https://colab.research.google.com/github/google-gemini/cookbook/blob/main/templates/aistudio_gemini_prompt_freeform.ipynb

```
# import necessary modules.
import base64
import copy
import json
import pathlib
import requests


import PIL.Image
import IPython.display
from IPython.display import Markdown

try:
    # The SDK will automatically read it from the GOOGLE_API_KEY
environment variable.
    # In Colab get the key from Colab-secrets ("🔑" in the left panel).
    import os
    from google.colab import userdata

    os.environ["GOOGLE_API_KEY"] = userdata.get("GOOGLE_API_KEY")
except ImportError:
    pass

import google.generativeai as genai

# Parse the arguments

model = 'gemini-2.0-flash-exp' # @param {isTemplate: true}
contents_b64 =
'W3sicGFydHMiOlt7InRleHQiOiJpbXBvcnQgdG9yY2hcXG5pbXBvcnQgdG9yY2gubm4gYXMgb
m5cXG5pbXBvcnQgdG9yY2gub3B0aW0gYXMgb3B0aW1cXG5mcm9tIHRvcmNodmlzaW9uIGltcG9
ydCBkYXRhc2V0cywgdHJhbnNmb3Jtc1xcbmZyb20gdG9yY2gudXRpbHMuZGF0YSBpbXBvcnQgR
GF0YUxvYWRlclxcbmZyb20gc2tsZWFybi5tZXRyaWNzIGltcG9ydCBjbGFzc2lmaWNhdGlvbl9
yZXBvcnRcXG5pbXBvcnQgbWF0cGxvdGxpYi5weXBsb3QgYXMgcGx0XFxuIyBEZXZpY2UgY
29uZmlndXJhdGlvblxcbmRldmljZSA9IHRvcmNoLmRldmljZSgnY3VkYScgaWYgdG9yY2guY3V
kYS5pc19hdmFpbGFibGUoKSBlbHNlICdjcHUnKVxcblxcbiMgSHlwZXJwYXJhbWV0ZXJzFxub
nVtX2Vwb2NocyA9IDNcXG5iYXRjaF9zaXplID0gNjRcXG5sZWFybmluZ19yYXRlID0gMC4wMVx
```

