model_size=model_size, models_dir="gpt2"
)

model = GPTModel(BASE_CONFIG)
load_weights_into_gpt(model, params)
model.eval()
```

After loading the model weights into the `GPTModel`, we reuse the text generation utility function from chapters 4 and 5 to ensure that the model generates coherent text:

```
from chapter04 import generate_text_simple
from chapter05 import text_to_token_ids, token_ids_to_text

text_1 = "Every effort moves you"
token_ids = generate_text_simple(
    model=model,
    idx=text_to_token_ids(text_1, tokenizer),
    max_new_tokens=15,
    context_size=BASE_CONFIG["context_length"]
)
print(token_ids_to_text(token_ids, tokenizer))
```

The following output shows the model generates coherent text, which is indicates that the model weights have been loaded correctly:

```
Every effort moves you forward.
The first step is to understand the importance of your work
```