```
!pip install wandb
```

```
Defaulting to user installation because normal site-packages is not
writeable
Looking in indexes: https://pypi.org/simple,
https://pypi.ngc.nvidia.com
Requirement already satisfied: wandb in
/home/work/.local/lib/python3.10/site-packages (0.18.7)
Requirement already satisfied: click!=8.0.0,>=7.1 in
/usr/local/lib/python3.10/dist-packages (from wandb) (8.1.7)
Requirement already satisfied: docker-pycreds>=0.4.0 in
/home/work/.local/lib/python3.10/site-packages (from wandb) (0.4.0)
Requirement already satisfied: gitpython!=3.1.29,>=1.0.0 in
/usr/local/lib/python3.10/dist-packages (from wandb) (3.1.26)
Requirement already satisfied: platformdirs in
/usr/local/lib/python3.10/dist-packages (from wandb) (4.2.0)
Requirement already satisfied: protobuf!=4.21.0,!=5.28.0,<6,>=3.19.0
in /usr/local/lib/python3.10/dist-packages (from wandb) (3.20.3)
Requirement already satisfied: psutil>=5.0.0 in
/usr/local/lib/python3.10/dist-packages (from wandb) (5.9.4)
Requirement already satisfied: pyyaml in
/usr/local/lib/python3.10/dist-packages (from wandb) (6.0.1)
Requirement already satisfied: requests<3,>=2.0.0 in
/usr/local/lib/python3.10/dist-packages (from wandb) (2.31.0)
Requirement already satisfied: sentry-sdk>=2.0.0 in
/home/work/.local/lib/python3.10/site-packages (from wandb) (2.19.0)
Requirement already satisfied: setproctitle in
/home/work/.local/lib/python3.10/site-packages (from wandb) (1.3.4)
Requirement already satisfied: setuptools in
/usr/local/lib/python3.10/dist-packages (from wandb) (68.2.2)
Requirement already satisfied: typing-extensions<5,>=4.4 in
/usr/local/lib/python3.10/dist-packages (from wandb) (4.10.0)
Requirement already satisfied: six>=1.4.0 in
/usr/local/lib/python3.10/dist-packages (from docker-pycreds>=0.4.0-
>wandb) (1.16.0)
Requirement already satisfied: gitdb<5,>=4.0.1 in
/usr/local/lib/python3.10/dist-packages (from gitpython!
=3.1.29,>=1.0.0->wandb) (4.0.5)
Requirement already satisfied: charset-normalizer<4,>=2 in
/usr/local/lib/python3.10/dist-packages (from requests<3,>=2.0.0-
>wandb) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in
/usr/local/lib/python3.10/dist-packages (from requests<3,>=2.0.0-
>wandb) (3.6)
Requirement already satisfied: urllib3<3,>=1.21.1 in
/usr/local/lib/python3.10/dist-packages (from requests<3,>=2.0.0-
>wandb) (1.26.18)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.10/dist-packages (from requests<3,>=2.0.0-
>wandb) (2024.2.2)
```

```
Requirement already satisfied: smmap<4,>=3.0.1 in
/usr/local/lib/python3.10/dist-packages (from gitdb<5,>=4.0.1-
>gitpython!=3.1.29,>=1.0.0->wandb) (3.0.5)
DEPRECATION: devscripts 2.22.1ubuntu1 has a non-standard version
number. pip 24.1 will enforce this behaviour change. A possible
replacement is to upgrade to a newer version of devscripts or contact
the author to suggest that they release a version with a conforming
version number. Discussion can be found at
https://github.com/pypa/pip/issues/12063

[notice] A new release of pip is available: 24.0 -> 24.3.1
[notice] To update, run: python -m pip install --upgrade pip

pip show wandb

Name: wandb
Version: 0.18.7
Summary: A CLI and library for interacting with the Weights & Biases
API.
Home-page:
Author:
Author-email: Weights & Biases <support@wandb.com>
License: MIT License

        Copyright (c) 2021 Weights and Biases, Inc.

        Permission is hereby granted, free of charge, to any person
obtaining a copy
        of this software and associated documentation files (the
"Software"), to deal
        in the Software without restriction, including without
limitation the rights
        to use, copy, modify, merge, publish, distribute, sublicense,
and/or sell
        copies of the Software, and to permit persons to whom the
Software is
        furnished to do so, subject to the following conditions:

        The above copyright notice and this permission notice shall be
included in all
        copies or substantial portions of the Software.

        THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY
KIND, EXPRESS OR
        IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
MERCHANTABILITY,
        FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO
EVENT SHALL THE
        AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES
OR OTHER
```

```
        LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR
OTHERWISE, ARISING FROM,
        OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER
DEALINGS IN THE
        SOFTWARE.
Location: /home/work/.local/lib/python3.10/site-packages
Requires: click, docker-pycreds, gitpython, platformdirs, protobuf,
psutil, pyyaml, requests, sentry-sdk, setproctitle, setuptools,
typing-extensions
Required-by:
Note: you may need to restart the kernel to use updated packages.
```

```
import wandb
wandb.login
```

```
<function wandb.sdk.wandb_login.login(anonymous:
Optional[Literal['must', 'allow', 'never']] = None, key: Optional[str]
= None, relogin: Optional[bool] = None, host: Optional[str] = None,
force: Optional[bool] = None, timeout: Optional[int] = None, verify:
bool = False) -> bool>
```

```
import wandb
wandb.login(key="afebcaccd9929fcb34d6e10db06a3c432acfc56e")
```

```
wandb: Using wandb-core as the SDK backend.  Please refer to
https://wandb.me/wandb-core for more information.
wandb: Currently logged in as: -ddj127 (-ddj127-korea-university-of-
technology-and-education). Use `wandb login --relogin` to force
relogin
wandb: WARNING If you're specifying your api key in code, ensure this
code is not shared publicly.
wandb: WARNING Consider setting the WANDB_API_KEY environment
variable, or running `wandb login` from the command line.
wandb: Appending key for api.wandb.ai to your netrc file:
/home/work/.netrc
```

```
True
```

```
# FashionMNIST
import torch
from torchvision import datasets, transforms
```

```
# FashionMNIST
mnist_train = datasets.FashionMNIST(root='data', train=True,
download=True, transform=transforms.ToTensor())
```

```
#
all_images = torch.cat([img[0].view(-1) for img, _ in mnist_train])
```

```
#
mean = all_images.mean().item()
```

```
std = all_images.std().item()

print(f"Calculated Mean: {mean}")
print(f"Calculated Std: {std}")
```

```
Calculated Mean: 0.28604060411453247
Calculated Std: 0.3530242443084717
```

□□ □□ □□

```
# Import necessary libraries
import os
import random
import torch
from torch import nn
from torch.utils.data import DataLoader, random_split
from torchvision import datasets, transforms
from torch.optim import Adam
from torch.optim.lr_scheduler import StepLR
from torchinfo import summary  # For model structure summary
import matplotlib.pyplot as plt

# Check the device (CPU or GPU)
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(f"Using device: {device}")
```

```
Using device: cuda
```

```
# Function to load Fashion MNIST training and validation data
def get_fashion_mnist_data(batch_size=64):
    # Define transformations for data augmentation
    transform = transforms.Compose([
        transforms.RandomHorizontalFlip(p=0.5),  # Randomly flip
images horizontally
        transforms.RandomRotation(15),          # Randomly rotate
images within ±15 degrees
        transforms.ToTensor(),                  # Convert images to
tensor
        transforms.Normalize(mean=0.286, std=0.353)  # Normalize using
pre-calculated mean and std
    ])
    # Load Fashion MNIST dataset
    dataset = datasets.FashionMNIST(root="./data", train=True,
download=True, transform=transform)
    # Split into training (55,000 samples) and validation (5,000
samples) sets
    train_set, val_set = random_split(dataset, [55000, 5000])
    # Create data loaders for training and validation
    train_loader = DataLoader(train_set, batch_size=batch_size,
```

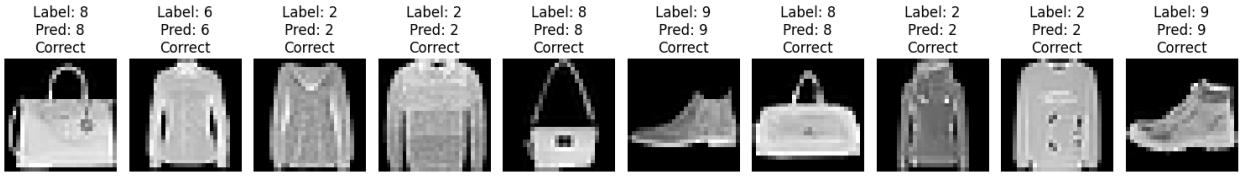
```
shuffle=True)
    val_loader = DataLoader(val_set, batch_size=batch_size)
    return train_loader, val_loader

# Function to load Fashion MNIST test data
def get_fashion_mnist_test_data(batch_size=64):
    # Define transformation (only normalization, no augmentation for
test data)
    transform = transforms.Compose([
        transforms.ToTensor(),
        transforms.Normalize(mean=0.286, std=0.353)
    ])
    # Load Fashion MNIST test dataset
    test_set = datasets.FashionMNIST(root="./data", train=False,
download=True, transform=transform)
    # Create a data loader for test data
    test_loader = DataLoader(test_set, batch_size=batch_size)
    return test_loader

# Define the CNN model
class CNNModel(nn.Module):
    def __init__(self):
        super(CNNModel, self).__init__()
        # Define convolutional layers
        self.conv_layers = nn.Sequential(
            nn.Conv2d(1, 64, kernel_size=5, padding="same"),  # First
conv layer, 64 filters
            nn.BatchNorm2d(64),                                  #
Normalize feature maps
            nn.ReLU(),                                           #
Activation function
            nn.MaxPool2d(kernel_size=2),                         #
Downsample using MaxPool

            nn.Conv2d(64, 128, kernel_size=5, padding="same"),  #
Second conv layer
            nn.BatchNorm2d(128),                                   #
Normalize feature maps
            nn.ReLU(),                                             #
Activation function
            nn.MaxPool2d(kernel_size=2),                           #
Downsample using MaxPool

            nn.Conv2d(128, 256, kernel_size=5, padding="same"), #
Third conv layer
            nn.BatchNorm2d(256),                                   #
Normalize feature maps
            nn.ReLU(),                                             #
Activation function
            nn.MaxPool2d(kernel_size=2)                            #
```

```
Downsample using MaxPool
        )
        # Define fully connected layers
        self.fc_layers = nn.Sequential(
            nn.Flatten(),                               # Flatten feature
maps for dense layers
            nn.Linear(256 * 3 * 3, 256),            # First dense
layer
            nn.ReLU(),                                  # Activation
function
            nn.Dropout(0.2),                            # Dropout for
regularization
            nn.Linear(256, 10)                          # Output layer (10
classes)
        )

    def forward(self, x):
        x = self.conv_layers(x)  # Pass input through convolutional
layers
        x = self.fc_layers(x)    # Pass through fully connected layers
        return x

# Initialize weights for model layers
def initialize_weights(m):
    if isinstance(m, nn.Conv2d) or isinstance(m, nn.Linear):
        nn.init.kaiming_normal_(m.weight)  # He initialization for
better training stability

# Function to train and evaluate the model
def train_and_evaluate(model, train_loader, val_loader, epochs=20,
learning_rate=0.001):
    model.to(device)  # Move model to the selected device
    model.apply(initialize_weights)  # Apply weight initialization
    optimizer = Adam(model.parameters(), lr=learning_rate)  #
Optimizer
    scheduler = StepLR(optimizer, step_size=10, gamma=0.1)  # Learning
rate scheduler
    criterion = nn.CrossEntropyLoss()  # Loss function

    for epoch in range(1, epochs + 1):
        # Training phase
        model.train()  # Set model to training mode
        train_loss, train_correct, train_total = 0, 0, 0
        for images, labels in train_loader:
            images, labels = images.to(device), labels.to(device)
            optimizer.zero_grad()  # Clear gradients
            outputs = model(images)  # Forward pass
            loss = criterion(outputs, labels)  # Compute loss
            loss.backward()  # Backpropagation
            optimizer.step()  # Update weights
```

```
            train_loss += loss.item()
            _, predicted = torch.max(outputs, 1)  # Get predictions
            train_total += labels.size(0)
            train_correct += (predicted == labels).sum().item()

        scheduler.step()  # Adjust learning rate
        train_accuracy = train_correct / train_total

        # Validation phase
        model.eval()  # Set model to evaluation mode
        val_loss, val_correct, val_total = 0, 0, 0
        with torch.no_grad():
            for images, labels in val_loader:
                images, labels = images.to(device), labels.to(device)
                outputs = model(images)
                loss = criterion(outputs, labels)
                val_loss += loss.item()
                _, predicted = torch.max(outputs, 1)
                val_total += labels.size(0)
                val_correct += (predicted == labels).sum().item()

        val_accuracy = val_correct / val_total
        print(f"Epoch {epoch}/{epochs} - Train Loss: {train_loss:.4f},
Train Acc: {train_accuracy:.4f}, "
              f"Val Loss: {val_loss:.4f}, Val Acc:
{val_accuracy:.4f}")

    return model

# Function to evaluate the model on test data
def evaluate_test_data(model, test_loader):
    model.eval()  # Set model to evaluation mode
    test_correct, test_total = 0, 0
    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)
            test_total += labels.size(0)
            test_correct += (predicted == labels).sum().item()
    test_accuracy = test_correct / test_total
    print(f"Test Accuracy: {test_accuracy:.4f}")
    return test_accuracy

# Function to visualize predictions
def visualize_predictions(model, test_loader, num_samples=10):
    model.eval()
    images, labels = next(iter(test_loader))  # Fetch a batch of test
data
    images, labels = images.to(device), labels.to(device)
```

```
    # Select random samples for visualization
    indices = random.sample(range(len(images)), num_samples)
    selected_images = images[indices]
    selected_labels = labels[indices]

    with torch.no_grad():
        outputs = model(selected_images)  # Get predictions for
selected samples
        _, predicted = torch.max(outputs, 1)

    # Plot the results
    plt.figure(figsize=(15, 5))
    for i in range(num_samples):
        img = selected_images[i].cpu().squeeze()
        label = selected_labels[i].item()
        pred = predicted[i].item()
        plt.subplot(1, num_samples, i + 1)
        plt.imshow(img, cmap="gray")
        plt.title(f"Label: {label}\nPred: {pred}\n{'Correct' if label
== pred else 'Wrong'}")
        plt.axis("off")
    plt.tight_layout()
    plt.show()


# Main execution
if __name__ == "__main__":
    batch_size = 64
    epochs = 20
    learning_rate = 0.001

    # Data loaders
    train_loader, val_loader = get_fashion_mnist_data(batch_size)
    test_loader = get_fashion_mnist_test_data(batch_size)

    # Model
    model = CNNModel()
    print("\nModel Summary:")
    summary(model, input_size=(batch_size, 1, 28, 28),
col_names=["input_size", "output_size", "num_params", "trainable"])

    # Training and evaluation
    model = train_and_evaluate(model, train_loader, val_loader,
epochs, learning_rate)

    # Test evaluation
    evaluate_test_data(model, test_loader)
```

```
    # Visualize predictions
    visualize_predictions(model, test_loader, num_samples=10)


Model Summary:
Epoch 1/20 - Train Loss: 489.7960, Train Acc: 0.7971, Val Loss:
32.5336, Val Acc: 0.8522
Epoch 2/20 - Train Loss: 322.9123, Train Acc: 0.8628, Val Loss:
25.3313, Val Acc: 0.8814
Epoch 3/20 - Train Loss: 281.1800, Train Acc: 0.8807, Val Loss:
25.0492, Val Acc: 0.8824
Epoch 4/20 - Train Loss: 257.1715, Train Acc: 0.8908, Val Loss:
24.5417, Val Acc: 0.8854
Epoch 5/20 - Train Loss: 240.5925, Train Acc: 0.8965, Val Loss:
21.9141, Val Acc: 0.8970
Epoch 6/20 - Train Loss: 228.9604, Train Acc: 0.9029, Val Loss:
23.3361, Val Acc: 0.8890
Epoch 7/20 - Train Loss: 215.2766, Train Acc: 0.9085, Val Loss:
20.8814, Val Acc: 0.9022
Epoch 8/20 - Train Loss: 205.7806, Train Acc: 0.9128, Val Loss:
19.6719, Val Acc: 0.9112
Epoch 9/20 - Train Loss: 196.6733, Train Acc: 0.9164, Val Loss:
21.2644, Val Acc: 0.8958
Epoch 10/20 - Train Loss: 188.5876, Train Acc: 0.9197, Val Loss:
18.7308, Val Acc: 0.9110
Epoch 11/20 - Train Loss: 153.1880, Train Acc: 0.9336, Val Loss:
16.3315, Val Acc: 0.9198
Epoch 12/20 - Train Loss: 144.2179, Train Acc: 0.9388, Val Loss:
16.7776, Val Acc: 0.9214
Epoch 13/20 - Train Loss: 139.2498, Train Acc: 0.9406, Val Loss:
16.4092, Val Acc: 0.9232
Epoch 14/20 - Train Loss: 136.2202, Train Acc: 0.9423, Val Loss:
15.9400, Val Acc: 0.9228
Epoch 15/20 - Train Loss: 133.0198, Train Acc: 0.9427, Val Loss:
16.2893, Val Acc: 0.9206
Epoch 16/20 - Train Loss: 128.9962, Train Acc: 0.9451, Val Loss:
15.8123, Val Acc: 0.9272
Epoch 17/20 - Train Loss: 126.2232, Train Acc: 0.9459, Val Loss:
16.2611, Val Acc: 0.9224
Epoch 18/20 - Train Loss: 125.3538, Train Acc: 0.9466, Val Loss:
16.1192, Val Acc: 0.9240
Epoch 19/20 - Train Loss: 121.0783, Train Acc: 0.9476, Val Loss:
16.0052, Val Acc: 0.9280
Epoch 20/20 - Train Loss: 119.7629, Train Acc: 0.9482, Val Loss:
16.4017, Val Acc: 0.9232
Test Accuracy: 0.9327
```