cblxcbiMgRGF0YSBhdWdtZW50YXRpb24gYW5kIG5vcm1hbGl6YXRpb25cXG50cmFuc2Zvcm0gP
SB0cmFuc2Zvcm1zLkNvbXBvc2UoW1xcbiAgICB0cmFuc2Zvcm1zLlJhbmRvbUhvcml6b250YWx
GbGlwKCksXFxuICAgIHRyYW5zZm9ybXMuUmFuZG9tUm90YXRpb24oMTApLFxcbiAgICB0cmFuc
2Zvcm1zLlRvVGVuc29yKCksXFxuICAgIHRyYW5zZm9ybXMuTm9ybWFsaXplKCgwLjUsKSwgKDA
uNSwpKSAgIyBOb3JtYWxpemUgdG8gWy0xLCAxXVxcbl0pXFxuXFxuIyBMb2FkIHRoZSBGYXNoa
W9uLU1OSVNUIERhdGFzZXRcXG50cmFpbl9kYXRhc2V0ID0gZGF0YXNldHMuRmFzaGlvbk1OSVN
UKHJvb3Q9Jy4vZGF0YScsXFxuICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgI
CB0cmFpbj1UcnVlLFxcbiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgdHJ
hbnNmb3JtPXRyYW5zZm9ybSxcXG4gICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgI
CAgIGRvd25sb2FkPVRydWUpXFxuXFxudGVzdF9kYXRhc2V0ID0gZGF0YXNldHMuRmFzaGlvbk1
OSVNUKHJvb3Q9Jy4vZGF0YScsXFxuICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgI
CAgIHRyYWluPUZhbHNlLFxcbiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICB
0cmFuc2Zvcm09dHJhbnNmb3Jtcy5Db21wb3NlKFtcXG4gICAgICAgICAgICAgICAgICAgICAgI
CAgICAgICAgICAgICAgIHRyYW5zZm9ybXMuVG9UZW5zb3IoKSxcXG4gICAgICAgICAgICAgICA
gICAgICAgICAgICAgICAgICAgICAgIHRyYW5zZm9ybXMuTm9ybWFsaXplKCgwLjUsK
SwgKDAuNSwpKVxcbiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICBdKSxcXG4
gICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgZG93bmxvYWQ9VHJ1ZSlcXG5cX
G4jIERhdGEgbG9hZGVyc1xcbnRyYWluX2xvYWRlciA9IERhdGFMb2FkZXIoZGF0YXNldD10cmF
pbl9kYXRhc2V0LFxcbiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgYmF0Y2hfc2l6ZT1iYXRja
F9zaXplLFxcbiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgc2h1ZmZsZT1UcnVlKVxcblxcbnR
lc3RfbG9hZGVyID0gRGF0YUxvYWRlcihkYXRhc2V0PXRlc3RfZGF0YXNldCxcXG4gICAgICAgI
CAgICAgICAgICAgICAgICAgYmF0Y2hfc2l6ZT1iYXRjaF9zaXplLFxcbiAgICAgICAgICAgICA
gICAgICAgICAgICBzaHVmZmxlPUZhbHNlKVxcblxcbiMgRGVmaW5lIHRoZSBDTk4gTW9kZWxcX
G5jbGFzcyBDTk5Nb2RlbChubi5Nb2R1bGUpOlxcbiAgICBkZWYgX19pbml0X18oc2VsZik6XFx
uICAgICAgICBzdXBlcihDTk5Nb2RlbCwgc2VsZikuX19pbml0X18oKVxcbiAgICAgICAgc2VsZ
i5jb252MSA9IG5uLkNvbnYyZCgxLCA4LCA1KVxcbiAgICAgICAgc2VsZi5ibjEgPSBubi5CYXR
jaE5vcm0yZCg4KSAgIyBCYXRjaCBOb3JtYWxpemF0aW9uXFxuICAgICAgICBzZWxmLnBvb2wgP
SBubi5NYXhQb29sMmQoMiwgMilcXG4gICAgICAgIHNlbGYuY29udjIgPSBubi5Db252MmQoOCw
gMTYsIDUpXFxuICAgICAgICBzZWxmLmJuMiA9IG5uLkJhdGNoTm9ybTJkKDE2KSAgIyBCYXRja
CBOb3JtYWxpemF0aW9uXFxuICAgICAgICBzZWxmLmRyb3BvdXQgPSBubi5Ecm9wb3V0KDAuMjU
pICAjIERyb3BvdXQgbGF5ZXJcXG4gICAgICAgIHNlbGYuZmMxID0gbm4uTGluZWFyKDE2ICogN
CAqIDQsIDEyMClcXG4gICAgICAgIHNlbGYuZmMyID0gbm4uTGluZWFyKDEyMCwgODQpXFxuICCA
gICAgICBzZWxmLmZjMyA9IG5uLkxpbmVhcig4NCwgMTApXFxuXFxuICAgIGRlZiBmb3J3YXJkK
HNlbGYsIHgpOlxcbiAgICAgICAgeCA9IHNlbGYucG9vbChGLnJlbHUoc2VsZi5ibjEoc2VsZi5
jb252MSh4KSkpKVxcbiAgICAgICAgeCA9IHNlbGYucG9vbChGLnJlbHUoc2VsZi5ibjIoc2VsZ
i5jb252Mih4KSkpKVxcbiAgICAgICAgeCA9IHgudmlldygtMSwgMTYgKiA0ICogNClcXG4gICA
gICAgIHggPSBGLnJlbHUoc2VsZi5mYzEoeCkpXFxuICAgICAgICB4ID0gc2VsZi5kcm9wb3V0K
HgpICAjIEFwcGx5IGRyb3BvdXRcXG4gICAgICAgIHggPSBGLnJlbHUoc2VsZi5mYzIoeCkpXFx
uICAgICAgICB4ID0gc2VsZi5mYzMoeClcXG4gICAgICAgIHJldHVybiB4XFxuXFxuIyBJbml0a
WFsaXplIHRoZSBtb2RlbCwgbG9zcyBmdW5jdGlvbiwgYW5kIG9wdGltaXplclxcbm1vZGVsID0

