```
validation_df.to_csv("validation.csv", index=None)
test_df.to_csv("test.csv", index=None)
```

Next, we create the `SpamDataset` instances.

Listing E.2 Instantiating PyTorch datasets

```
import torch
from torch.utils.data import Dataset
import tiktoken
from chapter06 import SpamDataset

tokenizer = tiktoken.get_encoding("gpt2")
train_dataset = SpamDataset("train.csv", max_length=None,
    tokenizer=tokenizer
)
val_dataset = SpamDataset("validation.csv",
    max_length=train_dataset.max_length, tokenizer=tokenizer
)
test_dataset = SpamDataset(
    "test.csv", max_length=train_dataset.max_length, tokenizer=tokenizer
)
```

After creating the PyTorch dataset objects, we instantiate the data loaders.

Listing E.3 Creating PyTorch data loaders

```
from torch.utils.data import DataLoader

num_workers = 0
batch_size = 8

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,
)
```

As a verification step, we iterate through the data loaders and check that the batches contain eight training examples each, where each training example consists of 120 tokens:

```
print("Train loader:")
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

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

Lastly, we 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")
```

In this case, we have the following number of batches per dataset:

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

E.3 Initializing the model

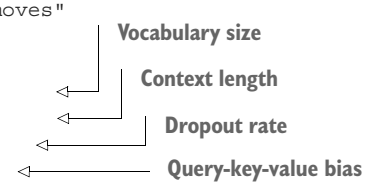
We repeat the code from chapter 6 to load and prepare the pretrained GPT model. We begin by downloading the model weights and loading them into the `GPTModel` class.

Listing E.4 Loading a pretrained GPT model

```
from gpt_download import download_and_load_gpt2
from chapter04 import GPTModel
from chapter05 import load_weights_into_gpt


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

BASE_CONFIG = {
    "vocab_size": 50257,
    "context_length": 1024,
    "drop_rate": 0.0,
    "qkv_bias": True
}
```



```
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])

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()
```

To ensure that the model was loaded corrected, let's double-check that it 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 that the model generates coherent text, which is an indicator that the model weights are loaded correctly:

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

Next, we prepare the model for classification fine-tuning, similar to chapter 6, where we replace the output layer:

```
torch.manual_seed(123)
num_classes = 2
model.out_head = torch.nn.Linear(in_features=768, out_features=num_classes)
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)
```

Lastly, we calculate the initial classification accuracy of the not-fine-tuned model (we expect this to be around 50%, which means that the model is not able to distinguish between spam and nonspam messages yet reliably):