ResNet18 □□ □□ □□□□ acc □ □□□□□□. 94 □□

out of memory 가 ㄴ ㅏ ㅇ ㅗ ㄴ ㅔ ㅇ ㅛ

```
import torch
from torch import nn
from torchvision.models.resnet import ResNet, Bottleneck
from torchinfo import summary
from torch.optim import Adam
from torch.utils.data import DataLoader, random_split
from torchvision import datasets, transforms
from torch.optim.lr_scheduler import StepLR

# Device configuration
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(f"Using device: {device}")

# Wide Residual Network (WRN) implementation using Bottleneck block
class WideResNet(ResNet):
    def __init__(self, widen_factor=4, num_classes=10):
        # Bottleneck block allows width_per_group to be adjusted
        super(WideResNet, self).__init__(
            block=Bottleneck,
            layers=[3, 4, 6, 3],  # ResNet50 configuration (deeper
than ResNet18)
            num_classes=num_classes,
            width_per_group=64 * widen_factor  # Apply widen factor
        )
        # Modify the first convolutional layer to accept grayscale
input
        self.conv1 = nn.Conv2d(
            1, 64, kernel_size=7, stride=2, padding=3, bias=False
        )

# Data loaders
def get_fashion_mnist_data(batch_size=64):
    transform = transforms.Compose([
        transforms.RandomCrop(28, padding=4),
        transforms.RandomHorizontalFlip(),
        transforms.ToTensor(),
        transforms.Normalize(mean=0.286, std=0.353),
    ])
```

```
    dataset = datasets.FashionMNIST(root="./data", train=True,
download=True, transform=transform)
    train_set, val_set = random_split(dataset, [55000, 5000])
    train_loader = DataLoader(train_set, batch_size=batch_size,
shuffle=True)
    val_loader = DataLoader(val_set, batch_size=batch_size)
    return train_loader, val_loader

def get_fashion_mnist_test_data(batch_size=64):
    transform = transforms.Compose([
        transforms.ToTensor(),
        transforms.Normalize(mean=0.286, std=0.353),
    ])
    test_set = datasets.FashionMNIST(root="./data", train=False,
download=True, transform=transform)
    test_loader = DataLoader(test_set, batch_size=batch_size)
    return test_loader

# Training and validation
def train_and_evaluate(model, train_loader, val_loader, epochs=20,
learning_rate=0.001):
    model.to(device)
    optimizer = Adam(model.parameters(), lr=learning_rate)
    scheduler = StepLR(optimizer, step_size=10, gamma=0.1)
    criterion = nn.CrossEntropyLoss()

    for epoch in range(1, epochs + 1):
        # Training phase
        model.train()
        train_loss, train_correct, train_total = 0, 0, 0
        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()

            train_loss += loss.item()
            _, predicted = torch.max(outputs, 1)
            train_total += labels.size(0)
            train_correct += (predicted == labels).sum().item()

        scheduler.step()
        train_accuracy = train_correct / train_total

        # Validation phase
        model.eval()
        val_loss, val_correct, val_total = 0, 0, 0
        with torch.no_grad():
```

```
            for images, labels in val_loader:
                images, labels = images.to(device), labels.to(device)
                outputs = model(images)
                loss = criterion(outputs, labels)

                val_loss += loss.item()
                _, predicted = torch.max(outputs, 1)
                val_total += labels.size(0)
                val_correct += (predicted == labels).sum().item()

        val_accuracy = val_correct / val_total
        print(f"Epoch {epoch}/{epochs} - Train Loss: {train_loss:.4f},
Train Acc: {train_accuracy:.4f}, "
              f"Val Loss: {val_loss:.4f}, Val Acc:
{val_accuracy:.4f}")

    return model

# Test evaluation
def evaluate_test_data(model, test_loader):
    model.eval()
    test_correct, test_total = 0, 0
    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)
            test_total += labels.size(0)
            test_correct += (predicted == labels).sum().item()

    test_accuracy = test_correct / test_total
    print(f"Test Accuracy: {test_accuracy:.4f}")
    return test_accuracy

# Main execution
if __name__ == "__main__":
    batch_size = 32
    epochs = 20
    learning_rate = 0.001

    # Load data
    train_loader, val_loader = get_fashion_mnist_data(batch_size)
    test_loader = get_fashion_mnist_test_data(batch_size)

    # Initialize WRN model
    model = WideResNet(widen_factor=4)
    print("\nModel Summary:")
    summary(model, input_size=(batch_size, 1, 28, 28),
col_names=["input_size", "output_size", "num_params", "trainable"])
```

```
    # Train and validate
    model = train_and_evaluate(model, train_loader, val_loader,
epochs, learning_rate)

    # Test evaluation
    evaluate_test_data(model, test_loader)

Using device: cuda

Model Summary:

----------------------------------------------------------------------
-----
OutOfMemoryError                          Traceback (most recent call
last)
Cell In[1], line 131
    128 summary(model, input_size=(batch_size, 1, 28, 28),
col_names=["input_size", "output_size", "num_params", "trainable"])
    130 # Train and validate
--> 131 model = train_and_evaluate(model, train_loader, val_loader,
epochs, learning_rate)
    133 # Test evaluation
    134 evaluate_test_data(model, test_loader)

Cell In[1], line 69, in train_and_evaluate(model, train_loader,
val_loader, epochs, learning_rate)
     67 loss = criterion(outputs, labels)
     68 loss.backward()
---> 69 optimizer.step()
     71 train_loss += loss.item()
     72 _, predicted = torch.max(outputs, 1)

File
/usr/local/lib/python3.10/dist-packages/torch/optim/lr_scheduler.py:75
, in
LRScheduler.__init__.<locals>.with_counter.<locals>.wrapper(*args,
**kwargs)
     73 instance._step_count += 1
     74 wrapped = func.__get__(instance, cls)
---> 75 return wrapped(*args, **kwargs)

File
/usr/local/lib/python3.10/dist-packages/torch/optim/optimizer.py:391,
in Optimizer.profile_hook_step.<locals>.wrapper(*args, **kwargs)
    386         else:
    387             raise RuntimeError(
    388                 f"{func} must return None or a tuple of
(new_args, new_kwargs), but got {result}."
    389             )
--> 391 out = func(*args, **kwargs)
```

```
    392 self._optimizer_step_code()
    394 # call optimizer step post hooks

File
/usr/local/lib/python3.10/dist-packages/torch/optim/optimizer.py:76,
in _use_grad_for_differentiable.<locals>._use_grad(self, *args,
**kwargs)
     74     torch.set_grad_enabled(self.defaults['differentiable'])
     75     torch._dynamo.graph_break()
---> 76     ret = func(self, *args, **kwargs)
     77 finally:
     78     torch._dynamo.graph_break()

File /usr/local/lib/python3.10/dist-packages/torch/optim/adam.py:168,
in Adam.step(self, closure)
    157     beta1, beta2 = group['betas']
    159     has_complex = self._init_group(
    160         group,
    161         params_with_grad,
   (...)
    165         max_exp_avg_sqs,
    166         state_steps)
--> 168     adam(
    169         params_with_grad,
    170         grads,
    171         exp_avgs,
    172         exp_avg_sqs,
    173         max_exp_avg_sqs,
    174         state_steps,
    175         amsgrad=group['amsgrad'],
    176         has_complex=has_complex,
    177         beta1=beta1,
    178         beta2=beta2,
    179         lr=group['lr'],
    180         weight_decay=group['weight_decay'],
    181         eps=group['eps'],
    182         maximize=group['maximize'],
    183         foreach=group['foreach'],
    184         capturable=group['capturable'],
    185         differentiable=group['differentiable'],
    186         fused=group['fused'],
    187         grad_scale=getattr(self, "grad_scale", None),
    188         found_inf=getattr(self, "found_inf", None),
    189     )
    191 return loss

File /usr/local/lib/python3.10/dist-packages/torch/optim/adam.py:318,
in adam(params, grads, exp_avgs, exp_avg_sqs, max_exp_avg_sqs,
state_steps, foreach, capturable, differentiable, fused, grad_scale,
found_inf, has_complex, amsgrad, beta1, beta2, lr, weight_decay, eps,
```

```
maximize)
    315 else:
    316     func = _single_tensor_adam
--> 318 func(params,
    319      grads,
    320      exp_avgs,
    321      exp_avg_sqs,
    322      max_exp_avg_sqs,
    323      state_steps,
    324      amsgrad=amsgrad,
    325      has_complex=has_complex,
    326      beta1=beta1,
    327      beta2=beta2,
    328      lr=lr,
    329      weight_decay=weight_decay,
    330      eps=eps,
    331      maximize=maximize,
    332      capturable=capturable,
    333      differentiable=differentiable,
    334      grad_scale=grad_scale,
    335      found_inf=found_inf)

File /usr/local/lib/python3.10/dist-packages/torch/optim/adam.py:581,
in _multi_tensor_adam(params, grads, exp_avgs, exp_avg_sqs,
max_exp_avg_sqs, state_steps, grad_scale, found_inf, amsgrad,
has_complex, beta1, beta2, lr, weight_decay, eps, maximize,
capturable, differentiable)
    579     exp_avg_sq_sqrt =
torch._foreach_sqrt(device_max_exp_avg_sqs)
    580 else:
--> 581     exp_avg_sq_sqrt = torch._foreach_sqrt(device_exp_avg_sqs)
    583 torch._foreach_div_(exp_avg_sq_sqrt, bias_correction2_sqrt)
    584 torch._foreach_add_(exp_avg_sq_sqrt, eps)

OutOfMemoryError: CUDA out of memory. Tried to allocate 36.00 MiB. GPU
0 has a total capacity of 31.74 GiB of which 4.88 MiB is free.
Including non-PyTorch memory, this process has 3.77 GiB memory in use.
Of the allocated memory 3.37 GiB is allocated by PyTorch, and 24.17
MiB is reserved by PyTorch but unallocated. If reserved but
unallocated memory is large try setting
PYTORCH_CUDA_ALLOC_CONF=expandable_segments:True to avoid
fragmentation.  See documentation for Memory Management
(https://pytorch.org/docs/stable/notes/cuda.html#environment-
variables)
```

```
#  3 ResNet18
from pathlib import Path
import os
import torch
import wandb
```

```
from torch import nn
from torch.utils.data import DataLoader, random_split
from torchvision import datasets, models
from torchvision.transforms import transforms
from torch.optim import Adam

# Use the current working directory
BASE_PATH = Path().resolve()
print(BASE_PATH)

import sys
sys.path.append(str(BASE_PATH))

from utils import get_num_cpu_cores, is_linux, is_windows


def get_fashion_mnist_data():
    data_path = os.path.join(BASE_PATH, "_00_data", "j_fashion_mnist")

    f_mnist_train = datasets.FashionMNIST(data_path, train=True,
download=True, transform=transforms.ToTensor())
    f_mnist_train, f_mnist_validation = random_split(f_mnist_train,
[55_000, 5_000])

    print("Num Train Samples: ", len(f_mnist_train))
    print("Num Validation Samples: ", len(f_mnist_validation))
    print("Sample Shape: ", f_mnist_train[0][0].shape)  #
torch.Size([1, 28, 28])

    num_data_loading_workers = get_num_cpu_cores() if is_linux() or
is_windows() else 0
    print("Number of Data Loading Workers:", num_data_loading_workers)

    train_data_loader = DataLoader(
        dataset=f_mnist_train, batch_size=wandb.config.batch_size,
shuffle=True,
        pin_memory=True, num_workers=num_data_loading_workers
    )

    validation_data_loader = DataLoader(
        dataset=f_mnist_validation,
batch_size=wandb.config.batch_size,
        pin_memory=True, num_workers=num_data_loading_workers
    )

    return train_data_loader, validation_data_loader


def train_and_evaluate(epochs, train_data_loader,
validation_data_loader):
```

```
    device = torch.device("cuda" if torch.cuda.is_available() else
"cpu")

    # Load ResNet18 and modify for FashionMNIST
    model = models.resnet18(pretrained=False)  # Load ResNet18 without
pre-trained weights
    model.conv1 = nn.Conv2d(1, 64, kernel_size=7, stride=2, padding=3,
bias=False)  # Modify for single-channel input
    model.fc = nn.Linear(model.fc.in_features, 10)  # Modify the
output layer for 10 classes
    model = model.to(device)

    optimizer = Adam(model.parameters(),
lr=wandb.config.learning_rate)
    criterion = nn.CrossEntropyLoss()

    for epoch in range(1, epochs + 1):
        # Training
        model.train()
        train_loss, train_correct, train_total = 0, 0, 0
        for images, labels in train_data_loader:
            images, labels = images.to(device), labels.to(device)
            optimizer.zero_grad()

            outputs = model(images)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()

            train_loss += loss.item()
            _, predicted = torch.max(outputs, 1)
            train_total += labels.size(0)
            train_correct += (predicted == labels).sum().item()

        train_accuracy = train_correct / train_total
        train_loss /= len(train_data_loader)

        # Validation
        model.eval()
        val_loss, val_correct, val_total = 0, 0, 0
        with torch.no_grad():
            for images, labels in validation_data_loader:
                images, labels = images.to(device), labels.to(device)

                outputs = model(images)
                loss = criterion(outputs, labels)

                val_loss += loss.item()
                _, predicted = torch.max(outputs, 1)
                val_total += labels.size(0)
```

```
                val_correct += (predicted == labels).sum().item()

        val_accuracy = val_correct / val_total
        val_loss /= len(validation_data_loader)

        # Log metrics to wandb
        wandb.log({
            "epoch": epoch,
            "train_loss": train_loss,
            "train_accuracy": train_accuracy,
            "val_loss": val_loss,
            "val_accuracy": val_accuracy
        })

        print(f"Epoch [{epoch}/{epochs}] - Train Loss:
{train_loss:.4f}, Train Acc: {train_accuracy:.4f}, "
              f"Val Loss: {val_loss:.4f}, Val Acc:
{val_accuracy:.4f}")


if __name__ == "__main__":
    config = {"batch_size": 64, "learning_rate": 0.0005, "epochs": 25}
    wandb.init(mode="disabled", config=config)

    train_data_loader, validation_data_loader =
get_fashion_mnist_data()

    # Start training and evaluating
    train_and_evaluate(wandb.config.epochs, train_data_loader,
validation_data_loader)

    wandb.finish()
```

```
/home/work/DL
Num Train Samples:  55000
Num Validation Samples:  5000
Sample Shape:  torch.Size([1, 28, 28])
Number of Data Loading Workers: 2

/usr/local/lib/python3.10/dist-packages/torchvision/models/
_utils.py:208: UserWarning: The parameter 'pretrained' is deprecated
since 0.13 and may be removed in the future, please use 'weights'
instead.
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/torchvision/models/_utils.py:2
23: UserWarning: Arguments other than a weight enum or `None` for
'weights' are deprecated since 0.13 and may be removed in the future.
The current behavior is equivalent to passing `weights=None`.
  warnings.warn(msg)
```

```
Epoch [1/25] - Train Loss: 0.4282, Train Acc: 0.8444, Val Loss:
0.3633, Val Acc: 0.8630
Epoch [2/25] - Train Loss: 0.3026, Train Acc: 0.8890, Val Loss:
0.2985, Val Acc: 0.8834
Epoch [3/25] - Train Loss: 0.2626, Train Acc: 0.9023, Val Loss:
0.2906, Val Acc: 0.8916
Epoch [4/25] - Train Loss: 0.2306, Train Acc: 0.9148, Val Loss:
0.3073, Val Acc: 0.8880
Epoch [5/25] - Train Loss: 0.2095, Train Acc: 0.9227, Val Loss:
0.2857, Val Acc: 0.8938
Epoch [6/25] - Train Loss: 0.1892, Train Acc: 0.9284, Val Loss:
0.2791, Val Acc: 0.9030
Epoch [7/25] - Train Loss: 0.1699, Train Acc: 0.9365, Val Loss:
0.2903, Val Acc: 0.9050
Epoch [8/25] - Train Loss: 0.1508, Train Acc: 0.9430, Val Loss:
0.3259, Val Acc: 0.8954
Epoch [9/25] - Train Loss: 0.1390, Train Acc: 0.9481, Val Loss:
0.2833, Val Acc: 0.9012
Epoch [10/25] - Train Loss: 0.1206, Train Acc: 0.9541, Val Loss:
0.3286, Val Acc: 0.8968
Epoch [11/25] - Train Loss: 0.1124, Train Acc: 0.9575, Val Loss:
0.2782, Val Acc: 0.9112
Epoch [12/25] - Train Loss: 0.1006, Train Acc: 0.9621, Val Loss:
0.2825, Val Acc: 0.9124
Epoch [13/25] - Train Loss: 0.0874, Train Acc: 0.9674, Val Loss:
0.3134, Val Acc: 0.9088
Epoch [14/25] - Train Loss: 0.0840, Train Acc: 0.9694, Val Loss:
0.3257, Val Acc: 0.9072
Epoch [15/25] - Train Loss: 0.0738, Train Acc: 0.9724, Val Loss:
0.3495, Val Acc: 0.9078
Epoch [16/25] - Train Loss: 0.0689, Train Acc: 0.9742, Val Loss:
0.3598, Val Acc: 0.9076
Epoch [17/25] - Train Loss: 0.0613, Train Acc: 0.9768, Val Loss:
0.3792, Val Acc: 0.9020
Epoch [18/25] - Train Loss: 0.0587, Train Acc: 0.9787, Val Loss:
0.3794, Val Acc: 0.9078
Epoch [19/25] - Train Loss: 0.0560, Train Acc: 0.9789, Val Loss:
0.3738, Val Acc: 0.9112



/home/work/DL
Num Train Samples:  55000
Num Validation Samples:  5000
Sample Shape:  torch.Size([1, 28, 28])
Number of Data Loading Workers: 2

/usr/local/lib/python3.10/dist-packages/torchvision/models/
_utils.py:208: UserWarning: The parameter 'pretrained' is deprecated
since 0.13 and may be removed in the future, please use 'weights'
```

```
instead.
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/torchvision/models/_utils.py:223: UserWarning: Arguments other than a weight enum or `None` for 'weights' are deprecated since 0.13 and may be removed in the future. The current behavior is equivalent to passing `weights=None`.
  warnings.warn(msg)

Epoch [1/25] - Train Loss: 0.5291, Train Acc: 0.8052, Val Loss: 0.3900, Val Acc: 0.8570
Epoch [2/25] - Train Loss: 0.3796, Train Acc: 0.8595, Val Loss: 0.3604, Val Acc: 0.8702
Epoch [3/25] - Train Loss: 0.3409, Train Acc: 0.8727, Val Loss: 0.3287, Val Acc: 0.8774
Epoch [4/25] - Train Loss: 0.3166, Train Acc: 0.8827, Val Loss: 0.3252, Val Acc: 0.8794
Epoch [5/25] - Train Loss: 0.2985, Train Acc: 0.8897, Val Loss: 0.3240, Val Acc: 0.8800
Epoch [6/25] - Train Loss: 0.2864, Train Acc: 0.8939, Val Loss: 0.2803, Val Acc: 0.9040
Epoch [7/25] - Train Loss: 0.2701, Train Acc: 0.8994, Val Loss: 0.2799, Val Acc: 0.8966
Epoch [8/25] - Train Loss: 0.2661, Train Acc: 0.9003, Val Loss: 0.2647, Val Acc: 0.9016
Epoch [9/25] - Train Loss: 0.2535, Train Acc: 0.9056, Val Loss: 0.2651, Val Acc: 0.9030
Epoch [10/25] - Train Loss: 0.2431, Train Acc: 0.9089, Val Loss: 0.2802, Val Acc: 0.8910
Epoch [11/25] - Train Loss: 0.2364, Train Acc: 0.9121, Val Loss: 0.2435, Val Acc: 0.9100
Epoch [12/25] - Train Loss: 0.2287, Train Acc: 0.9144, Val Loss: 0.2450, Val Acc: 0.9112
Epoch [13/25] - Train Loss: 0.2206, Train Acc: 0.9173, Val Loss: 0.2534, Val Acc: 0.9080
Epoch [14/25] - Train Loss: 0.2159, Train Acc: 0.9198, Val Loss: 0.2658, Val Acc: 0.9006
Epoch [15/25] - Train Loss: 0.2078, Train Acc: 0.9209, Val Loss: 0.2350, Val Acc: 0.9134
Epoch [16/25] - Train Loss: 0.2014, Train Acc: 0.9222, Val Loss: 0.2396, Val Acc: 0.9136
Epoch [17/25] - Train Loss: 0.1993, Train Acc: 0.9251, Val Loss: 0.2287, Val Acc: 0.9120
Epoch [18/25] - Train Loss: 0.1927, Train Acc: 0.9276, Val Loss: 0.2429, Val Acc: 0.9114
Epoch [19/25] - Train Loss: 0.1878, Train Acc: 0.9285, Val Loss: 0.2349, Val Acc: 0.9138
Epoch [20/25] - Train Loss: 0.1825, Train Acc: 0.9312, Val Loss: 0.2396, Val Acc: 0.9142
Epoch [21/25] - Train Loss: 0.1775, Train Acc: 0.9324, Val Loss: 0.2417, Val Acc: 0.9162
```

```
Epoch [22/25] - Train Loss: 0.1728, Train Acc: 0.9347, Val Loss:
0.2470, Val Acc: 0.9154
Epoch [23/25] - Train Loss: 0.1652, Train Acc: 0.9359, Val Loss:
0.2372, Val Acc: 0.9120
Epoch [24/25] - Train Loss: 0.1644, Train Acc: 0.9367, Val Loss:
0.2318, Val Acc: 0.9180
Epoch [25/25] - Train Loss: 0.1585, Train Acc: 0.9398, Val Loss:
0.2411, Val Acc: 0.9178
```

```python
from pathlib import Path
import os
import torch
import wandb
from torch import nn
from torch.utils.data import DataLoader, random_split
from torchvision import datasets, transforms
from torch.optim import Adam

# Use the current working directory
BASE_PATH = Path().resolve()
print(BASE_PATH)

import sys
sys.path.append(str(BASE_PATH))

from utils import get_num_cpu_cores, is_linux, is_windows


def get_fashion_mnist_data():
    data_path = os.path.join(BASE_PATH, "_00_data", "j_fashion_mnist")

    transform = transforms.Compose([
        transforms.ToTensor(),
        transforms.Normalize(mean=0.286, std=0.353)
    ])

    f_mnist_train = datasets.FashionMNIST(data_path, train=True,
download=True, transform=transform)
    f_mnist_train, f_mnist_validation = random_split(f_mnist_train,
[55_000, 5_000])

    print("Num Train Samples: ", len(f_mnist_train))
    print("Num Validation Samples: ", len(f_mnist_validation))
    print("Sample Shape: ", f_mnist_train[0][0].shape)  #
torch.Size([1, 28, 28])

    num_data_loading_workers = get_num_cpu_cores() if is_linux() or
is_windows() else 0
    print("Number of Data Loading Workers:", num_data_loading_workers)
```

```
    train_data_loader = DataLoader(
        dataset=f_mnist_train, batch_size=wandb.config.batch_size,
shuffle=True,
        pin_memory=True, num_workers=num_data_loading_workers
    )

    validation_data_loader = DataLoader(
        dataset=f_mnist_validation,
batch_size=wandb.config.batch_size,
        pin_memory=True, num_workers=num_data_loading_workers
    )

    return train_data_loader, validation_data_loader


def train_and_evaluate(epochs, train_data_loader,
validation_data_loader):
    device = torch.device("cuda" if torch.cuda.is_available() else
"cpu")

    # CNN    (Keras   )
    model = nn.Sequential(
        nn.Conv2d(1, 64, kernel_size=5, padding="same"),
        nn.ReLU(),
        nn.MaxPool2d(kernel_size=2),  # Output: (64, 14, 14)

        nn.Conv2d(64, 128, kernel_size=5, padding="same"),
        nn.ReLU(),
        nn.MaxPool2d(kernel_size=2),  # Output: (128, 7, 7)

        nn.Conv2d(128, 256, kernel_size=5, padding="same"),
        nn.ReLU(),
        nn.MaxPool2d(kernel_size=2),  # Output: (256, 3, 3)

        nn.Flatten(),                # Flatten for Fully Connected
Layer
        nn.Linear(256 * 3 * 3, 256), # Fully Connected Layer
        nn.ReLU(),
        nn.Linear(256, 10)           # Output Layer for 10 Classes
    ).to(device)

    optimizer = Adam(model.parameters(),
lr=wandb.config.learning_rate)
    criterion = nn.CrossEntropyLoss()

    for epoch in range(1, epochs + 1):
        # Training
        model.train()
        train_loss, train_correct, train_total = 0, 0, 0
```

```
        for images, labels in train_data_loader:
            images, labels = images.to(device), labels.to(device)
            optimizer.zero_grad()

            outputs = model(images)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()

            train_loss += loss.item()
            _, predicted = torch.max(outputs, 1)
            train_total += labels.size(0)
            train_correct += (predicted == labels).sum().item()

        train_accuracy = train_correct / train_total
        train_loss /= len(train_data_loader)

        # Validation
        model.eval()
        val_loss, val_correct, val_total = 0, 0, 0
        with torch.no_grad():
            for images, labels in validation_data_loader:
                images, labels = images.to(device), labels.to(device)

                outputs = model(images)
                loss = criterion(outputs, labels)

                val_loss += loss.item()
                _, predicted = torch.max(outputs, 1)
                val_total += labels.size(0)
                val_correct += (predicted == labels).sum().item()

        val_accuracy = val_correct / val_total
        val_loss /= len(validation_data_loader)

        # Log metrics to wandb
        wandb.log({
            "epoch": epoch,
            "train_loss": train_loss,
            "train_accuracy": train_accuracy,