gQ05OTW9kZWwoKS50byhkZXZpY2UpXFxuY3JpdGVyaW9uID0gbm4uQ3Jvc3NFbnRyb3B5TG9zc
ygpXFxub3B0aW1pemVyID0gb3B0aW0uQWRhbShtb2RlbC5wYXJhbWV0ZXJzKCksIGxyPWxlYXJ
uaW5nX3JhdGUpXFxuc2NoZWR1bGVyID0gb3B0aW0ubHJfc2NoZWR1bGVyLlN0ZXBMUihvcHRpb
Wl6ZXIsIHN0ZXBfc2l6ZT0xLCBnYW1tYT0wLjcpXFxuXFxuIyBUcmFpbmluZyBmdW5jdGlvblx
cbmRlZiB0cmFpbihtb2RlbCwgZGV2aWNlLCB0cmFpbl9sb2FkZXIsIG9wdGltaXplciwgY3Jpd
GVyaW9uKTpcXG4gICAgbW9kZWwudHJhaW4oKVxcbiAgICBlcG9jaF9sb3NzID0gMFxcbiAgICB
mb3IgYmF0Y2hfaWR4LCAoZGF0YSwgdGFyZ2V0KSBpbiBlbnVtZXJhdGUodHJhaW5fbG9hZGVyK
TpcXG4gICAgICAgIGRhdGEsIHRhcmdldCA9IGRhdGEudG8oZGV2aWNlKSwgdGFyZ2V0LnRvKGR
ldmljZSlcXG4gICAgICAgIG9wdGltaXplci56ZXJvX2dyYWQoKVxcbiAgICAgICAgb3V0cHV0I
D0gbW9kZWwoZGF0YSlcXG4gICAgICAgIGxvc3MgPSBjcml0ZXJpb24ob3V0cHV0LCB0YXJnZXQ
pXFxuICAgICAgICBsb3NzLmJhY2t3YXJkKClcXG4gICAgICAgIG9wdGltaXplci5zdGVwKClcX
G4gICAgICAgIGVwb2NoX2xvc3MgKz0gbG9zcy5pdGVtKClcXG4gICAgcmV0dXJuIGVwb2NoX2x
vc3MgLyBsZW4odHJhaW5fbG9hZGVyKVxcblxcbiMgVGVzdGluZyBmdW5jdGlvblxcbmRlZiB0Z
XN0KG1vZGVsLCBkZXZpY2UsIHRlc3RfbG9hZGVyLCBjcml0ZXJpb24pOlxcbiAgICBtb2RlbC5
ldmFsKClcXG4gICAgdGVzdF9sb3NzID0gMFxcbiAgICBjb3JyZWN0ID0gMFxcbiAgICBhbGxfc
HJlZHMgPSBbXVxcbiAgICBhbGxfdGFyZ2V0cyA9IFtdXFxuICAgIHdpdGggdG9yY2gubm9fZ3J
hZCgpOlxcbiAgICAgICAgZm9yIGRhdGEsIHRhcmdldCBpbiB0ZXN0X2xvYWRlcjpcXG4gICAgI
CAgICAgICBkYXRhLCB0YXJnZXQgPSBkYXRhLnRvKGRldmljZSksIHRhcmdldC50byhkZXZpY2U
pXFxuICAgICAgICAgICAgb3V0cHV0ID0gbW9kZWwoZGF0YSlcXG4gICAgICAgICAgICB0ZXN0X
2xvc3MgKz0gY3JpdGVyaW9uKG91dHB1dCwgdGFyZ2V0KS5pdGVtKCkgKiBkYXRhLnNpemUoMCl
cXG4gICAgICAgICAgICBwcmVkID0gb3V0cHV0LmFyZ21heChkaW09MSwga2VlcGRpbT1UcnVlK
VxcbiAgICAgICAgICAgIGNvcnJlY3QgKz0gcHJlZC5lcSh0YXJnZXQudmlld19hcyhwcmVkKSk
uc3VtKCkuaXRlbSgpXFxuICAgICAgICAgICAgYWxsX3ByZWRzLmV4dGVuZChwcmVkLmNwdSgpL
m51bXB5KCkpXFxuICAgICAgICAgICAgYWxsX3RhcmdldHMuZXh0ZW5kKHRhcmdldC5jcHUoKS5
udW1weSgpKVxcblxcbiAgICB0ZXN0X2xvc3MgLz0gbGVuKHRlc3RfbG9hZGVyLmRhdGFzZXQpX
FxuICAgIGFjY3VyYWN5ID0gMTAwLiAqIGNvcnJlY3QgLyBsZW4odGVzdF9sb2FkZXIuZGF0YXN
ldClcXG4gICAgcHJpbnQoZidcXFxcblRlc3Qgc2V0OiBBdmVyYWdlIGxvc3M6IHt0ZXN0X2xvc
3M6LjRmfSwgQWNjdXJhY3k6IHtjb3JyZWN0fS97bGVuKHRlc3RfbG9hZGVyLmRhdGFzZXQpfSA
oe2FjY3VyYWN5Oi4wZn0lKVxcXFxuJylcXG5cXG4gICAgcmV0dXJuIHRlc3RfbG9zcywgYWNjd
XJhY3ksIGFsbF9wcmVkcywgYWxsX3RhcmdldHNcXG5cXG4jIFRyYWluaW5nIGxvb3Agd2l0aCB
2aXN1YWxpemF0aW9uXFxudHJhaW5fbG9zc2VzID0gW11cXG50ZXN0X2xvc3NlcyA9IFtdXFxuY
WNjdXJhY2llcyA9IFtdXFxuXFxuZm9yIGVwb2NoIGluIHJhbmdlKDEsIG51bV9lcG9jaHMgKyA
xKTpcXG4gICAgcHJpbnQoZlxcXCJFcG9jaCB7ZXBvY2h9L3tudW1fZXBvY2hzfVxcXCIpXFxuI
CAgIHRyYWluX2xvc3MgPSB0cmFpbihtb2RlbCwgZGV2aWNlLCB0cmFpbl9sb2FkZXIsIG9wdGl
taXplciwgY3JpdGVyaW9uKVxcbiAgICB0ZXN0X2xvc3MsIGFjY3VyYWN5LCBhbGxfcHJlZHMsI
GFsbF90YXJnZXRzID0gdGVzdChtb2RlbCwgZGV2aWNlLCB0ZXN0X2xvYWRlciwgY3JpdGVyaW9
uKVxcbiAgICB0cmFpbl9sb3NzZXMuYXBwZW5kKHRyYWluX2xvc3MpXFxuICAgIHRlc3RfbG9zc
2VzLmFwcGVuZCh0ZXN0X2xvc3MpXFxuICAgIGFjY3VyYWNpZXMuYXBwZW5kKGFjY3VyYWN5KVx
cbiAgICBzY2hlZHVsZXIuc3RlcCgpXFxuXFxuIyBDbGFzc2lmaWNhdGlvbiByZXBvcnRcXG5wc
mludChcXFwiXFxcXG5DbGFzc2lmaWNhdGlvbiBSZXBvcnQ6XFxcIilcXG5wcmludChjbGFzc2l

```
maWNhdGlvbl9yZXBvcnQoYWxsX3RhcmdldHMsIGFsbF9wcmVkcywgdGFyZ2V0X25hbWVzPXRyY
WluX2RhdGFzZXQuY2xhc3NlcykpXFxuXFxuIyBQbG90IHRyYWluaW5nIGxvc3NcXG5wbHQuZml
ndXJlKGZpZ3NpemU9KDEwLCA1KSlcXG5wbHQucGxvdCh0cmFpbl9sb3NzZXMsIGxhYmVsPSdUc
mFpbmluZyBMb3NzJylcXG5wbHQucGxvdCh0ZXN0X2xvc3NlcywgbGFiZWw9J1Rlc3QgTG9zcyc
pXFxucGx0LnhsYWJlbCgnRXBvY2hzJylcXG5wbHQueWxhYmVsKCdMb3NzJylcXG5wbHQudGl0b
GUoJ1RyYWluaW5nIGFuZCBUZXN0IExvc3MnKVxcbnBsdC5sZWdlbmQoKVxcbnBsdC5zaG93KCkl
cXG5cXG4jIFNhdmUgdGhlIG1vZGVsXFxudG9yY2guc2F2ZShtb2RlbCwgJ2Zhc2hpb25fbW5pc
3RfY25uX2Z1bGwucHRoJylcXG5wcmludChcXFwiTW9kZWwgc2F2ZWQgYXMgJ2Zhc2hpb25fbW5
pc3RfY25uX2Z1bGwucHRoJy5cXFwiKSJ9LHsidGV4dCI6ImlucHV0OiAifSx7InRleHQiOiJvd
XRwdXQ6ICJ9XX1d' # @param {isTemplate: true}
generation_config_b64 =
'eyJ0ZW1wZXJhdHVyZSI6MSwidG9wX3AiOjAuOTUsInRvcF9rIjo0MCwibWF4X291dHB1dF90b
