PROJECT: BUILDING AN E-COMMERCE CLOTHING CLASSIFIER MODEL

Fashion Forward is a new AI-based e-commerce clothing retailer. They want to use image classification to automatically categorize new product listings, making it easier for customers to find what they're looking for. It will also assist in inventory management by quickly sorting items.

As a data scientist tasked with implementing a garment classifier, your primary objective is to develop a machine learning model capable of accurately categorizing images of clothing items into distinct garment types such as shirts, trousers, shoes, etc.

```
# Run the cells below first
```

```
import numpy as np
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import Dataset, DataLoader
from torchmetrics import Accuracy, Precision, Recall
import torch.nn.functional as F
```

```
# Load datasets
from torchvision import datasets
import torchvision.transforms as transforms

train_data = datasets.FashionMNIST(root='./data', train=True, download=True,
transform=transforms.ToTensor())
test_data = datasets.FashionMNIST(root='./data', train=False, download=True,
transform=transforms.ToTensor())
```

```


# Récupérer les classes
classes = train_data.classes
num_classes = len(classes)

print(f"Classes trouvées : {classes}")
print(f"Nombre de classes : {num_classes}")
```

```
Classes trouvées : ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']
Nombre de classes : 10
```

```
import random
# Prendre une image au hasard
idx = random.randint(0, len(train_data) - 1)
sample_image, _ = train_data[idx]  # Tensor [1,28,28]
```

```python
class SimpleCNN(nn.Module):
    def __init__(self, example_input, num_classes=10, input_channels=1):
        """
        CNN simple qui s'adapte automatiquement à la taille des images.

        Args:
            example_input (torch.Tensor): une image d'exemple (tensor) pour
calculer la taille des couches fully connected
            num_classes (int): nombre de classes de sortie
            input_channels (int): nombre de canaux d'entrée
        """
        super(SimpleCNN, self).__init__()

        # === Convolution layers ===
        self.conv1 = nn.Conv2d(in_channels=input_channels, out_channels=32,
kernel_size=3, stride=1, padding=1)
        self.bn1 = nn.BatchNorm2d(32)

        self.conv2 = nn.Conv2d(in_channels=32, out_channels=64, kernel_size=3,
stride=1, padding=1)
        self.bn2 = nn.BatchNorm2d(64)

        self.pool = nn.MaxPool2d(kernel_size=2, stride=2)
        self.dropout = nn.Dropout(0.25)

        # Calcul dynamique de la taille du flatten
        with torch.no_grad():
            x = self._forward_features(example_input.unsqueeze(0))  # batch
fictif
            flatten_dim = x.view(1, -1).shape[1]

        # Fully connected
        self.fc1 = nn.Linear(flatten_dim, 128)
        self.fc2 = nn.Linear(128, num_classes)

    def _forward_features(self, x):
        """Passe avant pour calculer la taille de sortie des
convolutions/pooling."""
        x = self.pool(F.relu(self.bn1(self.conv1(x))))
        x = self.pool(F.relu(self.bn2(self.conv2(x))))
        return x

    def forward(self, x):
        x = self._forward_features(x)
        x = x.view(x.size(0), -1)  # Flatten
        x = F.relu(self.fc1(x))
        x = self.dropout(x)
        x = self.fc2(x)
        return x
```

```
train_loader = DataLoader(train_data, batch_size=64, shuffle=True)
test_loader = DataLoader(test_data, batch_size=64, shuffle=False)
```

```
# === Initialisation ===
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = SimpleCNN(example_input=sample_image, num_classes=10, input_channels=1)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
```

```
# === Entraînement (1 epoch) ===
model.train()
for images, labels in train_loader:
    images, labels = images.to(device), labels.to(device)
    optimizer.zero_grad()
    outputs = model(images)
    loss = criterion(outputs, labels)
    loss.backward()
    optimizer.step()
```

```
# === Évaluation et métriques ===
model.eval()
predictions = []
all_labels = []

with torch.no_grad():
    for images, labels in test_loader:
        images, labels = images.to(device), labels.to(device)
        outputs = model(images)
        _, predicted = torch.max(outputs, 1)
        all_preds.extend(predicted.cpu().numpy())
        all_labels.extend(labels.cpu().numpy())
```

```python
from sklearn.metrics import accuracy_score, precision_score, recall_score

# === Calcul des métriques ===
accuracy = accuracy_score(all_labels, all_preds)
precision = precision_score(all_labels, all_preds, average='weighted')  #
weighted pour multi-classes
recall = recall_score(all_labels, all_preds, average='weighted')

print(f"Accuracy : {accuracy:.4f}")
print(f"Precision : {precision:.4f}")
print(f"Recall : {recall:.4f}")
```

```
Accuracy : 0.8797
Precision : 0.8840
Recall : 0.8797
```