            "val_loss": val_loss,
            "val_accuracy": val_accuracy
        })

        print(f"Epoch [{epoch}/{epochs}] - Train Loss:
{train_loss:.4f}, Train Acc: {train_accuracy:.4f}, "
              f"Val Loss: {val_loss:.4f}, Val Acc:
{val_accuracy:.4f}")
```

```python
if __name__ == "__main__":
    config = {"batch_size": 64, "learning_rate": 0.0005, "epochs": 25}
    wandb.init(mode="disabled", config=config)

    train_data_loader, validation_data_loader =
get_fashion_mnist_data()

    # Start training and evaluating
    train_and_evaluate(wandb.config.epochs, train_data_loader,
validation_data_loader)

    wandb.finish()
```

```
/home/work/DL
Num Train Samples:  55000
Num Validation Samples:  5000
Sample Shape:  torch.Size([1, 28, 28])
Number of Data Loading Workers: 2
Epoch [1/25] - Train Loss: 0.4332, Train Acc: 0.8412, Val Loss:
0.2992, Val Acc: 0.8882
Epoch [2/25] - Train Loss: 0.2721, Train Acc: 0.8996, Val Loss:
0.2647, Val Acc: 0.8976
Epoch [3/25] - Train Loss: 0.2205, Train Acc: 0.9179, Val Loss:
0.2585, Val Acc: 0.9022
Epoch [4/25] - Train Loss: 0.1840, Train Acc: 0.9329, Val Loss:
0.2224, Val Acc: 0.9188
Epoch [5/25] - Train Loss: 0.1539, Train Acc: 0.9436, Val Loss:
0.2132, Val Acc: 0.9212
Epoch [6/25] - Train Loss: 0.1259, Train Acc: 0.9523, Val Loss:
0.2209, Val Acc: 0.9238
Epoch [7/25] - Train Loss: 0.0999, Train Acc: 0.9629, Val Loss:
0.2361, Val Acc: 0.9170
Epoch [8/25] - Train Loss: 0.0811, Train Acc: 0.9691, Val Loss:
0.2755, Val Acc: 0.9194
Epoch [9/25] - Train Loss: 0.0642, Train Acc: 0.9755, Val Loss:
0.2852, Val Acc: 0.9188
Epoch [10/25] - Train Loss: 0.0542, Train Acc: 0.9799, Val Loss:
0.2847, Val Acc: 0.9230
Epoch [11/25] - Train Loss: 0.0457, Train Acc: 0.9832, Val Loss:
0.3045, Val Acc: 0.9220
Epoch [12/25] - Train Loss: 0.0429, Train Acc: 0.9840, Val Loss:
0.3461, Val Acc: 0.9162
Epoch [13/25] - Train Loss: 0.0360, Train Acc: 0.9866, Val Loss:
0.4099, Val Acc: 0.9168
Epoch [14/25] - Train Loss: 0.0321, Train Acc: 0.9883, Val Loss:
0.3390, Val Acc: 0.9226
Epoch [15/25] - Train Loss: 0.0323, Train Acc: 0.9877, Val Loss:
0.3932, Val Acc: 0.9210
Epoch [16/25] - Train Loss: 0.0248, Train Acc: 0.9908, Val Loss:
0.3715, Val Acc: 0.9184
```

```
Epoch [17/25] - Train Loss: 0.0255, Train Acc: 0.9904, Val Loss:
0.4151, Val Acc: 0.9182
Epoch [18/25] - Train Loss: 0.0274, Train Acc: 0.9899, Val Loss:
0.4113, Val Acc: 0.9204
Epoch [19/25] - Train Loss: 0.0186, Train Acc: 0.9936, Val Loss:
0.4480, Val Acc: 0.9216
Epoch [20/25] - Train Loss: 0.0195, Train Acc: 0.9935, Val Loss:
0.4914, Val Acc: 0.9200
Epoch [21/25] - Train Loss: 0.0204, Train Acc: 0.9931, Val Loss:
0.4794, Val Acc: 0.9190
Epoch [22/25] - Train Loss: 0.0214, Train Acc: 0.9925, Val Loss:
0.4515, Val Acc: 0.9212
Epoch [23/25] - Train Loss: 0.0180, Train Acc: 0.9938, Val Loss:
0.4825, Val Acc: 0.9202
Epoch [24/25] - Train Loss: 0.0173, Train Acc: 0.9942, Val Loss:
0.4377, Val Acc: 0.9208
Epoch [25/25] - Train Loss: 0.0170, Train Acc: 0.9939, Val Loss:
0.5480, Val Acc: 0.9198

#      val acc
#  93.7  23 ?
from pathlib import Path
import os
import torch
import wandb
from torch import nn
from torch.utils.data import DataLoader, random_split
from torchvision import datasets, transforms
from torch.optim import Adam
from torch.optim.lr_scheduler import StepLR

# Use the current working directory
BASE_PATH = Path().resolve()

def get_fashion_mnist_data():
    data_path = os.path.join(BASE_PATH, "_00_data", "j_fashion_mnist")

    # ㄷㅐㅇㅣㅌㅓ ㅈㅡㅇㄱㅏㅇㅊㅜㄱ가
    transform = transforms.Compose([
        transforms.RandomHorizontalFlip(p=0.5),  # 50%
        transforms.RandomRotation(15),          # -15  15
        transforms.ToTensor(),
        transforms.Normalize(mean=0.286, std=0.353)  #
    ])

    f_mnist_train = datasets.FashionMNIST(data_path, train=True,
download=True, transform=transform)
    f_mnist_train, f_mnist_validation = random_split(f_mnist_train,
[55_000, 5_000])
```

```
    train_data_loader = DataLoader(
        dataset=f_mnist_train, batch_size=wandb.config.batch_size,
shuffle=True, pin_memory=True, num_workers=0
    )

    validation_data_loader = DataLoader(
        dataset=f_mnist_validation,
batch_size=wandb.config.batch_size, pin_memory=True, num_workers=0
    )

    return train_data_loader, validation_data_loader


class CNNModel(nn.Module):
    def __init__(self):
        super(CNNModel, self).__init__()
        self.conv_layers = nn.Sequential(
            nn.Conv2d(1, 64, kernel_size=5, padding="same"),
            nn.BatchNorm2d(64),  # Batch Normalization 추가
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2),  # Output: (64, 14, 14)

            nn.Conv2d(64, 128, kernel_size=5, padding="same"),
            nn.BatchNorm2d(128),  # Batch Normalization 추가
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2),  # Output: (128, 7, 7)

            nn.Conv2d(128, 256, kernel_size=5, padding="same"),
            nn.BatchNorm2d(256),  # Batch Normalization 추가
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2)   # Output: (256, 3, 3)
        )

        self.fc_layers = nn.Sequential(
            nn.Flatten(),                               # Flatten for
Fully Connected Layer
            nn.Linear(256 * 3 * 3, 256),            # Fully Connected
Layer
            nn.ReLU(),
            nn.Dropout(0.2),                            # Dropout 추가
            nn.Linear(256, 10)                          # Output Layer for
10 Classes
        )

    def forward(self, x):
        x = self.conv_layers(x)
        x = self.fc_layers(x)
        return x
```

```
def train_and_evaluate(epochs, train_data_loader,
validation_data_loader):
    device = torch.device("cuda" if torch.cuda.is_available() else
"cpu")

    # Model ㅈㅓㅇㅇㅢㅁㅣ ㅊㅏㅈㅜㅇㅊㅣ / ㅊㅗㄱㅣㅎㅏ
    model = CNNModel().to(device)
    model.apply(initialize_weights)  # ㅏㅈㅜㅇㅊㅣ / ㅊㅗㄱㅣㅎㅏ

    optimizer = Adam(model.parameters(),
lr=wandb.config.learning_rate)
    scheduler = StepLR(optimizer, step_size=10, gamma=0.1)  # 10 epoch
  90%
    criterion = nn.CrossEntropyLoss()

    for epoch in range(1, epochs + 1):
        # Training
        model.train()
        train_loss, train_correct, train_total = 0, 0, 0
        for images, labels in train_data_loader:
            images, labels = images.to(device), labels.to(device)
            optimizer.zero_grad()

            outputs = model(images)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()

            train_loss += loss.item()
            _, predicted = torch.max(outputs, 1)
            train_total += labels.size(0)
            train_correct += (predicted == labels).sum().item()

        scheduler.step()  #

        train_accuracy = train_correct / train_total
        train_loss /= len(train_data_loader)

        # Validation
        model.eval()
        val_loss, val_correct, val_total = 0, 0, 0
        with torch.no_grad():
            for images, labels in validation_data_loader:
                images, labels = images.to(device), labels.to(device)

                outputs = model(images)
                loss = criterion(outputs, labels)

                val_loss += loss.item()
                _, predicted = torch.max(outputs, 1)
```

```
                val_total += labels.size(0)
                val_correct += (predicted == labels).sum().item()

        val_accuracy = val_correct / val_total
        val_loss /= len(validation_data_loader)

        # Log metrics to wandb
        wandb.log({
            "epoch": epoch,
            "train_loss": train_loss,
            "train_accuracy": train_accuracy,
            "val_loss": val_loss,
            "val_accuracy": val_accuracy
        })

        print(f"Epoch [{epoch}/{epochs}] - Train Loss:
{train_loss:.4f}, Train Acc: {train_accuracy:.4f}, "
              f"Val Loss: {val_loss:.4f}, Val Acc:
{val_accuracy:.4f}")


# 가즈ㅜㅇㅊ / ㅊㅗㄱ / ㅎㅛㅎㅏㅁㅅㅜ
def initialize_weights(m):
    if isinstance(m, nn.Conv2d) or isinstance(m, nn.Linear):
        nn.init.kaiming_normal_(m.weight)  # He Initialization


if __name__ == "__main__":
    config = {"batch_size": 64, "learning_rate": 0.0005, "epochs":
100}
    wandb.init(project="fashion-mnist-project", config=config,
name="64, 0.0005,100")

    train_data_loader, validation_data_loader =
get_fashion_mnist_data()

    # Start training and evaluating
    train_and_evaluate(wandb.config.epochs, train_data_loader,
validation_data_loader)

    wandb.finish()
```

```
<IPython.core.display.HTML object>
```

```
{"model_id":"","version_major":2,"version_minor":0}
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>

{"model_id":"74d4f8f440354732a6f6b4f931f42a76","version_major":2,"vers
ion_minor":0}

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

Epoch [1/100] - Train Loss: 0.5497, Train Acc: 0.8029, Val Loss:
0.3988, Val Acc: 0.8482
Epoch [2/100] - Train Loss: 0.3792, Train Acc: 0.8609, Val Loss:
0.3348, Val Acc: 0.8748
Epoch [3/100] - Train Loss: 0.3306, Train Acc: 0.8780, Val Loss:
0.3082, Val Acc: 0.8862
Epoch [4/100] - Train Loss: 0.3020, Train Acc: 0.8894, Val Loss:
0.3225, Val Acc: 0.8790
Epoch [5/100] - Train Loss: 0.2817, Train Acc: 0.8973, Val Loss:
0.2590, Val Acc: 0.9016
Epoch [6/100] - Train Loss: 0.2653, Train Acc: 0.9028, Val Loss:
0.2721, Val Acc: 0.9008
Epoch [7/100] - Train Loss: 0.2530, Train Acc: 0.9072, Val Loss:
0.2521, Val Acc: 0.9070
Epoch [8/100] - Train Loss: 0.2437, Train Acc: 0.9086, Val Loss:
0.2350, Val Acc: 0.9090
Epoch [9/100] - Train Loss: 0.2317, Train Acc: 0.9151, Val Loss:
0.2413, Val Acc: 0.9112
Epoch [10/100] - Train Loss: 0.2234, Train Acc: 0.9175, Val Loss:
0.2353, Val Acc: 0.9110
Epoch [11/100] - Train Loss: 0.1876, Train Acc: 0.9309, Val Loss:
0.2050, Val Acc: 0.9224
Epoch [12/100] - Train Loss: 0.1734, Train Acc: 0.9356, Val Loss:
0.1964, Val Acc: 0.9278
Epoch [13/100] - Train Loss: 0.1703, Train Acc: 0.9372, Val Loss:
0.2011, Val Acc: 0.9260
Epoch [14/100] - Train Loss: 0.1671, Train Acc: 0.9386, Val Loss:
0.2003, Val Acc: 0.9282
Epoch [15/100] - Train Loss: 0.1645, Train Acc: 0.9392, Val Loss:
0.1972, Val Acc: 0.9322
Epoch [16/100] - Train Loss: 0.1599, Train Acc: 0.9407, Val Loss:
0.1973, Val Acc: 0.9288
Epoch [17/100] - Train Loss: 0.1559, Train Acc: 0.9423, Val Loss:
0.1959, Val Acc: 0.9318
Epoch [18/100] - Train Loss: 0.1541, Train Acc: 0.9436, Val Loss:
0.1937, Val Acc: 0.9336
```

```
Epoch [19/100] - Train Loss: 0.1519, Train Acc: 0.9434, Val Loss:
0.1870, Val Acc: 0.9336
Epoch [20/100] - Train Loss: 0.1489, Train Acc: 0.9448, Val Loss:
0.1952, Val Acc: 0.9326
Epoch [21/100] - Train Loss: 0.1437, Train Acc: 0.9468, Val Loss:
0.1844, Val Acc: 0.9368
Epoch [22/100] - Train Loss: 0.1447, Train Acc: 0.9464, Val Loss:
0.1831, Val Acc: 0.9362
Epoch [23/100] - Train Loss: 0.1418, Train Acc: 0.9471, Val Loss:
0.1894, Val Acc: 0.9324
Epoch [24/100] - Train Loss: 0.1414, Train Acc: 0.9478, Val Loss:
0.1843, Val Acc: 0.9348
Epoch [25/100] - Train Loss: 0.1419, Train Acc: 0.9476, Val Loss:
0.1894, Val Acc: 0.9318
Epoch [26/100] - Train Loss: 0.1425, Train Acc: 0.9474, Val Loss:
0.1901, Val Acc: 0.9326
Epoch [27/100] - Train Loss: 0.1407, Train Acc: 0.9486, Val Loss:
0.1889, Val Acc: 0.9320
Epoch [28/100] - Train Loss: 0.1415, Train Acc: 0.9479, Val Loss:
0.1819, Val Acc: 0.9358
Epoch [29/100] - Train Loss: 0.1404, Train Acc: 0.9479, Val Loss:
0.1874, Val Acc: 0.9330
Epoch [30/100] - Train Loss: 0.1393, Train Acc: 0.9485, Val Loss:
0.1904, Val Acc: 0.9296
Epoch [31/100] - Train Loss: 0.1387, Train Acc: 0.9490, Val Loss:
0.1811, Val Acc: 0.9350
Epoch [32/100] - Train Loss: 0.1388, Train Acc: 0.9483, Val Loss:
0.1838, Val Acc: 0.9346
Epoch [33/100] - Train Loss: 0.1403, Train Acc: 0.9482, Val Loss:
0.1835, Val Acc: 0.9354
Epoch [34/100] - Train Loss: 0.1390, Train Acc: 0.9480, Val Loss:
0.1900, Val Acc: 0.9326
Epoch [35/100] - Train Loss: 0.1401, Train Acc: 0.9482, Val Loss:
0.1887, Val Acc: 0.9314
Epoch [36/100] - Train Loss: 0.1400, Train Acc: 0.9476, Val Loss:
0.1878, Val Acc: 0.9334

---------------------------------------------------------------------
-----
KeyboardInterrupt                         Traceback (most recent call
last)
Cell In[5], line 153
    150 train_data_loader, validation_data_loader =
get_fashion_mnist_data()
    152 # Start training and evaluating
--> 153 train_and_evaluate(wandb.config.epochs, train_data_loader,
validation_data_loader)
    155 wandb.finish()

Cell In[5], line 90, in train_and_evaluate(epochs, train_data_loader,
```

```
validation_data_loader)
     88 model.train()
     89 train_loss, train_correct, train_total = 0, 0, 0
---> 90 for images, labels in train_data_loader:
     91     images, labels = images.to(device), labels.to(device)
     92     optimizer.zero_grad()

File
/usr/local/lib/python3.10/dist-packages/torch/utils/data/dataloader.py
:631, in _BaseDataLoaderIter.__next__(self)
    628 if self._sampler_iter is None:
    629     # TODO(https://github.com/pytorch/pytorch/issues/76750)
    630     self._reset()  # type: ignore[call-arg]
--> 631 data = self._next_data()
    632 self._num_yielded += 1
    633 if self._dataset_kind == _DatasetKind.Iterable and \
    634         self._IterableDataset_len_called is not None and \
    635         self._num_yielded > self._IterableDataset_len_called:

File
/usr/local/lib/python3.10/dist-packages/torch/utils/data/dataloader.py
:675, in _SingleProcessDataLoaderIter._next_data(self)
    673 def _next_data(self):
    674     index = self._next_index()  # may raise StopIteration
--> 675     data = self._dataset_fetcher.fetch(index)  # may raise
StopIteration
    676     if self._pin_memory:
    677         data = _utils.pin_memory.pin_memory(data,
self._pin_memory_device)

File
/usr/local/lib/python3.10/dist-packages/torch/utils/data/_utils/fetch.
py:49, in _MapDatasetFetcher.fetch(self, possibly_batched_index)
     47 if self.auto_collation:
     48     if hasattr(self.dataset, "__getitems__") and
self.dataset.__getitems__:
---> 49         data =
self.dataset.__getitems__(possibly_batched_index)
     50     else:
     51         data = [self.dataset[idx] for idx in
possibly_batched_index]

File
/usr/local/lib/python3.10/dist-packages/torch/utils/data/dataset.py:41
9, in Subset.__getitems__(self, indices)
    417     return self.dataset.__getitems__([self.indices[idx] for
idx in indices])  # type: ignore[attr-defined]
    418 else:
--> 419     return [self.dataset[self.indices[idx]] for idx in
indices]
```

```
File
/usr/local/lib/python3.10/dist-packages/torch/utils/data/dataset.py:41
9, in <listcomp>(.0)
    417     return self.dataset.__getitems__([self.indices[idx] for
idx in indices])  # type: ignore[attr-defined]
    418 else:
--> 419     return [self.dataset[self.indices[idx]] for idx in
indices]

File
/usr/local/lib/python3.10/dist-packages/torchvision/datasets/mnist.py:
145, in MNIST.__getitem__(self, index)
    142 img = Image.fromarray(img.numpy(), mode="L")
    144 if self.transform is not None:
--> 145     img = self.transform(img)
    147 if self.target_transform is not None:
    148     target = self.target_transform(target)

File
/usr/local/lib/python3.10/dist-packages/torchvision/transforms/transfo
rms.py:95, in Compose.__call__(self, img)
     93 def __call__(self, img):
     94     for t in self.transforms:
---> 95         img = t(img)
     96     return img

File
/usr/local/lib/python3.10/dist-packages/torch/nn/modules/module.py:151
1, in Module._wrapped_call_impl(self, *args, **kwargs)
   1509     return self._compiled_call_impl(*args, **kwargs)  # type:
ignore[misc]
   1510 else:
-> 1511     return self._call_impl(*args, **kwargs)

File
/usr/local/lib/python3.10/dist-packages/torch/nn/modules/module.py:152
0, in Module._call_impl(self, *args, **kwargs)
   1515 # If we don't have any hooks, we want to skip the rest of the
logic in
   1516 # this function, and just call forward.
   1517 if not (self._backward_hooks or self._backward_pre_hooks or
self._forward_hooks or self._forward_pre_hooks
   1518         or _global_backward_pre_hooks or
_global_backward_hooks
   1519         or _global_forward_hooks or
_global_forward_pre_hooks):
-> 1520     return forward_call(*args, **kwargs)
   1522 try:
   1523     result = None
```

```
File
/usr/local/lib/python3.10/dist-packages/torchvision/transforms/transfo
rms.py:1370, in RandomRotation.forward(self, img)
   1368     else:
   1369         fill = [float(f) for f in fill]
-> 1370 angle = self.get_params(self.degrees)
   1372 return F.rotate(img, angle, self.interpolation, self.expand,
self.center, fill)

KeyboardInterrupt:
```

```python
from pathlib import Path
import os
import torch
import wandb
from torch import nn
from torch.utils.data import DataLoader, random_split
from torchvision import datasets, transforms
from torch.optim import Adam
from torch.optim.lr_scheduler import StepLR

# Use the current working directory
BASE_PATH = Path().resolve()

def get_fashion_mnist_data():
    data_path = os.path.join(BASE_PATH, "_00_data", "j_fashion_mnist")

    # ㄷㅐㅇㅣㅌㅓㅈㅡㅇㄱㅏㅇㅊㅜ가
    transform = transforms.Compose([
        transforms.RandomHorizontalFlip(p=0.5),  # 50%
        transforms.RandomRotation(15),          # -15  15
        transforms.ToTensor(),
        transforms.Normalize(mean=0.286, std=0.353)  #
    ])

    f_mnist_train = datasets.FashionMNIST(data_path, train=True,
download=True, transform=transform)
    f_mnist_train, f_mnist_validation = random_split(f_mnist_train,
[55_000, 5_000])

    train_data_loader = DataLoader(
        dataset=f_mnist_train, batch_size=wandb.config.batch_size,
shuffle=True, pin_memory=True, num_workers=0
    )

    validation_data_loader = DataLoader(
        dataset=f_mnist_validation,
batch_size=wandb.config.batch_size, pin_memory=True, num_workers=0
    )
```

```
    return train_data_loader, validation_data_loader


def get_fashion_mnist_test_data():
    data_path = os.path.join(BASE_PATH, "_00_data", "j_fashion_mnist")
    transform = transforms.Compose([
        transforms.ToTensor(),
        transforms.Normalize(mean=0.286, std=0.353)  #
    ])
    f_mnist_test = datasets.FashionMNIST(data_path, train=False,
download=True, transform=transform)

    test_data_loader = DataLoader(
        dataset=f_mnist_test, batch_size=wandb.config.batch_size,
pin_memory=True, num_workers=0
    )
    return test_data_loader


class CNNModel(nn.Module):
    def __init__(self):
        super(CNNModel, self).__init__()
        self.conv_layers = nn.Sequential(
            nn.Conv2d(1, 64, kernel_size=5, padding="same"),
            nn.BatchNorm2d(64),  # Batch Normalization 추가
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2),  # Output: (64, 14, 14)

            nn.Conv2d(64, 128, kernel_size=5, padding="same"),
            nn.BatchNorm2d(128),  # Batch Normalization 추가
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2),  # Output: (128, 7, 7)

            nn.Conv2d(128, 256, kernel_size=5, padding="same"),
            nn.BatchNorm2d(256),  # Batch Normalization 추가
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2)   # Output: (256, 3, 3)
        )

        self.fc_layers = nn.Sequential(
            nn.Flatten(),                           # Flatten for
Fully Connected Layer
            nn.Linear(256 * 3 * 3, 256),            # Fully Connected
Layer
            nn.ReLU(),
            nn.Dropout(0.2),                        # Dropout 추가
            nn.Linear(256, 10)                      # Output Layer for
10 Classes
        )
```

```
    def forward(self, x):
        x = self.conv_layers(x)
        x = self.fc_layers(x)
        return x


def train_and_evaluate(epochs, train_data_loader,
validation_data_loader):
    device = torch.device("cuda" if torch.cuda.is_available() else
"cpu")

    # Model ㅈㅓㅇㅇㅢ ㅁㅣㅊ가ㅈㅜㅇㅊㅣ ㅊㅗㄱㅣㅎㅛ
    model = CNNModel().to(device)
    model.apply(initialize_weights)  # 가ㅈㅜㅇㅊㅣ ㅊㅗㄱㅣㅎㅛ

    optimizer = Adam(model.parameters(),
lr=wandb.config.learning_rate)
    scheduler = StepLR(optimizer, step_size=10, gamma=0.1)  # 10 epoch
   90%
    criterion = nn.CrossEntropyLoss()

    for epoch in range(1, epochs + 1):
        # Training
        model.train()
        train_loss, train_correct, train_total = 0, 0, 0
        for images, labels in train_data_loader:
            images, labels = images.to(device), labels.to(device)
            optimizer.zero_grad()

            outputs = model(images)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()

            train_loss += loss.item()
            _, predicted = torch.max(outputs, 1)
            train_total += labels.size(0)
            train_correct += (predicted == labels).sum().item()

        scheduler.step()  #

        train_accuracy = train_correct / train_total
        train_loss /= len(train_data_loader)

        # Validation
        model.eval()
        val_loss, val_correct, val_total = 0, 0, 0
        with torch.no_grad():
            for images, labels in validation_data_loader:
```

```
                images, labels = images.to(device), labels.to(device)

                outputs = model(images)
                loss = criterion(outputs, labels)

                val_loss += loss.item()
                _, predicted = torch.max(outputs, 1)
                val_total += labels.size(0)
                val_correct += (predicted == labels).sum().item()

        val_accuracy = val_correct / val_total
        val_loss /= len(validation_data_loader)

        # Log metrics to wandb
        wandb.log({
            "epoch": epoch,
            "train_loss": train_loss,
            "train_accuracy": train_accuracy,
            "val_loss": val_loss,
            "val_accuracy": val_accuracy
        })

        print(f"Epoch [{epoch}/{epochs}] - Train Loss:
{train_loss:.4f}, Train Acc: {train_accuracy:.4f}, "
              f"Val Loss: {val_loss:.4f}, Val Acc:
{val_accuracy:.4f}")

    return model  #


def evaluate_test_data(model, test_data_loader, device):
    model.eval()
    test_correct, test_total = 0, 0
    with torch.no_grad():
        for images, labels in test_data_loader:
            images, labels = images.to(device), labels.to(device)

            outputs = model(images)
            _, predicted = torch.max(outputs, 1)
            test_total += labels.size(0)
            test_correct += (predicted == labels).sum().item()

    test_accuracy = test_correct / test_total
    return test_accuracy


# 가즈ㅜㅇㅊ / ㅊㅗㄱ / ㅎㅐㅎㅏㅁㅅㅜ
def initialize_weights(m):
    if isinstance(m, nn.Conv2d) or isinstance(m, nn.Linear):
        nn.init.kaiming_normal_(m.weight)  # He Initialization
```

```
if __name__ == "__main__":
    config = {"batch_size": 64, "learning_rate": 0.0005, "epochs": 50}
    wandb.init(project="fashion-mnist-project", config=config,
name="64, 0.0005,100,tstAcc")

    train_data_loader, validation_data_loader =
get_fashion_mnist_data()
    test_data_loader = get_fashion_mnist_test_data()

    # Start training and evaluating
    trained_model = train_and_evaluate(wandb.config.epochs,
train_data_loader, validation_data_loader)

    # Evaluate on test data
    device = torch.device("cuda" if torch.cuda.is_available() else
"cpu")
    test_accuracy = evaluate_test_data(trained_model,
test_data_loader, device)

    # Log test accuracy to wandb
    wandb.log({"test_accuracy": test_accuracy})
    print(f"Test Accuracy: {test_accuracy:.4f}")

    wandb.finish()
```

```
<IPython.core.display.HTML object>
```

```
{"model_id":"","version_major":2,"version_minor":0}
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