2tlbnMiOjgxOTJ9' # @param {isTemplate: true}
safety_settings_b64 = "e30="  # @param {isTemplate: true}

gais_contents = json.loads(base64.b64decode(contents_b64))

generation_config = json.loads(base64.b64decode(generation_config_b64))
safety_settings = json.loads(base64.b64decode(safety_settings_b64))

stream = False

# Convert and upload the files

tempfiles = pathlib.Path(f"tempfiles")
tempfiles.mkdir(parents=True, exist_ok=True)


drive = None
def upload_file_data(file_data, index):
    """Upload files to the Files API.

    For each file, Google AI Studio either sent:
    - a Google Drive ID,
    - a URL,
    - a file path, or
    - The raw bytes (`inline_data`).

    The API only understands `inline_data` or it's Files API.
```

```
    This code, uploads files to the files API where the API can access
them.
    """

    mime_type = file_data["mime_type"]
    if drive_id := file_data.pop("drive_id", None):
        if drive is None:
          from google.colab import drive
          drive.mount("/gdrive")

        path = next(

pathlib.Path(f"/gdrive/.shortcut-targets-by-id/{drive_id}").glob("*")
        )
        print("Uploading:", str(path))
        file_info = genai.upload_file(path=path, mime_type=mime_type)
        file_data["file_uri"] = file_info.uri
        return

    if url := file_data.pop("url", None):
        response = requests.get(url)
        data = response.content
        name = url.split("/")[-1]
        path = tempfiles / str(index)
        path.write_bytes(data)
        print("Uploading:", url)
        file_info = genai.upload_file(path, display_name=name,
mime_type=mime_type)
        file_data["file_uri"] = file_info.uri
        return

    if name := file_data.get("filename", None):
        if not pathlib.Path(name).exists():
            raise IOError(
                f"local file: `{name}` does not exist. You can upload
files "
                'to Colab using the file manager ("📁 Files" in the left '
                "toolbar)"
            )
```

```
        file_info = genai.upload_file(path, display_name=name,
mime_type=mime_type)
        file_data["file_uri"] = file_info.uri
        return

    if "inline_data" in file_data:
        return

    raise ValueError("Either `drive_id`, `url` or `inline_data` must be
provided.")


contents = copy.deepcopy(gais_contents)

index = 0
for content in contents:
    for n, part in enumerate(content["parts"]):
        if file_data := part.get("file_data", None):
            upload_file_data(file_data, index)
            index += 1

import json
print(json.dumps(contents, indent=4))
```

```
import torch
import torch.nn as nn
import torch.nn.functional as F # Import F
import torch.optim as optim
from torchvision import datasets, transforms
from torch.utils.data import DataLoader
from sklearn.metrics import classification_report
import matplotlib.pyplot as plt

# Device configuration
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

# Hyperparameters
num_epochs = 7
batch_size = 64
learning_rate = 0.03

# Data augmentation and normalization
transform = transforms.Compose([
    transforms.RandomHorizontalFlip(),
    transforms.RandomRotation(10),
    transforms.ToTensor(),
    transforms.Normalize((0.5,), (0.5,))  # Normalize to [-1, 1]
])

# Load the Fashion-MNIST Dataset
train_dataset = datasets.FashionMNIST(root='./data',
                                      train=True,
                                      transform=transform,
                                      download=True)

test_dataset = datasets.FashionMNIST(root='./data',
                                     train=False,
                                     transform=transforms.Compose([
                                         transforms.ToTensor(),
