```python
import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
from torch.utils.tensorboard import SummaryWriter
from torchsummary import summary
from torchvision.models import resnet50

import argparse
import time
import copy
from tqdm import tqdm
import os.path as osp
import sys

from utils import Config
# from model import model
from data import get_dataloader
from datetime import datetime
import csv
import numpy as np


def train_model(dataloader, model, criterion, optimizer, device, num_epochs, dataset_size):

    ts = datetime.now().strftime('%Y%m%d_%H%M%S')
    writer = SummaryWriter(f'logs/polyvore/{ts}')

    since = time.time()
    best_model = model
    best_model_wts = copy.deepcopy(model.state_dict())
    best_acc = 0.0
    # exp_lr_scheduler = optim.lr_scheduler.StepLR(
    #     optimizer, step_size=5, gamma=0.1)

    for epoch in range(num_epochs):
        print('Epoch [{}/{}]'.format(epoch, num_epochs - 1))
        print('-' * 10)

        # exp_lr_scheduler.step()

        for phase in ['train', 'val']:
            print('   Phase: {}'.format(phase))
            print('   ', '-' * 10)

            if phase == 'train':
                model.train()
            else:
                model.eval()

            running_loss = 0.0
            running_corrects = 0
            total = 0

            for inputs, labels in tqdm(dataloaders[phase]):
                inputs = inputs.to(device)
                labels = labels.to(device)
                optimizer.zero_grad()

                with torch.set_grad_enabled(phase == 'train'):
                    outputs = model(inputs)
                    _, pred = outputs.max(1)
                    total += labels.size(0)
                    loss = criterion(outputs, labels)
```

```
                if phase == 'train':
                    loss.backward()
                    optimizer.step()

                running_loss += loss.item() * inputs.size(0)
                running_corrects += pred.eq(labels).sum().item()

            epoch_loss = running_loss / dataset_size[phase]
            epoch_acc = running_corrects * 100 / total

            writer.add_scalars('data/loss', {phase: epoch_acc}, epoch+1)
            writer.add_scalars('data/accuracy', {phase: epoch_loss}, epoch+1)

            print('\n   Loss: {:.4f} Acc: {:.4f}'.format(
                epoch_loss, epoch_acc))

            if phase == 'val' and epoch_acc > best_acc:
                best_acc = epoch_acc
                best_model_wts = copy.deepcopy(model.state_dict())
                best_model = model

        torch.save(best_model, osp.join(
            Config['root_path'], Config['checkpoint_path'], 'categorical-model.pth'))
        print('Model saved at: {}'.format(osp.join(
            Config['root_path'], Config['checkpoint_path'], 'categorical-model.pth')))

    time_elapsed = time.time() - since
    print('Time taken to complete training: {:0f}m {:0f}s'.format(
        time_elapsed // 60, time_elapsed % 60))
    print('Best Validation Accuracy: {:.4f}'.format(best_acc))
    writer.close()

    model.eval()
    corrects = 0
    test_loader = dataloaders['test']

    with open(osp.join(Config['root_path'], "category.csv"), 'w') as csv_file:
        writer = csv.writer(csv_file, delimiter=' ')
        with torch.no_grad():
            for i, (inputs, labels, item_ids) in tqdm(enumerate(test_loader), 0):

                inputs = inputs.to(device)
                labels = labels.to(device)

                outputs = model(inputs)
                _, pred = outputs.max(1)

                total += labels.size(0)
                corrects += pred.eq(labels).sum().item()

                preds_list = pred.cpu().detach().numpy().tolist()
                labels_list = labels.cpu().detach().numpy().tolist()

                stacked_data = np.column_stack(
                    (list(item_ids), preds_list, labels_list))
                writer.writerows(stacked_data)

            final_acc = corrects * 100 / total
            print('Best Test Accuracy: {:.4f}'.format(final_acc))


if __name__ == '__main__':

    dataloaders, classes, dataset_size = get_dataloader(
        debug=Config['debug'], batch_size=Config['batch_size'], num_workers=Config['num_workers'])
```

```
model = resnet50(pretrained=True)

for param in model.parameters():
    param.requires_grad = False

num_ftrs = model.fc.in_features

model.fc = nn.Sequential(

    nn.Linear(num_ftrs, 1024),
    nn.ReLU(inplace=True),
    nn.Dropout2d(p=.5),

    nn.Linear(1024, 512),
    nn.ReLU(inplace=True),
    nn.Dropout2d(p=.5),

    nn.Linear(512, 153)
)

criterion = nn.CrossEntropyLoss()

optimizer = optim.Adam(
    model.fc.parameters(), lr=Config['learning_rate'], weight_decay=1e-5)

device = torch.device('cuda:0' if torch.cuda.is_available()
                      and Config['use_cuda'] else 'cpu')
model.to(device)

# To write the model summary and network architecture to a text file
original_stdout = sys.stdout  # Save a reference to the original standard output

with open(osp.join(Config['root_path'], 'categorical-model.txt'), 'w') as f:
    sys.stdout = f  # Change the standard output to the file we created.
    print('------------ MODEL SUMMARY ---------------\n')
    summary(model, input_size=(3, 224, 224), batch_size=-1)
    print('\n-----------------------------------------\n')
    print('------------ MODEL ---------------\n')
    print(model)
    print('\n---------------- END OF FILE ------------\n')
    sys.stdout = original_stdout  # Reset the standard output to its original value

train_model(dataloaders, model, criterion, optimizer, device,
            num_epochs=Config['num_epochs'], dataset_size=dataset_size)
```