```
{"model_id":"6b894b1720b04ab68b6d90790a9c26e4","version_major":2,"vers
ion_minor":0}
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

```
Epoch [1/50] - Train Loss: 0.5589, Train Acc: 0.8009, Val Loss:
0.3725, Val Acc: 0.8574
Epoch [2/50] - Train Loss: 0.3834, Train Acc: 0.8595, Val Loss:
0.3178, Val Acc: 0.8812
Epoch [3/50] - Train Loss: 0.3306, Train Acc: 0.8781, Val Loss:
0.3074, Val Acc: 0.8800
Epoch [4/50] - Train Loss: 0.3072, Train Acc: 0.8881, Val Loss:
0.2890, Val Acc: 0.8918
Epoch [5/50] - Train Loss: 0.2843, Train Acc: 0.8954, Val Loss:
0.2773, Val Acc: 0.8952
Epoch [6/50] - Train Loss: 0.2661, Train Acc: 0.9014, Val Loss:
0.2615, Val Acc: 0.9016
Epoch [7/50] - Train Loss: 0.2557, Train Acc: 0.9062, Val Loss:
0.2465, Val Acc: 0.9054
Epoch [8/50] - Train Loss: 0.2421, Train Acc: 0.9100, Val Loss:
0.2923, Val Acc: 0.8932
Epoch [9/50] - Train Loss: 0.2344, Train Acc: 0.9139, Val Loss:
0.2584, Val Acc: 0.8972
Epoch [10/50] - Train Loss: 0.2258, Train Acc: 0.9161, Val Loss:
0.2375, Val Acc: 0.9126
Epoch [11/50] - Train Loss: 0.1864, Train Acc: 0.9318, Val Loss:
0.1996, Val Acc: 0.9228
Epoch [12/50] - Train Loss: 0.1754, Train Acc: 0.9357, Val Loss:
0.2048, Val Acc: 0.9248
Epoch [13/50] - Train Loss: 0.1714, Train Acc: 0.9370, Val Loss:
0.1951, Val Acc: 0.9294
Epoch [14/50] - Train Loss: 0.1665, Train Acc: 0.9388, Val Loss:
0.2020, Val Acc: 0.9276
Epoch [15/50] - Train Loss: 0.1618, Train Acc: 0.9402, Val Loss:
0.1962, Val Acc: 0.9278
Epoch [16/50] - Train Loss: 0.1613, Train Acc: 0.9405, Val Loss:
0.1960, Val Acc: 0.9284
Epoch [17/50] - Train Loss: 0.1576, Train Acc: 0.9414, Val Loss:
0.1913, Val Acc: 0.9280
Epoch [18/50] - Train Loss: 0.1535, Train Acc: 0.9437, Val Loss:
0.1964, Val Acc: 0.9286
Epoch [19/50] - Train Loss: 0.1524, Train Acc: 0.9431, Val Loss:
0.1928, Val Acc: 0.9310
Epoch [20/50] - Train Loss: 0.1508, Train Acc: 0.9436, Val Loss:
0.1923, Val Acc: 0.9314
Epoch [21/50] - Train Loss: 0.1444, Train Acc: 0.9471, Val Loss:
0.1888, Val Acc: 0.9324
Epoch [22/50] - Train Loss: 0.1437, Train Acc: 0.9462, Val Loss:
0.1951, Val Acc: 0.9270
Epoch [23/50] - Train Loss: 0.1415, Train Acc: 0.9474, Val Loss:
0.1974, Val Acc: 0.9276
Epoch [24/50] - Train Loss: 0.1404, Train Acc: 0.9478, Val Loss:
0.1871, Val Acc: 0.9304
Epoch [25/50] - Train Loss: 0.1407, Train Acc: 0.9479, Val Loss:
0.1947, Val Acc: 0.9314
```

```
Epoch [26/50] - Train Loss: 0.1424, Train Acc: 0.9472, Val Loss:
0.1809, Val Acc: 0.9370
Epoch [27/50] - Train Loss: 0.1410, Train Acc: 0.9476, Val Loss:
0.1865, Val Acc: 0.9348
Epoch [28/50] - Train Loss: 0.1417, Train Acc: 0.9477, Val Loss:
0.1848, Val Acc: 0.9330
Epoch [29/50] - Train Loss: 0.1405, Train Acc: 0.9478, Val Loss:
0.1880, Val Acc: 0.9344
Epoch [30/50] - Train Loss: 0.1408, Train Acc: 0.9480, Val Loss:
0.1849, Val Acc: 0.9350
Epoch [31/50] - Train Loss: 0.1399, Train Acc: 0.9491, Val Loss:
0.1906, Val Acc: 0.9332
Epoch [32/50] - Train Loss: 0.1397, Train Acc: 0.9485, Val Loss:
0.1864, Val Acc: 0.9366
Epoch [33/50] - Train Loss: 0.1398, Train Acc: 0.9489, Val Loss:
0.1919, Val Acc: 0.9310
Epoch [34/50] - Train Loss: 0.1396, Train Acc: 0.9489, Val Loss:
0.1874, Val Acc: 0.9350
Epoch [35/50] - Train Loss: 0.1398, Train Acc: 0.9482, Val Loss:
0.1909, Val Acc: 0.9316
Epoch [36/50] - Train Loss: 0.1396, Train Acc: 0.9478, Val Loss:
0.1885, Val Acc: 0.9330
Epoch [37/50] - Train Loss: 0.1393, Train Acc: 0.9487, Val Loss:
0.1912, Val Acc: 0.9342
Epoch [38/50] - Train Loss: 0.1391, Train Acc: 0.9474, Val Loss:
0.1919, Val Acc: 0.9348
Epoch [39/50] - Train Loss: 0.1385, Train Acc: 0.9483, Val Loss:
0.1921, Val Acc: 0.9332
Epoch [40/50] - Train Loss: 0.1398, Train Acc: 0.9484, Val Loss:
0.1820, Val Acc: 0.9344
Epoch [41/50] - Train Loss: 0.1393, Train Acc: 0.9478, Val Loss:
0.1855, Val Acc: 0.9328
Epoch [42/50] - Train Loss: 0.1391, Train Acc: 0.9484, Val Loss:
0.1931, Val Acc: 0.9348
Epoch [43/50] - Train Loss: 0.1389, Train Acc: 0.9478, Val Loss:
0.1963, Val Acc: 0.9318
Epoch [44/50] - Train Loss: 0.1392, Train Acc: 0.9488, Val Loss:
0.1943, Val Acc: 0.9338
Epoch [45/50] - Train Loss: 0.1396, Train Acc: 0.9481, Val Loss:
0.1890, Val Acc: 0.9342
Epoch [46/50] - Train Loss: 0.1381, Train Acc: 0.9487, Val Loss:
0.1879, Val Acc: 0.9340
Epoch [47/50] - Train Loss: 0.1389, Train Acc: 0.9484, Val Loss:
0.1957, Val Acc: 0.9296
Epoch [48/50] - Train Loss: 0.1374, Train Acc: 0.9490, Val Loss:
0.1888, Val Acc: 0.9308
Epoch [49/50] - Train Loss: 0.1404, Train Acc: 0.9485, Val Loss:
0.1909, Val Acc: 0.9330
Epoch [50/50] - Train Loss: 0.1383, Train Acc: 0.9481, Val Loss:
```

```
0.1912, Val Acc: 0.9292
Test Accuracy: 0.9293

{"model_id":"","version_major":2,"version_minor":0}

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>
```