                                         transforms.Normalize((0.5,),
(0.5,))
                                     ]),
                                     download=True)
```

```
# Data loaders
train_loader = DataLoader(dataset=train_dataset,
                          batch_size=batch_size,
                          shuffle=True)

test_loader = DataLoader(dataset=test_dataset,
                         batch_size=batch_size,
                         shuffle=False)

# Define the CNN Model
class CNNModel(nn.Module):
    def __init__(self):
        super(CNNModel, self).__init__()
        self.conv1 = nn.Conv2d(1, 8, 5)
        self.bn1 = nn.BatchNorm2d(8)  # Batch Normalization
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(8, 16, 5)
        self.bn2 = nn.BatchNorm2d(16)  # Batch Normalization
        self.dropout = nn.Dropout(0.25)  # Dropout layer
        self.fc1 = nn.Linear(16 * 4 * 4, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        x = self.pool(F.relu(self.bn1(self.conv1(x))))
        x = self.pool(F.relu(self.bn2(self.conv2(x))))
        x = x.view(-1, 16 * 4 * 4)
        x = F.relu(self.fc1(x))
        x = self.dropout(x)  # Apply dropout
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x

# Initialize the model, loss function, and optimizer
model = CNNModel().to(device)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=learning_rate)
scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=1, gamma=0.7)

# Training function
```

```python
def train(model, device, train_loader, optimizer, criterion):
    model.train()
    epoch_loss = 0
    for batch_idx, (data, target) in enumerate(train_loader):
        data, target = data.to(device), target.to(device)
        optimizer.zero_grad()
        output = model(data)
        loss = criterion(output, target)
        loss.backward()
        optimizer.step()
        epoch_loss += loss.item()
    return epoch_loss / len(train_loader)

# Testing function
def test(model, device, test_loader, criterion):
    model.eval()
    test_loss = 0
    correct = 0
    all_preds = []
    all_targets = []
    with torch.no_grad():
        for data, target in test_loader:
            data, target = data.to(device), target.to(device)
            output = model(data)
            test_loss += criterion(output, target).item() * data.size(0)
            pred = output.argmax(dim=1, keepdim=True)
            correct += pred.eq(target.view_as(pred)).sum().item()
            all_preds.extend(pred.cpu().numpy())
            all_targets.extend(target.cpu().numpy())

    test_loss /= len(test_loader.dataset)
    accuracy = 100. * correct / len(test_loader.dataset)
    print(f'\nTest set: Average loss: {test_loss:.4f}, Accuracy:
{correct}/{len(test_loader.dataset)} ({accuracy:.0f}%)\n')

    return test_loss, accuracy, all_preds, all_targets

# Training loop with visualization
train_losses = []
test_losses = []
```

```
accuracies = []

for epoch in range(1, num_epochs + 1):
    print(f"Epoch {epoch}/{num_epochs}")
    train_loss = train(model, device, train_loader, optimizer, criterion)
    test_loss, accuracy, all_preds, all_targets = test(model, device,
test_loader, criterion)
    train_losses.append(train_loss)
    test_losses.append(test_loss)
    accuracies.append(accuracy)
    scheduler.step()

# Classification report
print("\nClassification Report:")
print(classification_report(all_targets, all_preds,
target_names=train_dataset.classes))

# Plot training loss
plt.figure(figsize=(10, 5))
plt.plot(train_losses, label='Training Loss')
plt.plot(test_losses, label='Test Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Training and Test Loss')
plt.legend()
plt.show()

# Save the model
torch.save(model, 'fashion_mnist_cnn_full.pth')
print("Model saved as 'fashion_mnist_cnn_full.pth'.")
```

