NAME : NAGAVENI L G

SRN:PES2UG21CS315

SEC:5F

MI LAB-3-CNN

```
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision
import torchvision.transforms as transforms
from torch.utils.data import DataLoader
import torch.nn.functional as F

# Dataset Preparation
def prepareData(classes_list, class_dict):
    transform = transforms.Compose([
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.5, 0.5, 0.5], std=[0.5, 0.5, 0.5])
    ])
    # Classes to include
    class_indices_to_keep = [0, 1, 2]

    # Training Dataset
    train_dataset = torchvision.datasets.CIFAR10(root='./data', train=True,
transform=transform, download=True)
    for i in range(len(train_dataset)):
        if train_dataset.targets[i] in classes_list:
            train_dataset.targets[i] = class_dict[train_dataset.targets[i]]
        else:
            train_dataset.targets[i] = -1
    train_dataset = torch.utils.data.Subset(train_dataset, [i for i in
range(len(train_dataset)) if train_dataset.targets[i] in
class_indices_to_keep])
    train_loader = DataLoader(train_dataset, batch_size=16, shuffle=True)

    # Testing Dataset
    test_dataset = torchvision.datasets.CIFAR10(root='./data', train=False,
transform=transform, download=True)
    for i in range(len(test_dataset)):
        if test_dataset.targets[i] in classes_list:
            test_dataset.targets[i] = class_dict[test_dataset.targets[i]]
        else:
            test_dataset.targets[i] = -1
    test_dataset = torch.utils.data.Subset(test_dataset, [i for i in
range(len(test_dataset)) if test_dataset.targets[i] in class_indices_to_keep])
    test_loader = DataLoader(test_dataset, batch_size=16, shuffle=False)
```

```
    return train_loader, test_loader

# The CNN Model Class
# The CNN Model Class
class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        self.conv1 = nn.Conv2d(in_channels=3, out_channels=16, kernel_size=3,
padding=1)
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2)
        self.conv2 = nn.Conv2d(in_channels=16, out_channels=32, kernel_size=3,
padding=1)
        self.fc1 = nn.Linear(in_features=32 * 8 * 8, out_features=128)
        self.fc2 = nn.Linear(in_features=128, out_features=3)  # 3 classes

        # Define the loss criterion (cross-entropy) and optimizer (Adam) here
        self.criterion = nn.CrossEntropyLoss()
        self.optimizer = optim.Adam(self.parameters(), lr=0.002)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(-1, 32 * 8 * 8)  # Flatten the tensor
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return x


    # Set Values of self.optimizer and self.criterion
    def setCriterionAndOptimizer(self):
        self.optimizer = optim.Adam(self.parameters(), lr=0.002)
        self.criterion = nn.CrossEntropyLoss()


def train(model, train_loader):
    model.train()  # Set the model to training mode
    running_loss = 0.0
    correct_predictions = 0
    total_samples = 0
    criterion = model.criterion

    for inputs, labels in train_loader:
        model.optimizer.zero_grad()  # Zero the parameter gradients

        # Forward pass
        outputs = model(inputs)

        # Calculate loss
        loss = criterion(outputs, labels)
```

```
        # Backward pass and optimize
        loss.backward()
        model.optimizer.step()

        # Update statistics
        running_loss += loss.item()
        _, predicted = torch.max(outputs, 1)
        correct_predictions += (predicted == labels).sum().item()
        total_samples += labels.size(0)

    # Calculate training loss and accuracy
    train_loss = running_loss / len(train_loader)
    train_accuracy = 100.0 * correct_predictions / total_samples

    print(f'Training Loss: {train_loss:.4f}, Training Accuracy:
{train_accuracy:.2f}%')
    return train_accuracy

# Implement evaluation here
# Input: 1) model: CNN object
#        2) test_loader: DataLoader object for testing
# Output: 1) test_accuracy: float
def evaluate(model, test_loader):
    model.eval()  # Set the model to evaluation mode
    running_loss = 0.0
    correct_predictions = 0
    total_samples = 0
    criterion = model.criterion

    with torch.no_grad():
        for inputs, labels in test_loader:

            # Forward pass
            outputs = model(inputs)

            # Calculate loss
            loss = criterion(outputs, labels)

            # Update statistics
            running_loss += loss.item()
            _, predicted = torch.max(outputs, 1)
            correct_predictions += (predicted == labels).sum().item()
            total_samples += labels.size(0)

    # Calculate validation/test loss and accuracy
    test_loss = running_loss / len(test_loader)
    test_accuracy = 100.0 * correct_predictions / total_samples
```

```
    print(f'Test Loss: {test_loss:.4f}, Test Accuracy: {test_accuracy:.2f}%')
    return test_accuracy
```

OUTPUT:

```
(venv) PS C:\Users\Praka\OneDrive\Documents\5thSem\MI\CNN (Student)> python Test.py --ID EC_F_PES2UG21CS315_Lab3
Files already downloaded and verified
Files already downloaded and verified
Training Loss: 0.5867, Training Accuracy: 75.19%
Test Loss: 0.4709, Test Accuracy: 81.20%
Test Case 1 for the function train PASSED
Test Case 2 for the function evaluate PASSED
Test Case 3 for the function __init__ PASSED
(venv) PS C:\Users\Praka\OneDrive\Documents\5thSem\MI\CNN (Student)>
```