□□ googlene □□ □□□ □□ □□□
ㅎㅏㅅㅕㅆㄷㅓㄴ ㅁㅏㄹㅇㅣ ㄱㅣㅇㅓㄱㄴㅏㅅㅓ ㅊㅜ가ㅎㅏㅁ...

..?

```
from torchvision import datasets, transforms, models  # GoogLeNet

googlenet = models.googlenet(pretrained=False, aux_logits=False)
print("\nGoogLeNet Model Structure:\n")
print(googlenet)


GoogLeNet Model Structure:

GoogLeNet(
  (conv1): BasicConv2d(
    (conv): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2),
padding=(3, 3), bias=False)
    (bn): BatchNorm2d(64, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
  )
  (maxpool1): MaxPool2d(kernel_size=3, stride=2, padding=0,
dilation=1, ceil_mode=True)
  (conv2): BasicConv2d(
    (conv): Conv2d(64, 64, kernel_size=(1, 1), stride=(1, 1),
bias=False)
    (bn): BatchNorm2d(64, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
  )
  (conv3): BasicConv2d(
    (conv): Conv2d(64, 192, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
    (bn): BatchNorm2d(192, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
  )
  (maxpool2): MaxPool2d(kernel_size=3, stride=2, padding=0,
dilation=1, ceil_mode=True)
  (inception3a): Inception(
    (branch1): BasicConv2d(
```

```
      (conv): Conv2d(192, 64, kernel_size=(1, 1), stride=(1, 1),
bias=False)
      (bn): BatchNorm2d(64, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
    )
    (branch2): Sequential(
      (0): BasicConv2d(
        (conv): Conv2d(192, 96, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (bn): BatchNorm2d(96, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
      )
      (1): BasicConv2d(
        (conv): Conv2d(96, 128, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
        (bn): BatchNorm2d(128, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (branch3): Sequential(
      (0): BasicConv2d(
        (conv): Conv2d(192, 16, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (bn): BatchNorm2d(16, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
      )
      (1): BasicConv2d(
        (conv): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
        (bn): BatchNorm2d(32, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (branch4): Sequential(
      (0): MaxPool2d(kernel_size=3, stride=1, padding=1, dilation=1,
ceil_mode=True)
      (1): BasicConv2d(
        (conv): Conv2d(192, 32, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (bn): BatchNorm2d(32, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
  )
  (inception3b): Inception(
    (branch1): BasicConv2d(
      (conv): Conv2d(256, 128, kernel_size=(1, 1), stride=(1, 1),
bias=False)
      (bn): BatchNorm2d(128, eps=0.001, momentum=0.1, affine=True,
```

```
track_running_stats=True)
    )
    (branch2): Sequential(
      (0): BasicConv2d(
        (conv): Conv2d(256, 128, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (bn): BatchNorm2d(128, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
      )
      (1): BasicConv2d(
        (conv): Conv2d(128, 192, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
        (bn): BatchNorm2d(192, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (branch3): Sequential(
      (0): BasicConv2d(
        (conv): Conv2d(256, 32, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (bn): BatchNorm2d(32, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
      )
      (1): BasicConv2d(
        (conv): Conv2d(32, 96, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
        (bn): BatchNorm2d(96, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (branch4): Sequential(
      (0): MaxPool2d(kernel_size=3, stride=1, padding=1, dilation=1,
ceil_mode=True)
      (1): BasicConv2d(
        (conv): Conv2d(256, 64, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (bn): BatchNorm2d(64, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
  )
  (maxpool3): MaxPool2d(kernel_size=3, stride=2, padding=0,
dilation=1, ceil_mode=True)
  (inception4a): Inception(
    (branch1): BasicConv2d(
      (conv): Conv2d(480, 192, kernel_size=(1, 1), stride=(1, 1),
bias=False)
      (bn): BatchNorm2d(192, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
```

```
    )
    (branch2): Sequential(
      (0): BasicConv2d(
        (conv): Conv2d(480, 96, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (bn): BatchNorm2d(96, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
      )
      (1): BasicConv2d(
        (conv): Conv2d(96, 208, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
        (bn): BatchNorm2d(208, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (branch3): Sequential(
      (0): BasicConv2d(
        (conv): Conv2d(480, 16, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (bn): BatchNorm2d(16, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
      )
      (1): BasicConv2d(
        (conv): Conv2d(16, 48, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
        (bn): BatchNorm2d(48, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (branch4): Sequential(
      (0): MaxPool2d(kernel_size=3, stride=1, padding=1, dilation=1,
ceil_mode=True)
      (1): BasicConv2d(
        (conv): Conv2d(480, 64, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (bn): BatchNorm2d(64, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
  )
  (inception4b): Inception(
    (branch1): BasicConv2d(
      (conv): Conv2d(512, 160, kernel_size=(1, 1), stride=(1, 1),
bias=False)
      (bn): BatchNorm2d(160, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
    )
    (branch2): Sequential(
      (0): BasicConv2d(
```

```
        (conv): Conv2d(512, 112, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (bn): BatchNorm2d(112, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
      )
      (1): BasicConv2d(
        (conv): Conv2d(112, 224, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
        (bn): BatchNorm2d(224, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (branch3): Sequential(
      (0): BasicConv2d(
        (conv): Conv2d(512, 24, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (bn): BatchNorm2d(24, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
      )
      (1): BasicConv2d(
        (conv): Conv2d(24, 64, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
        (bn): BatchNorm2d(64, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (branch4): Sequential(
      (0): MaxPool2d(kernel_size=3, stride=1, padding=1, dilation=1,
ceil_mode=True)
      (1): BasicConv2d(
        (conv): Conv2d(512, 64, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (bn): BatchNorm2d(64, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
  )
  (inception4c): Inception(
    (branch1): BasicConv2d(
      (conv): Conv2d(512, 128, kernel_size=(1, 1), stride=(1, 1),
bias=False)
      (bn): BatchNorm2d(128, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
    )
    (branch2): Sequential(
      (0): BasicConv2d(
        (conv): Conv2d(512, 128, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (bn): BatchNorm2d(128, eps=0.001, momentum=0.1, affine=True,
```

```
track_running_stats=True)
      )
      (1): BasicConv2d(
        (conv): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
        (bn): BatchNorm2d(256, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (branch3): Sequential(
      (0): BasicConv2d(
        (conv): Conv2d(512, 24, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (bn): BatchNorm2d(24, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
      )
      (1): BasicConv2d(
        (conv): Conv2d(24, 64, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
        (bn): BatchNorm2d(64, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (branch4): Sequential(
      (0): MaxPool2d(kernel_size=3, stride=1, padding=1, dilation=1,
ceil_mode=True)
      (1): BasicConv2d(
        (conv): Conv2d(512, 64, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (bn): BatchNorm2d(64, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
  )
  (inception4d): Inception(
    (branch1): BasicConv2d(
      (conv): Conv2d(512, 112, kernel_size=(1, 1), stride=(1, 1),
bias=False)
      (bn): BatchNorm2d(112, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
    )
    (branch2): Sequential(
      (0): BasicConv2d(
        (conv): Conv2d(512, 144, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (bn): BatchNorm2d(144, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
      )
      (1): BasicConv2d(
```

```
        (conv): Conv2d(144, 288, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
        (bn): BatchNorm2d(288, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (branch3): Sequential(
      (0): BasicConv2d(
        (conv): Conv2d(512, 32, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (bn): BatchNorm2d(32, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
      )
      (1): BasicConv2d(
        (conv): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
        (bn): BatchNorm2d(64, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (branch4): Sequential(
      (0): MaxPool2d(kernel_size=3, stride=1, padding=1, dilation=1,
ceil_mode=True)
      (1): BasicConv2d(
        (conv): Conv2d(512, 64, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (bn): BatchNorm2d(64, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
  )
  (inception4e): Inception(
    (branch1): BasicConv2d(
      (conv): Conv2d(528, 256, kernel_size=(1, 1), stride=(1, 1),
bias=False)
      (bn): BatchNorm2d(256, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
    )
    (branch2): Sequential(
      (0): BasicConv2d(
        (conv): Conv2d(528, 160, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (bn): BatchNorm2d(160, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
      )
      (1): BasicConv2d(
        (conv): Conv2d(160, 320, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
        (bn): BatchNorm2d(320, eps=0.001, momentum=0.1, affine=True,
```

```
track_running_stats=True)
      )
    )
    (branch3): Sequential(
      (0): BasicConv2d(
        (conv): Conv2d(528, 32, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (bn): BatchNorm2d(32, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
      )
      (1): BasicConv2d(
        (conv): Conv2d(32, 128, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
        (bn): BatchNorm2d(128, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (branch4): Sequential(
      (0): MaxPool2d(kernel_size=3, stride=1, padding=1, dilation=1,
ceil_mode=True)
      (1): BasicConv2d(
        (conv): Conv2d(528, 128, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (bn): BatchNorm2d(128, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
  )
  (maxpool4): MaxPool2d(kernel_size=2, stride=2, padding=0,
dilation=1, ceil_mode=True)
  (inception5a): Inception(
    (branch1): BasicConv2d(
      (conv): Conv2d(832, 256, kernel_size=(1, 1), stride=(1, 1),
bias=False)
      (bn): BatchNorm2d(256, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
    )
    (branch2): Sequential(
      (0): BasicConv2d(
        (conv): Conv2d(832, 160, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (bn): BatchNorm2d(160, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
      )
      (1): BasicConv2d(
        (conv): Conv2d(160, 320, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
        (bn): BatchNorm2d(320, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
```

```
      )
    )
    (branch3): Sequential(
      (0): BasicConv2d(
        (conv): Conv2d(832, 32, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (bn): BatchNorm2d(32, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
      )
      (1): BasicConv2d(
        (conv): Conv2d(32, 128, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
        (bn): BatchNorm2d(128, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (branch4): Sequential(
      (0): MaxPool2d(kernel_size=3, stride=1, padding=1, dilation=1,
ceil_mode=True)
      (1): BasicConv2d(
        (conv): Conv2d(832, 128, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (bn): BatchNorm2d(128, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
  )
  (inception5b): Inception(
    (branch1): BasicConv2d(
      (conv): Conv2d(832, 384, kernel_size=(1, 1), stride=(1, 1),
bias=False)
      (bn): BatchNorm2d(384, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
    )
    (branch2): Sequential(
      (0): BasicConv2d(
        (conv): Conv2d(832, 192, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (bn): BatchNorm2d(192, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
      )
      (1): BasicConv2d(
        (conv): Conv2d(192, 384, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
        (bn): BatchNorm2d(384, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (branch3): Sequential(
```

```
      (0): BasicConv2d(
        (conv): Conv2d(832, 48, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (bn): BatchNorm2d(48, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
      )
      (1): BasicConv2d(
        (conv): Conv2d(48, 128, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
        (bn): BatchNorm2d(128, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (branch4): Sequential(
      (0): MaxPool2d(kernel_size=3, stride=1, padding=1, dilation=1,
ceil_mode=True)
      (1): BasicConv2d(
        (conv): Conv2d(832, 128, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (bn): BatchNorm2d(128, eps=0.001, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
  (dropout): Dropout(p=0.2, inplace=False)
  (fc): Linear(in_features=1024, out_features=1000, bias=True)
)

/usr/local/lib/python3.10/dist-packages/torchvision/models/
_utils.py:208: UserWarning: The parameter 'pretrained' is deprecated
since 0.13 and may be removed in the future, please use 'weights'
instead.
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/torchvision/models/_utils.py:2
23: UserWarning: Arguments other than a weight enum or `None` for
'weights' are deprecated since 0.13 and may be removed in the future.
The current behavior is equivalent to passing `weights=None`.
  warnings.warn(msg)
/usr/local/lib/python3.10/dist-packages/torchvision/models/googlenet.p
y:47: FutureWarning: The default weight initialization of GoogleNet
will be changed in future releases of torchvision. If you wish to keep
the old behavior (which leads to long initialization times due to
scipy/scipy#11299), please set init_weights=True.
  warnings.warn(

!pip install torchinfo

Defaulting to user installation because normal site-packages is not
writeable
```

```
Looking in indexes: https://pypi.org/simple,
https://pypi.ngc.nvidia.com
Collecting torchinfo
  Downloading torchinfo-1.8.0-py3-none-any.whl.metadata (21 kB)
Downloading torchinfo-1.8.0-py3-none-any.whl (23 kB)
DEPRECATION: devscripts 2.22.1ubuntu1 has a non-standard version
number. pip 24.1 will enforce this behaviour change. A possible
replacement is to upgrade to a newer version of devscripts or contact
the author to suggest that they release a version with a conforming
version number. Discussion can be found at
https://github.com/pypa/pip/issues/12063
Installing collected packages: torchinfo
Successfully installed torchinfo-1.8.0

[notice] A new release of pip is available: 24.0 -> 24.3.1
[notice] To update, run: python -m pip install --upgrade pip
```

```python
import torch
from torch import nn
from torchinfo import summary  # torchinfo import

# Define CNN model
class SimpleCNN(nn.Module):
    def __init__(self):
        super(SimpleCNN, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, kernel_size=3, padding=1)
        self.conv2 = nn.Conv2d(32, 64, kernel_size=3, padding=1)
        self.pool = nn.MaxPool2d(kernel_size=2)
        self.fc1 = nn.Linear(64 * 14 * 14, 128)
        self.fc2 = nn.Linear(128, 10)

    def forward(self, x):
        x = torch.relu(self.conv1(x))
        x = self.pool(torch.relu(self.conv2(x)))
        x = x.view(x.size(0), -1)
        x = torch.relu(self.fc1(x))
        x = self.fc2(x)
        return x

if __name__ == "__main__":
    # Initialize the model
    model = SimpleCNN()

    # Print model summary using torchinfo
    input_size = (128, 1, 28, 28)  # batch_size=128, channels=1,
height=28, width=28
    print(summary(model, input_size=input_size,
col_names=["input_size", "output_size", "num_params", "trainable"]))
```

```
==========================================================================
==========================================================================
Layer (type:depth-idx)                   Input Shape
Output Shape              Param #                   Trainable
==========================================================================
==========================================================================
SimpleCNN                                [128, 1, 28, 28]
[128, 10]                 --                        True
├─Conv2d: 1-1                            [128, 1, 28, 28]
[128, 32, 28, 28]         320                       True
├─Conv2d: 1-2                            [128, 32, 28, 28]
[128, 64, 28, 28]         18,496                    True
├─MaxPool2d: 1-3                         [128, 64, 28, 28]
[128, 64, 14, 14]         --                        --
├─Linear: 1-4                            [128, 12544]
[128, 128]                1,605,760                 True
├─Linear: 1-5                            [128, 128]
[128, 10]                 1,290                     True
==========================================================================
==========================================================================
Total params: 1,625,866
Trainable params: 1,625,866
Non-trainable params: 0
Total mult-adds (G): 2.09
==========================================================================
==========================================================================
Input size (MB): 0.40
Forward/backward pass size (MB): 77.21
Params size (MB): 6.50
Estimated Total Size (MB): 84.12
==========================================================================
==========================================================================
```