Epoch 1/3

Test set: Average loss: 0.4532, Accuracy: 8378/10000 (84%)

Epoch 2/3

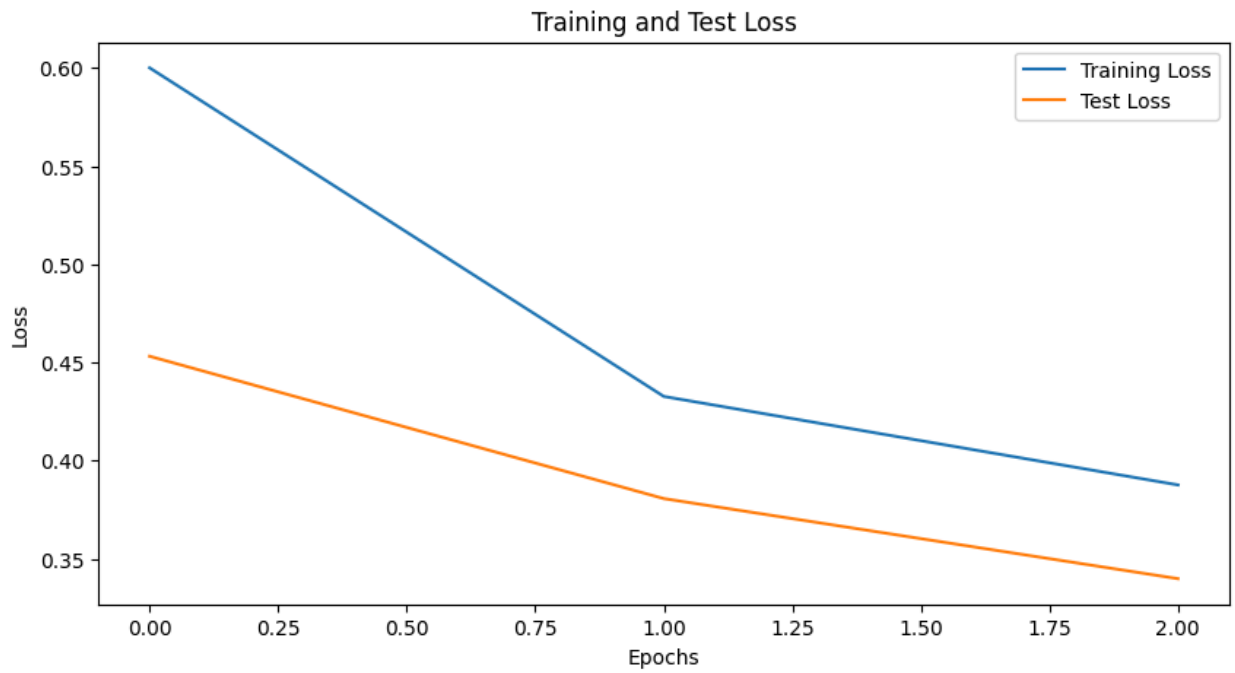
Test set: Average loss: 0.3807, Accuracy: 8594/10000 (86%)