□□ 3, □□ 4

4 1ㄱㅐㅇㅓㅌㅔㅅㅡㅌㅡ ㅇㅣㅁㅣㅈㅣ가ㄷㅗㅇㅇㅣㄹㅎㅏㅈㅣ ㅇㅏㅀㅇㅡㅁㅇㅣㅇㅏㄴㅣㄴㄹㅔㅇㅣㅂㅡㄹㄱㅘㅂㅜㄴㄹㅠㅇㅔㅊㅡㄱㅇㅣ ㄷㅗㅇㅇㅣㄹㅎㅏㄱㅔㅅㅓㄹ .1ㄱㅐ가ㅁㅜㅈㅗㄱㅓㄴ ㅇㅏㄴㅣㄱㅔ ㅎㅏㄴㅡㄴ ㄱㅓㅅㅇㅡㄴ ㅎㅏㄷㅏㄴㅇㅔ

```python
# Import libraries
import os
import random
import torch
from torch import nn
from torch.utils.data import DataLoader, random_split
from torchvision import datasets, transforms, models
from torch.optim import Adam
```

```
from torch.optim.lr_scheduler import StepLR
from torchinfo import summary
import matplotlib.pyplot as plt

# Check device
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(f"Using device: {device}")

# Define data loaders
def get_fashion_mnist_data(batch_size=64):
    transform = transforms.Compose([
        transforms.RandomHorizontalFlip(p=0.5),
        transforms.RandomRotation(15),
        transforms.ToTensor(),
        transforms.Normalize(mean=0.286, std=0.353)
    ])
    dataset = datasets.FashionMNIST(root="./data", train=True,
download=True, transform=transform)
    train_set, val_set = random_split(dataset, [55000, 5000])
    train_loader = DataLoader(train_set, batch_size=batch_size,
shuffle=True)
    val_loader = DataLoader(val_set, batch_size=batch_size)
    return train_loader, val_loader

def get_fashion_mnist_test_data(batch_size=64):
    transform = transforms.Compose([
        transforms.ToTensor(),
        transforms.Normalize(mean=0.286, std=0.353)
    ])
    test_set = datasets.FashionMNIST(root="./data", train=False,
download=True, transform=transform)
    test_loader = DataLoader(test_set, batch_size=batch_size)
    return test_loader

# Define CNN model
class CNNModel(nn.Module):
    def __init__(self):
        super(CNNModel, self).__init__()
        self.conv_layers = nn.Sequential(
            nn.Conv2d(1, 64, kernel_size=5, padding="same"),
            nn.BatchNorm2d(64),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2),

            nn.Conv2d(64, 128, kernel_size=5, padding="same"),
            nn.BatchNorm2d(128),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2),

            nn.Conv2d(128, 256, kernel_size=5, padding="same"),
```

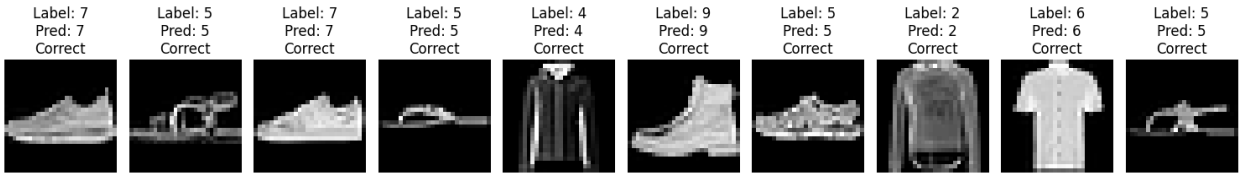
```
            nn.BatchNorm2d(256),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2)
        )
        self.fc_layers = nn.Sequential(
            nn.Flatten(),
            nn.Linear(256 * 3 * 3, 256),
            nn.ReLU(),
            nn.Dropout(0.2),
            nn.Linear(256, 10)
        )

    def forward(self, x):
        x = self.conv_layers(x)
        x = self.fc_layers(x)
        return x

# Initialize weights
def initialize_weights(m):
    if isinstance(m, nn.Conv2d) or isinstance(m, nn.Linear):
        nn.init.kaiming_normal_(m.weight)

# Training and evaluation
def train_and_evaluate(model, train_loader, val_loader, epochs=20,
learning_rate=0.001):
    model.to(device)
    model.apply(initialize_weights)
    optimizer = Adam(model.parameters(), lr=learning_rate)
    scheduler = StepLR(optimizer, step_size=10, gamma=0.1)
    criterion = nn.CrossEntropyLoss()

    for epoch in range(1, epochs + 1):
        # Training
        model.train()
        train_loss, train_correct, train_total = 0, 0, 0
        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()
            train_loss += loss.item()
            _, predicted = torch.max(outputs, 1)
            train_total += labels.size(0)
            train_correct += (predicted == labels).sum().item()

        scheduler.step()
        train_accuracy = train_correct / train_total
```

```
        # Validation
        model.eval()
        val_loss, val_correct, val_total = 0, 0, 0
        with torch.no_grad():
            for images, labels in val_loader:
                images, labels = images.to(device), labels.to(device)
                outputs = model(images)
                loss = criterion(outputs, labels)
                val_loss += loss.item()
                _, predicted = torch.max(outputs, 1)
                val_total += labels.size(0)
                val_correct += (predicted == labels).sum().item()

        val_accuracy = val_correct / val_total
        print(f"Epoch {epoch}/{epochs} - Train Loss: {train_loss:.4f},
Train Acc: {train_accuracy:.4f}, "
              f"Val Loss: {val_loss:.4f}, Val Acc:
{val_accuracy:.4f}")

    return model

def evaluate_test_data(model, test_loader):
    model.eval()
    test_correct, test_total = 0, 0
    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)
            test_total += labels.size(0)
            test_correct += (predicted == labels).sum().item()
    test_accuracy = test_correct / test_total
    print(f"Test Accuracy: {test_accuracy:.4f}")
    return test_accuracy


# Visualize predictions
def visualize_predictions(model, test_loader, num_samples=10):
    model.eval()
    images, labels = next(iter(test_loader))
    images, labels = images.to(device), labels.to(device)

    indices = random.sample(range(len(images)), num_samples)
    selected_images = images[indices]
    selected_labels = labels[indices]

    with torch.no_grad():
        outputs = model(selected_images)
        _, predicted = torch.max(outputs, 1)
```

```
    plt.figure(figsize=(15, 5))
    for i in range(num_samples):
        img = selected_images[i].cpu().squeeze()
        label = selected_labels[i].item()
        pred = predicted[i].item()
        plt.subplot(1, num_samples, i + 1)
        plt.imshow(img, cmap="gray")
        plt.title(f"Label: {label}\nPred: {pred}\n{'Correct' if label
== pred else 'Wrong'}")
        plt.axis("off")
    plt.tight_layout()
    plt.show()

# Main execution
if __name__ == "__main__":
    batch_size = 64
    epochs = 20
    learning_rate = 0.001

    # Data loaders
    train_loader, val_loader = get_fashion_mnist_data(batch_size)
    test_loader = get_fashion_mnist_test_data(batch_size)

    # Model
    model = CNNModel()
    print("\nModel Summary:")
    summary(model, input_size=(batch_size, 1, 28, 28),
col_names=["input_size", "output_size", "num_params", "trainable"])

    # Training and evaluation
    model = train_and_evaluate(model, train_loader, val_loader,
epochs, learning_rate)

    # Test evaluation
    evaluate_test_data(model, test_loader)

    # Visualize predictions
    visualize_predictions(model, test_loader, num_samples=10)
```

```
Using device: cuda

Model Summary:
Epoch 1/20 - Train Loss: 492.7378, Train Acc: 0.7947, Val Loss:
33.0454, Val Acc: 0.8374
Epoch 2/20 - Train Loss: 328.1750, Train Acc: 0.8606, Val Loss:
26.9135, Val Acc: 0.8724
Epoch 3/20 - Train Loss: 284.3532, Train Acc: 0.8778, Val Loss:
21.7702, Val Acc: 0.8976
Epoch 4/20 - Train Loss: 260.7601, Train Acc: 0.8886, Val Loss:
21.8276, Val Acc: 0.8966
```

```
Epoch 5/20 - Train Loss: 241.8697, Train Acc: 0.8970, Val Loss:
21.8927, Val Acc: 0.8950
Epoch 6/20 - Train Loss: 228.5601, Train Acc: 0.9021, Val Loss:
20.7777, Val Acc: 0.8996
Epoch 7/20 - Train Loss: 213.6508, Train Acc: 0.9087, Val Loss:
18.5639, Val Acc: 0.9114
Epoch 8/20 - Train Loss: 202.8541, Train Acc: 0.9116, Val Loss:
19.6473, Val Acc: 0.9086
Epoch 9/20 - Train Loss: 196.6834, Train Acc: 0.9153, Val Loss:
19.4991, Val Acc: 0.9086
Epoch 10/20 - Train Loss: 188.0848, Train Acc: 0.9181, Val Loss:
17.9062, Val Acc: 0.9188
Epoch 11/20 - Train Loss: 152.7850, Train Acc: 0.9340, Val Loss:
15.6826, Val Acc: 0.9266
Epoch 12/20 - Train Loss: 142.1526, Train Acc: 0.9381, Val Loss:
15.4411, Val Acc: 0.9284
Epoch 13/20 - Train Loss: 136.4327, Train Acc: 0.9410, Val Loss:
14.8171, Val Acc: 0.9354
Epoch 14/20 - Train Loss: 133.3231, Train Acc: 0.9423, Val Loss:
15.1476, Val Acc: 0.9316
Epoch 15/20 - Train Loss: 130.7323, Train Acc: 0.9433, Val Loss:
14.8775, Val Acc: 0.9356
Epoch 16/20 - Train Loss: 125.6844, Train Acc: 0.9459, Val Loss:
15.4331, Val Acc: 0.9306
Epoch 17/20 - Train Loss: 124.6524, Train Acc: 0.9457, Val Loss:
15.2534, Val Acc: 0.9322
Epoch 18/20 - Train Loss: 121.1426, Train Acc: 0.9476, Val Loss:
15.1485, Val Acc: 0.9348
Epoch 19/20 - Train Loss: 118.9053, Train Acc: 0.9478, Val Loss:
14.9872, Val Acc: 0.9344
Epoch 20/20 - Train Loss: 115.5316, Train Acc: 0.9495, Val Loss:
14.9175, Val Acc: 0.9336
Test Accuracy: 0.9281
```




```python
# Import libraries
import os
import random
import torch
from torch import nn
from torch.utils.data import DataLoader, random_split
from torchvision import datasets, transforms, models
from torch.optim import Adam
from torch.optim.lr_scheduler import StepLR
```

```
from torchinfo import summary
import matplotlib.pyplot as plt

# Check device
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(f"Using device: {device}")

# Define data loaders
def get_fashion_mnist_data(batch_size=64):
    transform = transforms.Compose([
        transforms.RandomHorizontalFlip(p=0.5),
        transforms.RandomRotation(15),
        transforms.ToTensor(),
        transforms.Normalize(mean=0.286, std=0.353)
    ])
    dataset = datasets.FashionMNIST(root="./data", train=True,
download=True, transform=transform)
    train_set, val_set = random_split(dataset, [55000, 5000])
    train_loader = DataLoader(train_set, batch_size=batch_size,
shuffle=True)
    val_loader = DataLoader(val_set, batch_size=batch_size)
    return train_loader, val_loader

def get_fashion_mnist_test_data(batch_size=64):
    transform = transforms.Compose([
        transforms.ToTensor(),
        transforms.Normalize(mean=0.286, std=0.353)
    ])
    test_set = datasets.FashionMNIST(root="./data", train=False,
download=True, transform=transform)
    test_loader = DataLoader(test_set, batch_size=batch_size)
    return test_loader

# Define CNN model
class CNNModel(nn.Module):
    def __init__(self):
        super(CNNModel, self).__init__()
        self.conv_layers = nn.Sequential(
            nn.Conv2d(1, 64, kernel_size=5, padding="same"),
            nn.BatchNorm2d(64),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2),

            nn.Conv2d(64, 128, kernel_size=5, padding="same"),
            nn.BatchNorm2d(128),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2),

            nn.Conv2d(128, 256, kernel_size=5, padding="same"),
            nn.BatchNorm2d(256),
```

```
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2)
        )
        self.fc_layers = nn.Sequential(
            nn.Flatten(),
            nn.Linear(256 * 3 * 3, 256),
            nn.ReLU(),
            nn.Dropout(0.2),
            nn.Linear(256, 10)
        )

    def forward(self, x):
        x = self.conv_layers(x)
        x = self.fc_layers(x)
        return x

# Initialize weights
def initialize_weights(m):
    if isinstance(m, nn.Conv2d) or isinstance(m, nn.Linear):
        nn.init.kaiming_normal_(m.weight)

# Training and evaluation
def train_and_evaluate(model, train_loader, val_loader, epochs=20,
learning_rate=0.001):
    model.to(device)
    model.apply(initialize_weights)
    optimizer = Adam(model.parameters(), lr=learning_rate)
    scheduler = StepLR(optimizer, step_size=10, gamma=0.1)
    criterion = nn.CrossEntropyLoss()

    for epoch in range(1, epochs + 1):
        # Training
        model.train()
        train_loss, train_correct, train_total = 0, 0, 0
        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()
            train_loss += loss.item()
            _, predicted = torch.max(outputs, 1)
            train_total += labels.size(0)
            train_correct += (predicted == labels).sum().item()

        scheduler.step()
        train_accuracy = train_correct / train_total

        # Validation
```

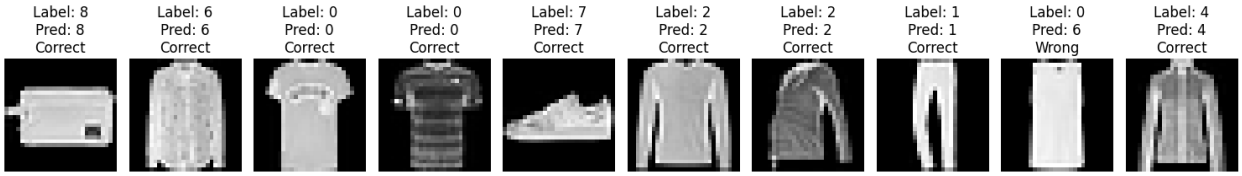
```
        model.eval()
        val_loss, val_correct, val_total = 0, 0, 0
        with torch.no_grad():
            for images, labels in val_loader:
                images, labels = images.to(device), labels.to(device)
                outputs = model(images)
                loss = criterion(outputs, labels)
                val_loss += loss.item()
                _, predicted = torch.max(outputs, 1)
                val_total += labels.size(0)
                val_correct += (predicted == labels).sum().item()

        val_accuracy = val_correct / val_total
        print(f"Epoch {epoch}/{epochs} - Train Loss: {train_loss:.4f},
Train Acc: {train_accuracy:.4f}, "
              f"Val Loss: {val_loss:.4f}, Val Acc:
{val_accuracy:.4f}")

    return model

def evaluate_test_data(model, test_loader):
    model.eval()
    test_correct, test_total = 0, 0
    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)
            test_total += labels.size(0)
            test_correct += (predicted == labels).sum().item()
    test_accuracy = test_correct / test_total
    print(f"Test Accuracy: {test_accuracy:.4f}")
    return test_accuracy


def visualize_predictions_with_misclassification(model, test_loader,
num_samples=10):
    """
    Visualize random predictions including at least one misclassified
sample.
    """
    model.eval()
    images, labels = next(iter(test_loader))
    images, labels = images.to(device), labels.to(device)

    # Make predictions
    with torch.no_grad():
        outputs = model(images)
        _, predicted = torch.max(outputs, 1)
```

```
    # Find misclassified samples
    correct = (predicted == labels)
    incorrect = ~correct
    incorrect_indices = torch.nonzero(incorrect).squeeze().tolist()
    if isinstance(incorrect_indices, int):  # Handle single incorrect
case
        incorrect_indices = [incorrect_indices]

    # Ensure at least one misclassified sample
    selected_indices = random.sample(range(len(images)), num_samples -
1)  # Select random samples
    if incorrect_indices:  # If there are misclassified samples,
include one
        selected_indices.append(incorrect_indices[0])
    else:
        print("Warning: No misclassified samples found in this
batch.")
        selected_indices = random.sample(range(len(images)),
num_samples)  # Fallback to random samples

    # Visualize selected samples
    selected_images = images[selected_indices]
    selected_labels = labels[selected_indices]
    selected_predictions = predicted[selected_indices]

    plt.figure(figsize=(15, 5))
    for i in range(num_samples):
        img = selected_images[i].cpu().squeeze()
        label = selected_labels[i].item()
        pred = selected_predictions[i].item()
        plt.subplot(1, num_samples, i + 1)
        plt.imshow(img, cmap="gray")
        plt.title(f"Label: {label}\nPred: {pred}\n{'Correct' if label
== pred else 'Wrong'}")
        plt.axis("off")
    plt.tight_layout()
    plt.show()

    # Analyze the misclassified sample
    if incorrect_indices:
        print("\nMisclassified Sample Analysis:")
        misclassified_image =
images[incorrect_indices[0]].cpu().squeeze()
        misclassified_label = labels[incorrect_indices[0]].item()
        misclassified_pred = predicted[incorrect_indices[0]].item()
        plt.figure(figsize=(3, 3))
        plt.imshow(misclassified_image, cmap="gray")
        plt.title(f"Label: {misclassified_label}, Pred:
{misclassified_pred} (Wrong)")
        plt.axis("off")
```

```
        plt.show()


# Main execution
if __name__ == "__main__":
    batch_size = 64
    epochs = 20
    learning_rate = 0.001

    # Data loaders
    train_loader, val_loader = get_fashion_mnist_data(batch_size)
    test_loader = get_fashion_mnist_test_data(batch_size)

    # Model
    model = CNNModel()
    print("\nModel Summary:")
    summary(model, input_size=(batch_size, 1, 28, 28),
col_names=["input_size", "output_size", "num_params", "trainable"])

    # Training and evaluation
    model = train_and_evaluate(model, train_loader, val_loader,
epochs, learning_rate)

    # Test evaluation
    evaluate_test_data(model, test_loader)

    # Visualize predictions
    visualize_predictions(model, test_loader, num_samples=10)
```

```
Using device: cuda

Model Summary:
Epoch 1/20 - Train Loss: 508.3955, Train Acc: 0.7921, Val Loss:
39.9118, Val Acc: 0.8132
Epoch 2/20 - Train Loss: 329.3808, Train Acc: 0.8603, Val Loss:
28.8398, Val Acc: 0.8672
Epoch 3/20 - Train Loss: 286.7039, Train Acc: 0.8795, Val Loss:
25.1831, Val Acc: 0.8818
Epoch 4/20 - Train Loss: 261.1377, Train Acc: 0.8885, Val Loss:
27.4458, Val Acc: 0.8776
Epoch 5/20 - Train Loss: 246.8587, Train Acc: 0.8938, Val Loss:
22.5386, Val Acc: 0.8944
Epoch 6/20 - Train Loss: 227.8867, Train Acc: 0.9026, Val Loss:
26.2912, Val Acc: 0.8790
Epoch 7/20 - Train Loss: 217.8819, Train Acc: 0.9062, Val Loss:
20.2346, Val Acc: 0.9066
Epoch 8/20 - Train Loss: 208.7526, Train Acc: 0.9097, Val Loss:
20.4965, Val Acc: 0.9120
Epoch 9/20 - Train Loss: 199.5646, Train Acc: 0.9148, Val Loss:
20.5986, Val Acc: 0.9066
```

```
Epoch 10/20 - Train Loss: 188.1443, Train Acc: 0.9192, Val Loss:
19.1594, Val Acc: 0.9110
Epoch 11/20 - Train Loss: 155.0726, Train Acc: 0.9343, Val Loss:
17.7000, Val Acc: 0.9214
Epoch 12/20 - Train Loss: 146.3248, Train Acc: 0.9380, Val Loss:
17.5810, Val Acc: 0.9234
Epoch 13/20 - Train Loss: 142.7166, Train Acc: 0.9384, Val Loss:
17.4210, Val Acc: 0.9258
Epoch 14/20 - Train Loss: 137.0445, Train Acc: 0.9416, Val Loss:
16.8176, Val Acc: 0.9230
Epoch 15/20 - Train Loss: 133.8361, Train Acc: 0.9417, Val Loss:
16.6635, Val Acc: 0.9258
Epoch 16/20 - Train Loss: 131.0920, Train Acc: 0.9434, Val Loss:
17.4009, Val Acc: 0.9230
Epoch 17/20 - Train Loss: 129.7344, Train Acc: 0.9439, Val Loss:
17.1526, Val Acc: 0.9270
Epoch 18/20 - Train Loss: 124.4361, Train Acc: 0.9472, Val Loss:
16.6782, Val Acc: 0.9226
Epoch 19/20 - Train Loss: 122.1095, Train Acc: 0.9477, Val Loss:
16.7930, Val Acc: 0.9246
Epoch 20/20 - Train Loss: 121.0121, Train Acc: 0.9483, Val Loss:
16.6938, Val Acc: 0.9276
Test Accuracy: 0.9323
```



ㅇ ㅔ ㅊ ㅡ ㄱ ㄱ ㅕ ㄹ ㄱ ㅏ 가 ㅌ ㅡ ㄹ ㄹ ㅣ ㄴ ㅇ ㅣ ㅇ ㅠ...
, flatten relu, adam

ㄹ ㅠ ㅇ ㅔ ㅊ ㅡ ㄱ ㄱ ㅏ ㅆ ㅇ ㅣ ㄷ ㅗ ㅇ ㅇ ㅣ ㄹ ㅎ ㅏ ㄱ ㅣ 가 ㅇ ㅓ ㄹ ㅕ ㅂ ㄱ ㅣ ㄸ ㅐ ㅁ ㅜ ㄴ ㅇ ㅡ ㄹ ㅗ ㅂ ㅗ
....

□□ 4

□□ □□

. 94 ... ..
ㅁ ㅏ ㄴ 가 ㅈ ㅕ ㅇ ㅏ ㅆ ㅅ ㅡ ㅂ ㄴ ㅣ ㄷ ㅏ.

github : https://github.com/zalandoresearch/fashion-mnist classifier , ㅁ ㅔ ㄴ
ㅁ ㅗ ㄷ ㅔ ㄹ ㅇ ㅡ ㄹ ㅅ ㅏ ㅇ ㅛ ㅇ ㅎ ㅐ ㅆ ㅇ ㅡ ㄹ ㄸ ㅐ ㅇ ㅓ ㄸ ㅓ ㄴ acc 가 ㄴ ㅏ ㅇ ㅏ ㅆ ㄴ ㅡ ㄴ ㅈ ㅣ ㅇ ㅔ

.ㅊㅏㅁㄱㅗㅎㅏㅁㅕㄴ ㅈㅗㅎㄱㅔㅆㄷㅏㄴㅡㄴ ㅅㅐㅇㄱㅏㄱㅇㅣ ㄷㅡㄹㅇㅓㅅㅓ가
...