Epoch 3/3

Test set: Average loss: 0.3399, Accuracy: 8760/10000 (88%)

Classification Report:

	precision	recall	f1-score	support
T-shirt/top	0.78	0.87	0.82	1000
Trouser	1.00	0.97	0.98	1000
Pullover	0.76	0.87	0.81	1000
Dress	0.87	0.91	0.89	1000
Coat	0.80	0.79	0.80	1000
Sandal	0.97	0.96	0.97	1000
Shirt	0.73	0.52	0.61	1000
Sneaker	0.92	0.96	0.94	1000
Bag	0.96	0.98	0.97	1000
Ankle boot	0.96	0.94	0.95	1000
accuracy			0.88	10000
macro avg	0.87	0.88	0.87	10000
weighted avg	0.87	0.88	0.87	10000



Model saved as 'fashion_mnist_cnn_full.pth'.

Epoch 1/5

Test set: Average loss: 0.5421, Accuracy: 8116/10000 (81%)

Epoch 2/5

Test set: Average loss: 0.4704, Accuracy: 8319/10000 (83%)

Epoch 3/5

Test set: Average loss: 0.4342, Accuracy: 8404/10000 (84%)

Epoch 4/5

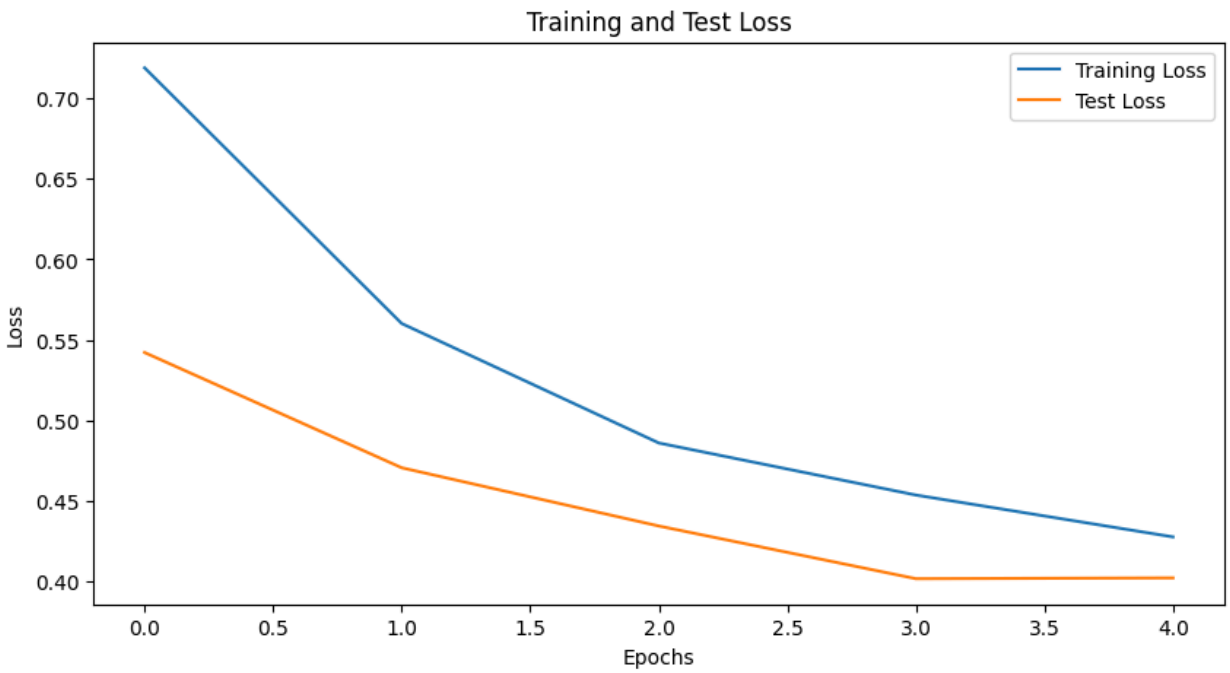
Test set: Average loss: 0.4015, Accuracy: 8504/10000 (85%)

Epoch 5/5

Test set: Average loss: 0.4019, Accuracy: 8487/10000 (85%)

Classification Report:

	precision	recall	f1-score	support
T-shirt/top	0.72	0.88	0.79	1000
Trouser	0.97	0.98	0.98	1000
Pullover	0.64	0.90	0.75	1000
Dress	0.91	0.86	0.88	1000
Coat	0.79	0.68	0.73	1000
Sandal	0.98	0.92	0.95	1000
Shirt	0.70	0.38	0.49	1000
Sneaker	0.90	0.98	0.94	1000
Bag	0.96	0.97	0.96	1000
Ankle boot	0.97	0.94	0.95	1000
accuracy			0.85	10000
macro avg	0.85	0.85	0.84	10000
weighted avg	0.85	0.85	0.84	10000



Model saved as 'fashion_mnist_cnn_full.pth'.