Classifier	Preprocessing	Fashion test accuracy	MNIST test accuracy	Submitter
2 Conv+pooling	None	0.876	-	Kashif Rasul
2 Conv+pooling	None	0.916	-	Tensorflow's doc
2 Conv+pooling+ELU activation (PyTorch)	None	0.903	-	@AbhirajHinge
2 Conv	Normalization, random horizontal flip, random vertical flip, random translation, random rotation.	0.919	0.971	Kyriakos Efthymiadis
2 Conv <100K parameters	None	0.925	0.992	@hardmaru
2 Conv ~113K parameters	Normalization	0.922	0.993	Abel G.
2 Conv+3 FC	Normalization	0.932	0.994	@Xfan1025

```python
from pathlib import Path
import os
import torch
import wandb
from torch import nn
from torch.utils.data import DataLoader, random_split
from torchvision import datasets, transforms, models
from torch.optim import Adam
from torch.optim.lr_scheduler import StepLR
from torchinfo import summary  # torchinfo

# Use the current working directory
BASE_PATH = Path().resolve()

def get_fashion_mnist_data():
    data_path = os.path.join(BASE_PATH, "_00_data", "j_fashion_mnist")

    # ㄷㅐㅇㅣㅌㅓ ㅈㅡㅇㄱㅏㅇ ㅊㅜㄱ가
    transform = transforms.Compose([
        transforms.RandomHorizontalFlip(p=0.5),  # 50%
        transforms.RandomRotation(15),           # -15  15
        transforms.ToTensor(),
        transforms.Normalize(mean=0.286, std=0.353)  #
    ])
```

```
    f_mnist_train = datasets.FashionMNIST(data_path, train=True,
download=True, transform=transform)
    f_mnist_train, f_mnist_validation = random_split(f_mnist_train,
[55_000, 5_000])

    train_data_loader = DataLoader(
        dataset=f_mnist_train, batch_size=wandb.config.batch_size,
shuffle=True, pin_memory=True, num_workers=0
    )

    validation_data_loader = DataLoader(
        dataset=f_mnist_validation,
batch_size=wandb.config.batch_size, pin_memory=True, num_workers=0
    )

    return train_data_loader, validation_data_loader


def get_fashion_mnist_test_data():
    data_path = os.path.join(BASE_PATH, "_00_data", "j_fashion_mnist")
    transform = transforms.Compose([
        transforms.ToTensor(),
        transforms.Normalize(mean=0.286, std=0.353)  #
    ])
    f_mnist_test = datasets.FashionMNIST(data_path, train=False,
download=True, transform=transform)

    test_data_loader = DataLoader(
        dataset=f_mnist_test, batch_size=wandb.config.batch_size,
pin_memory=True, num_workers=0
    )
    return test_data_loader


class CNNModel(nn.Module):
    def __init__(self):
        super(CNNModel, self).__init__()
        self.conv_layers = nn.Sequential(
            nn.Conv2d(1, 64, kernel_size=5, padding="same"),
            nn.BatchNorm2d(64),  # Batch Normalization 추가
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2),  # Output: (64, 14, 14)

            nn.Conv2d(64, 128, kernel_size=5, padding="same"),
            nn.BatchNorm2d(128),  # Batch Normalization 추가
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2),  # Output: (128, 7, 7)

            nn.Conv2d(128, 256, kernel_size=5, padding="same"),
```

```
            nn.BatchNorm2d(256),  # Batch Normalization 추가
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2)   # Output: (256, 3, 3)
        )

        self.fc_layers = nn.Sequential(
            nn.Flatten(),                           # Flatten for
Fully Connected Layer
            nn.Linear(256 * 3 * 3, 256),            # Fully Connected
Layer
            nn.ReLU(),
            nn.Dropout(0.2),                        # Dropout 추가
            nn.Linear(256, 10)                      # Output Layer for
10 Classes
        )

    def forward(self, x):
        x = self.conv_layers(x)
        x = self.fc_layers(x)
        return x


def train_and_evaluate(epochs, train_data_loader,
validation_data_loader):
    device = torch.device("cuda" if torch.cuda.is_available() else
"cpu")

    # Model 초기화 및 가중치 설정
    model = CNNModel().to(device)
    model.apply(initialize_weights)  # 가중치 설정

    optimizer = Adam(model.parameters(),
lr=wandb.config.learning_rate)
    scheduler = StepLR(optimizer, step_size=10, gamma=0.1)  # 10 epoch
  90%
    criterion = nn.CrossEntropyLoss()

    for epoch in range(1, epochs + 1):
        # Training
        model.train()
        train_loss, train_correct, train_total = 0, 0, 0
        for images, labels in train_data_loader:
            images, labels = images.to(device), labels.to(device)
            optimizer.zero_grad()

            outputs = model(images)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()
```

```
            train_loss += loss.item()
            _, predicted = torch.max(outputs, 1)
            train_total += labels.size(0)
            train_correct += (predicted == labels).sum().item()

        scheduler.step()  #

        train_accuracy = train_correct / train_total
        train_loss /= len(train_data_loader)

        # Validation
        model.eval()
        val_loss, val_correct, val_total = 0, 0, 0
        with torch.no_grad():
            for images, labels in validation_data_loader:
                images, labels = images.to(device), labels.to(device)

                outputs = model(images)
                loss = criterion(outputs, labels)

                val_loss += loss.item()
                _, predicted = torch.max(outputs, 1)
                val_total += labels.size(0)
                val_correct += (predicted == labels).sum().item()

        val_accuracy = val_correct / val_total
        val_loss /= len(validation_data_loader)

        # Log metrics to wandb
        wandb.log({
            "epoch": epoch,
            "train_loss": train_loss,
            "train_accuracy": train_accuracy,
            "val_loss": val_loss,
            "val_accuracy": val_accuracy
        })

        print(f"Epoch [{epoch}/{epochs}] - Train Loss:
{train_loss:.4f}, Train Acc: {train_accuracy:.4f}, "
              f"Val Loss: {val_loss:.4f}, Val Acc:
{val_accuracy:.4f}")

    return model  #


def evaluate_test_data(model, test_data_loader, device):
    model.eval()
    test_correct, test_total = 0, 0
    with torch.no_grad():
        for images, labels in test_data_loader:
```

```
            images, labels = images.to(device), labels.to(device)

            outputs = model(images)
            _, predicted = torch.max(outputs, 1)
            test_total += labels.size(0)
            test_correct += (predicted == labels).sum().item()

    test_accuracy = test_correct / test_total
    return test_accuracy


# 가중치 초기화 함수
def initialize_weights(m):
    if isinstance(m, nn.Conv2d) or isinstance(m, nn.Linear):
        nn.init.kaiming_normal_(m.weight)  # He Initialization


if __name__ == "__main__":
    config = {"batch_size": 64, "learning_rate": 0.0005, "epochs": 20}
    wandb.init(project="fashion-mnist-project", config=config,
name="64, 0.0005,100")

    train_data_loader, validation_data_loader =
get_fashion_mnist_data()
    test_data_loader = get_fashion_mnist_test_data()

    # GoogLeNet
    googlenet = models.googlenet(pretrained=False, aux_logits=False)
    print("\nGoogLeNet Model Structure Summary:\n")
    summary(googlenet, input_size=(64, 1, 28, 28),
col_names=["input_size", "output_size", "num_params", "trainable"])

    # Start training and evaluating
    trained_model = train_and_evaluate(wandb.config.epochs,
train_data_loader, validation_data_loader)

    # Evaluate on test data
    device = torch.device("cuda" if torch.cuda.is_available() else
"cpu")
    test_accuracy = evaluate_test_data(trained_model,
test_data_loader, device)

    # Log test accuracy to wandb
    wandb.log({"test_accuracy": test_accuracy})
    print(f"Test Accuracy: {test_accuracy:.4f}")

    wandb.finish()
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>


GoogLeNet Model Structure Summary:


---------------------------------------------------------------------------
-----
RuntimeError                              Traceback (most recent call
last)
File ~/.local/lib/python3.10/site-packages/torchinfo/torchinfo.py:295,
in forward_pass(model, x, batch_dim, cache_forward_pass, device, mode,
**kwargs)
    294 if isinstance(x, (list, tuple)):
--> 295     _ = model(*x, **kwargs)
    296 elif isinstance(x, dict):

File
/usr/local/lib/python3.10/dist-packages/torch/nn/modules/module.py:151
1, in Module._wrapped_call_impl(self, *args, **kwargs)
   1510 else:
-> 1511     return self._call_impl(*args, **kwargs)

File
/usr/local/lib/python3.10/dist-packages/torch/nn/modules/module.py:156
1, in Module._call_impl(self, *args, **kwargs)
   1559     args = bw_hook.setup_input_hook(args)
-> 1561 result = forward_call(*args, **kwargs)
   1562 if _global_forward_hooks or self._forward_hooks:

File
/usr/local/lib/python3.10/dist-packages/torchvision/models/googlenet.p
y:174, in GoogLeNet.forward(self, x)
    173 x = self._transform_input(x)
--> 174 x, aux1, aux2 = self._forward(x)
    175 aux_defined = self.training and self.aux_logits

File
/usr/local/lib/python3.10/dist-packages/torchvision/models/googlenet.p
y:112, in GoogLeNet._forward(self, x)
```

```
    110 def _forward(self, x: Tensor) -> Tuple[Tensor,
Optional[Tensor], Optional[Tensor]]:
    111     # N x 3 x 224 x 224
--> 112     x = self.conv1(x)
    113     # N x 64 x 112 x 112

File
/usr/local/lib/python3.10/dist-packages/torch/nn/modules/module.py:151
1, in Module._wrapped_call_impl(self, *args, **kwargs)
   1510 else:
-> 1511     return self._call_impl(*args, **kwargs)

File
/usr/local/lib/python3.10/dist-packages/torch/nn/modules/module.py:156
1, in Module._call_impl(self, *args, **kwargs)
   1559     args = bw_hook.setup_input_hook(args)
-> 1561 result = forward_call(*args, **kwargs)
   1562 if _global_forward_hooks or self._forward_hooks:

File
/usr/local/lib/python3.10/dist-packages/torchvision/models/googlenet.p
y:273, in BasicConv2d.forward(self, x)
    272 def forward(self, x: Tensor) -> Tensor:
--> 273     x = self.conv(x)
    274     x = self.bn(x)

File
/usr/local/lib/python3.10/dist-packages/torch/nn/modules/module.py:151
1, in Module._wrapped_call_impl(self, *args, **kwargs)
   1510 else:
-> 1511     return self._call_impl(*args, **kwargs)

File
/usr/local/lib/python3.10/dist-packages/torch/nn/modules/module.py:156
1, in Module._call_impl(self, *args, **kwargs)
   1559     args = bw_hook.setup_input_hook(args)
-> 1561 result = forward_call(*args, **kwargs)
   1562 if _global_forward_hooks or self._forward_hooks:

File
/usr/local/lib/python3.10/dist-packages/torch/nn/modules/conv.py:460,
in Conv2d.forward(self, input)
    459 def forward(self, input: Tensor) -> Tensor:
--> 460     return self._conv_forward(input, self.weight, self.bias)

File
/usr/local/lib/python3.10/dist-packages/torch/nn/modules/conv.py:456,
in Conv2d._conv_forward(self, input, weight, bias)
    453     return F.conv2d(F.pad(input,
self._reversed_padding_repeated_twice, mode=self.padding_mode),
```

```
    454                     weight, bias, self.stride,
    455                     _pair(0), self.dilation, self.groups)
--> 456 return F.conv2d(input, weight, bias, self.stride,
    457                 self.padding, self.dilation, self.groups)

RuntimeError: Given groups=1, weight of size [64, 3, 7, 7], expected
input[64, 1, 28, 28] to have 3 channels, but got 1 channels instead

The above exception was the direct cause of the following exception:

RuntimeError                              Traceback (most recent call
last)
Cell In[3], line 187
    185 googlenet = models.googlenet(pretrained=False,
aux_logits=False)
    186 print("\nGoogLeNet Model Structure Summary:\n")
--> 187 summary(googlenet, input_size=(64, 1, 28, 28),
col_names=["input_size", "output_size", "num_params", "trainable"])
    189 # Start training and evaluating
    190 trained_model = train_and_evaluate(wandb.config.epochs,
train_data_loader, validation_data_loader)

File ~/.local/lib/python3.10/site-packages/torchinfo/torchinfo.py:223,
in summary(model, input_size, input_data, batch_dim,
cache_forward_pass, col_names, col_width, depth, device, dtypes, mode,
row_settings, verbose, **kwargs)
    216 validate_user_params(
    217     input_data, input_size, columns, col_width, device,
dtypes, verbose
    218 )
    220 x, correct_input_size = process_input(
    221     input_data, input_size, batch_dim, device, dtypes
    222 )
--> 223 summary_list = forward_pass(
    224     model, x, batch_dim, cache_forward_pass, device,
model_mode, **kwargs
    225 )
    226 formatting = FormattingOptions(depth, verbose, columns,
col_width, rows)
    227 results = ModelStatistics(
    228     summary_list, correct_input_size,
get_total_memory_used(x), formatting
    229 )

File ~/.local/lib/python3.10/site-packages/torchinfo/torchinfo.py:304,
in forward_pass(model, x, batch_dim, cache_forward_pass, device, mode,
**kwargs)
    302 except Exception as e:
    303     executed_layers = [layer for layer in summary_list if
layer.executed]
```

```
--> 304     raise RuntimeError(
    305         "Failed to run torchinfo. See above stack traces for
more details. "
    306         f"Executed layers up to: {executed_layers}"
    307     ) from e
    308 finally:
    309     if hooks:

RuntimeError: Failed to run torchinfo. See above stack traces for more
details. Executed layers up to: []
```

```
def get_fashion_mnist_data():
    data_path = os.path.join(BASE_PATH, "_00_data", "j_fashion_mnist")

    # ㄷㅐㅇㅣㅌㅣㅈㅡㅇㄱㅏㅌㅇㅊㅜㄱㅏ
    transform = transforms.Compose([
        transforms.RandomHorizontalFlip(p=0.5),  # 50%
        transforms.RandomRotation(15),          # -15  15
        transforms.ToTensor(),
        transforms.Normalize(mean=0.286, std=0.353)  #
    ])

    f_mnist_train = datasets.FashionMNIST(data_path, train=True,
download=True, transform=transform)
    f_mnist_train, f_mnist_validation = random_split(f_mnist_train,
[55_000, 5_000])


class CNNModel(nn.Module):
    def __init__(self):
        super(CNNModel, self).__init__()
        self.conv_layers = nn.Sequential(
            nn.Conv2d(1, 64, kernel_size=5, padding="same"),
            nn.BatchNorm2d(64),  # Batch Normalization ㅊㅜㄱㅏ
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2),  # Output: (64, 14, 14)

            nn.Conv2d(64, 128, kernel_size=5, padding="same"),
            nn.BatchNorm2d(128),  # Batch Normalization ㅊㅜㄱㅏ
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2),  # Output: (128, 7, 7)

            nn.Conv2d(128, 256, kernel_size=5, padding="same"),
            nn.BatchNorm2d(256),  # Batch Normalization ㅊㅜㄱㅏ
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2)   # Output: (256, 3, 3)
        )

        self.fc_layers = nn.Sequential(
            nn.Flatten(),                           # Flatten for
```

```
Fully Connected Layer
            nn.Linear(256 * 3 * 3, 256),            # Fully Connected
Layer
            nn.ReLU(),
            nn.Dropout(0.2),                        # Dropout ㅊㅜ가
            nn.Linear(256, 10)                      # Output Layer for
10 Classes
        )

    def forward(self, x):
        x = self.conv_layers(x)
        x = self.fc_layers(x)
        return x


def train_and_evaluate(epochs, train_data_loader,
validation_data_loader):
    device = torch.device("cuda" if torch.cuda.is_available() else
"cpu")

    # Model ㅈㅣㅇㅇㅣㅁㅣㅊ가ㅈㅜㅇㅊㅣ ㅊㅗㄱㅣㅎㅏ

    model = CNNModel().to(device)
    model.apply(initialize_weights)  # 가ㅈㅜㅇㅊㅣ ㅊㅗㄱㅣㅎㅏ

    optimizer = Adam(model.parameters(),
lr=wandb.config.learning_rate)
    scheduler = StepLR(optimizer, step_size=10, gamma=0.1)  # 10 epoch
  90%
    criterion = nn.CrossEntropyLoss()

    for epoch in range(1, epochs + 1):
        # Training
        model.train()
        train_loss, train_correct, train_total = 0, 0, 0
        for images, labels in train_data_loader:
            images, labels = images.to(device), labels.to(device)
            optimizer.zero_grad()

            outputs = model(images)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()

            train_loss += loss.item()
            _, predicted = torch.max(outputs, 1)
            train_total += labels.size(0)
            train_correct += (predicted == labels).sum().item()

        scheduler.step()  #
```

```
# 가중치 초기화 함수
def initialize_weights(m):
    if isinstance(m, nn.Conv2d) or isinstance(m, nn.Linear):
        nn.init.kaiming_normal_(m.weight)  # He Initialization


if __name__ == "__main__":
    config = {"batch_size": 64, "learning_rate": 0.001, "epochs": 100}
```

```
wandb: Currently logged in as: -ddj127 (-ddj127-korea-university-of-technology-and-education). Use `wandb login --relogin` to force relogin
```


```
{"model_id":"b0e5eea466e0493283e27011f0de2323","version_major":2,"version_minor":0}
```


```
<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

Epoch [1/100] - Train Loss: 0.5602, Train Acc: 0.8006, Val Loss: 0.3983, Val Acc: 0.8528
Epoch [2/100] - Train Loss: 0.3753, Train Acc: 0.8632, Val Loss: 0.3567, Val Acc: 0.8732
Epoch [3/100] - Train Loss: 0.3267, Train Acc: 0.8803, Val Loss: 0.2959, Val Acc: 0.8910
Epoch [4/100] - Train Loss: 0.2978, Train Acc: 0.8908, Val Loss: 0.2906, Val Acc: 0.8936
Epoch [5/100] - Train Loss: 0.2812, Train Acc: 0.8971, Val Loss: 0.3019, Val Acc: 0.8912
Epoch [6/100] - Train Loss: 0.2646, Train Acc: 0.9031, Val Loss: 0.2712, Val Acc: 0.9008
Epoch [7/100] - Train Loss: 0.2483, Train Acc: 0.9060, Val Loss: 0.2619, Val Acc: 0.9066
Epoch [8/100] - Train Loss: 0.2368, Train Acc: 0.9126, Val Loss: 0.2543, Val Acc: 0.9086
Epoch [9/100] - Train Loss: 0.2265, Train Acc: 0.9176, Val Loss: 0.2683, Val Acc: 0.9052
Epoch [10/100] - Train Loss: 0.2166, Train Acc: 0.9202, Val Loss: 0.2570, Val Acc: 0.9110
Epoch [11/100] - Train Loss: 0.1765, Train Acc: 0.9357, Val Loss: 0.2148, Val Acc: 0.9244
```

```
Epoch [12/100] - Train Loss: 0.1657, Train Acc: 0.9381, Val Loss:
0.2196, Val Acc: 0.9218
Epoch [13/100] - Train Loss: 0.1600, Train Acc: 0.9414, Val Loss:
0.2193, Val Acc: 0.9212
Epoch [14/100] - Train Loss: 0.1545, Train Acc: 0.9423, Val Loss:
0.2214, Val Acc: 0.9186
Epoch [15/100] - Train Loss: 0.1511, Train Acc: 0.9444, Val Loss:
0.2134, Val Acc: 0.9252
Epoch [16/100] - Train Loss: 0.1481, Train Acc: 0.9445, Val Loss:
0.2118, Val Acc: 0.9218
Epoch [17/100] - Train Loss: 0.1461, Train Acc: 0.9456, Val Loss:
0.2146, Val Acc: 0.9224
Epoch [18/100] - Train Loss: 0.1420, Train Acc: 0.9469, Val Loss:
0.2142, Val Acc: 0.9286
Epoch [19/100] - Train Loss: 0.1372, Train Acc: 0.9501, Val Loss:
0.2076, Val Acc: 0.9262
Epoch [20/100] - Train Loss: 0.1357, Train Acc: 0.9499, Val Loss:
0.2115, Val Acc: 0.9280
Epoch [21/100] - Train Loss: 0.1308, Train Acc: 0.9519, Val Loss:
0.2178, Val Acc: 0.9268
Epoch [22/100] - Train Loss: 0.1283, Train Acc: 0.9532, Val Loss:
0.2093, Val Acc: 0.9324
Epoch [23/100] - Train Loss: 0.1292, Train Acc: 0.9520, Val Loss:
0.2087, Val Acc: 0.9264
Epoch [24/100] - Train Loss: 0.1272, Train Acc: 0.9521, Val Loss:
0.2088, Val Acc: 0.9264
Epoch [25/100] - Train Loss: 0.1267, Train Acc: 0.9529, Val Loss:
0.2093, Val Acc: 0.9286
Epoch [26/100] - Train Loss: 0.1255, Train Acc: 0.9537, Val Loss:
0.2106, Val Acc: 0.9290
Epoch [27/100] - Train Loss: 0.1266, Train Acc: 0.9520, Val Loss:
0.2061, Val Acc: 0.9284
Epoch [28/100] - Train Loss: 0.1255, Train Acc: 0.9537, Val Loss:
0.2079, Val Acc: 0.9316
Epoch [29/100] - Train Loss: 0.1255, Train Acc: 0.9544, Val Loss:
0.2093, Val Acc: 0.9268
Epoch [30/100] - Train Loss: 0.1248, Train Acc: 0.9545, Val Loss:
0.2093, Val Acc: 0.9306
Epoch [31/100] - Train Loss: 0.1247, Train Acc: 0.9540, Val Loss:
0.2103, Val Acc: 0.9266
Epoch [32/100] - Train Loss: 0.1253, Train Acc: 0.9536, Val Loss:
0.2099, Val Acc: 0.9252
Epoch [33/100] - Train Loss: 0.1247, Train Acc: 0.9538, Val Loss:
0.2137, Val Acc: 0.9286
Epoch [34/100] - Train Loss: 0.1233, Train Acc: 0.9549, Val Loss:
0.2127, Val Acc: 0.9280
Epoch [35/100] - Train Loss: 0.1265, Train Acc: 0.9534, Val Loss:
0.2084, Val Acc: 0.9278
Epoch [36/100] - Train Loss: 0.1243, Train Acc: 0.9541, Val Loss:
```

```
0.2135, Val Acc: 0.9290
Epoch [37/100] - Train Loss: 0.1239, Train Acc: 0.9540, Val Loss:
0.2099, Val Acc: 0.9282
Epoch [38/100] - Train Loss: 0.1241, Train Acc: 0.9537, Val Loss:
0.2102, Val Acc: 0.9290
Epoch [39/100] - Train Loss: 0.1222, Train Acc: 0.9546, Val Loss:
0.2102, Val Acc: 0.9304
Epoch [40/100] - Train Loss: 0.1231, Train Acc: 0.9550, Val Loss:
0.2162, Val Acc: 0.9296
Epoch [41/100] - Train Loss: 0.1251, Train Acc: 0.9543, Val Loss:
0.2130, Val Acc: 0.9260
Epoch [42/100] - Train Loss: 0.1251, Train Acc: 0.9537, Val Loss:
0.2112, Val Acc: 0.9288
Epoch [43/100] - Train Loss: 0.1239, Train Acc: 0.9541, Val Loss:
0.2144, Val Acc: 0.9260
Epoch [44/100] - Train Loss: 0.1246, Train Acc: 0.9532, Val Loss:
0.2111, Val Acc: 0.9256

---------------------------------------------------------------------
-----
KeyboardInterrupt                         Traceback (most recent call
last)
Cell In[3], line 153
    150 train_data_loader, validation_data_loader =
get_fashion_mnist_data()
    152 # Start training and evaluating
--> 153 train_and_evaluate(wandb.config.epochs, train_data_loader,
validation_data_loader)
    155 wandb.finish()

Cell In[3], line 90, in train_and_evaluate(epochs, train_data_loader,
validation_data_loader)
     88 model.train()
     89 train_loss, train_correct, train_total = 0, 0, 0
---> 90 for images, labels in train_data_loader:
     91     images, labels = images.to(device), labels.to(device)
     92     optimizer.zero_grad()

File
/usr/local/lib/python3.10/dist-packages/torch/utils/data/dataloader.py
:631, in _BaseDataLoaderIter.__next__(self)
    628 if self._sampler_iter is None:
    629     # TODO(https://github.com/pytorch/pytorch/issues/76750)
    630     self._reset()  # type: ignore[call-arg]
--> 631 data = self._next_data()
    632 self._num_yielded += 1
    633 if self._dataset_kind == _DatasetKind.Iterable and \
    634         self._IterableDataset_len_called is not None and \
    635         self._num_yielded > self._IterableDataset_len_called:
```

```
File
/usr/local/lib/python3.10/dist-packages/torch/utils/data/dataloader.py
:675, in _SingleProcessDataLoaderIter._next_data(self)
    673 def _next_data(self):
    674     index = self._next_index()  # may raise StopIteration
--> 675     data = self._dataset_fetcher.fetch(index)  # may raise
StopIteration
    676     if self._pin_memory:
    677         data = _utils.pin_memory.pin_memory(data,
self._pin_memory_device)

File
/usr/local/lib/python3.10/dist-packages/torch/utils/data/_utils/fetch.
py:54, in _MapDatasetFetcher.fetch(self, possibly_batched_index)
     52 else:
     53     data = self.dataset[possibly_batched_index]
---> 54 return self.collate_fn(data)

File
/usr/local/lib/python3.10/dist-packages/torch/utils/data/_utils/collat
e.py:277, in default_collate(batch)
    216 def default_collate(batch):
    217     r"""
    218     Take in a batch of data and put the elements within the
batch into a tensor with an additional outer dimension - batch size.
    219
   (...)
    275         >>> default_collate(batch)  # Handle `CustomType`
automatically
    276     """
--> 277     return collate(batch,
collate_fn_map=default_collate_fn_map)

File
/usr/local/lib/python3.10/dist-packages/torch/utils/data/_utils/collat
e.py:144, in collate(batch, collate_fn_map)
    141 transposed = list(zip(*batch))  # It may be accessed twice, so
we use a list.
    143 if isinstance(elem, tuple):
--> 144     return [collate(samples, collate_fn_map=collate_fn_map)
for samples in transposed]  # Backwards compatibility.
    145 else:
    146     try:

File
/usr/local/lib/python3.10/dist-packages/torch/utils/data/_utils/collat
e.py:144, in <listcomp>(.0)
    141 transposed = list(zip(*batch))  # It may be accessed twice, so
we use a list.
    143 if isinstance(elem, tuple):
```

```
--> 144     return [collate(samples, collate_fn_map=collate_fn_map)
for samples in transposed]  # Backwards compatibility.
    145 else:
    146     try:

File
/usr/local/lib/python3.10/dist-packages/torch/utils/data/_utils/collat
e.py:121, in collate(batch, collate_fn_map)
    119 if collate_fn_map is not None:
    120     if elem_type in collate_fn_map:
--> 121         return collate_fn_map[elem_type](batch,
collate_fn_map=collate_fn_map)
    123     for collate_type in collate_fn_map:
    124         if isinstance(elem, collate_type):

File
/usr/local/lib/python3.10/dist-packages/torch/utils/data/_utils/collat
e.py:174, in collate_tensor_fn(batch, collate_fn_map)
    172     storage = elem._typed_storage()._new_shared(numel,
device=elem.device)
    173     out = elem.new(storage).resize_(len(batch),
*list(elem.size()))
--> 174 return torch.stack(batch, 0, out=out)

KeyboardInterrupt:
```