Epoch 1/7

Test set: Average loss: 0.5426, Accuracy: 8065/10000 (81%)

Epoch 2/7

Test set: Average loss: 0.5100, Accuracy: 8144/10000 (81%)

Epoch 3/7

Test set: Average loss: 0.4346, Accuracy: 8403/10000 (84%)

Epoch 4/7

Test set: Average loss: 0.4262, Accuracy: 8425/10000 (84%)

Epoch 5/7

Test set: Average loss: 0.4010, Accuracy: 8543/10000 (85%)

Epoch 6/7

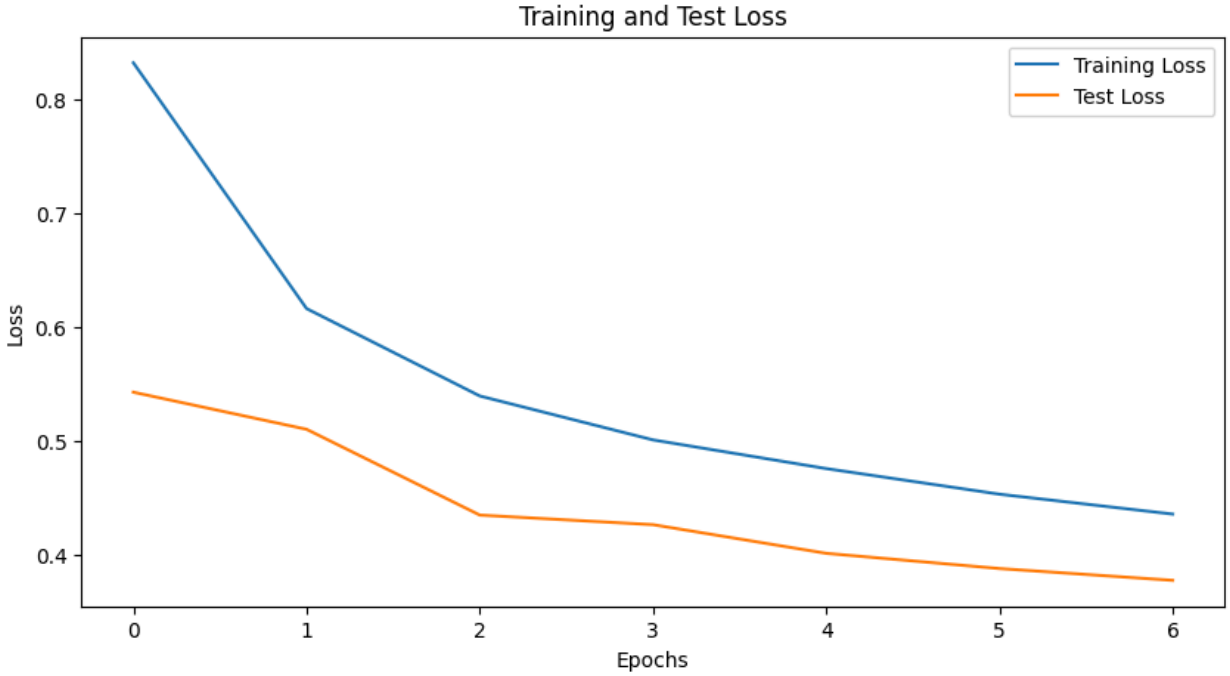
Test set: Average loss: 0.3877, Accuracy: 8581/10000 (86%)

Epoch 7/7

Test set: Average loss: 0.3773, Accuracy: 8603/10000 (86%)

Classification Report:

	precision	recall	f1-score	support
T-shirt/top	0.78	0.81	0.79	1000
Trouser	0.99	0.97	0.98	1000
Pullover	0.77	0.77	0.77	1000
Dress	0.85	0.90	0.88	1000
Coat	0.74	0.82	0.78	1000
Sandal	0.97	0.92	0.94	1000
Shirt	0.68	0.54	0.60	1000
Sneaker	0.90	0.96	0.93	1000
Bag	0.95	0.97	0.96	1000
Ankle boot	0.95	0.95	0.95	1000
accuracy			0.86	10000
macro avg	0.86	0.86	0.86	10000
weighted avg	0.86	0.86	0.86	10000



Model saved as 'fashion_mnist_cnn_full.pth'.