Artificial Intelligence Final Report Assignment 問題1 (Problem 1)

レポート解答用紙 (Report Answer Sheet)

Group Leader

学生証番号 (Student ID):

名前(Name):

Group Members

学生証番号 (Student ID): 21522544

名前(Name): Nguyễn Ngọc Thanh Sang

学生証番号 (Student ID): 21522515

名前(Name): Nguyễn Việt Quang

学生証番号 (Student ID):

名前(Name):

問題1 (Problem 1)のレポート

**Program:**

# -\*- coding: utf-8 -\*-

"""Problem 1.ipynb

Automatically generated by Colab.

Original file is located at

    https://colab.research.google.com/drive/1ch8SkfFA9NXeFHrnStyue-wkE4YtDdpZ

"""

import numpy as np

import matplotlib.pyplot as plt

import torch

import torch.nn as nn

import torch.nn.functional as F

import torch.optim as optim

import torchvision as tv

transform\_train = tv.transforms.Compose([

    tv.transforms.RandomCrop(32, padding=4),

    tv.transforms.RandomHorizontalFlip(),

    tv.transforms.ToTensor(),

    tv.transforms.Normalize((0.4914, 0.4822, 0.4465), (0.2023, 0.1994, 0.2010)),

])

transform\_test = tv.transforms.Compose([

    tv.transforms.ToTensor(),

    tv.transforms.Normalize((0.4914, 0.4822, 0.4465), (0.2023, 0.1994, 0.2010)),

])

train\_dataset = tv.datasets.CIFAR10(root='./', train=True, download=True, transform=transform\_train)

test\_dataset = tv.datasets.CIFAR10(root='./', train=False, download=True, transform=transform\_test)

train\_loader = torch.utils.data.DataLoader(train\_dataset, batch\_size=100, shuffle=True)

test\_loader = torch.utils.data.DataLoader(test\_dataset, batch\_size=100, shuffle=False)

MODELNAME = "cifar10\_vgg16.model"

EPOCHS = 50

DEVICE = "cuda" if torch.cuda.is\_available() else "cpu"

class VGG16(nn.Module):

    def \_\_init\_\_(self):

        super(VGG16, self).\_\_init\_\_()

        self.features = nn.Sequential(

            nn.Conv2d(3, 64, kernel\_size=3, padding=1),

            nn.ReLU(inplace=True),

            nn.Conv2d(64, 64, kernel\_size=3, padding=1),

            nn.ReLU(inplace=True),

            nn.MaxPool2d(kernel\_size=2, stride=2),

            nn.Conv2d(64, 128, kernel\_size=3, padding=1),

            nn.ReLU(inplace=True),

            nn.Conv2d(128, 128, kernel\_size=3, padding=1),

            nn.ReLU(inplace=True),

            nn.MaxPool2d(kernel\_size=2, stride=2),

            nn.Conv2d(128, 256, kernel\_size=3, padding=1),

            nn.ReLU(inplace=True),

            nn.Conv2d(256, 256, kernel\_size=3, padding=1),

            nn.ReLU(inplace=True),

            nn.Conv2d(256, 256, kernel\_size=3, padding=1),

            nn.ReLU(inplace=True),

            nn.MaxPool2d(kernel\_size=2, stride=2),

            nn.Conv2d(256, 512, kernel\_size=3, padding=1),

            nn.ReLU(inplace=True),

            nn.Conv2d(512, 512, kernel\_size=3, padding=1),

            nn.ReLU(inplace=True),

            nn.Conv2d(512, 512, kernel\_size=3, padding=1),

            nn.ReLU(inplace=True),

            nn.MaxPool2d(kernel\_size=2, stride=2),

            nn.Conv2d(512, 512, kernel\_size=3, padding=1),

            nn.ReLU(inplace=True),

            nn.Conv2d(512, 512, kernel\_size=3, padding=1),

            nn.ReLU(inplace=True),

            nn.Conv2d(512, 512, kernel\_size=3, padding=1),

            nn.ReLU(inplace=True),

            nn.MaxPool2d(kernel\_size=2, stride=2),

        )

        self.classifier = nn.Sequential(

            nn.Linear(512 \* 1 \* 1, 4096),

            nn.ReLU(inplace=True),

            nn.Dropout(),

            nn.Linear(4096, 4096),

            nn.ReLU(inplace=True),

            nn.Dropout(),

            nn.Linear(4096, 10),

        )

    def forward(self, x):

        x = self.features(x)

        x = x.view(x.size(0), -1)

        x = self.classifier(x)

        return x

model = VGG16().to(DEVICE)

def train():

    optimizer = optim.Adam(model.parameters(), lr=0.0001)

    scheduler = torch.optim.lr\_scheduler.StepLR(optimizer, step\_size=20, gamma=0.1)

    criterion = nn.CrossEntropyLoss()

    for epoch in range(EPOCHS):

        model.train()

        running\_loss = 0.0

        for images, labels in train\_loader:

            images = images.to(DEVICE)

            labels = labels.to(DEVICE)

            optimizer.zero\_grad()

            outputs = model(images)

            loss = criterion(outputs, labels)

            loss.backward()

            optimizer.step()

            running\_loss += loss.item()

        scheduler.step()

        print(f"Epoch {epoch+1}/{EPOCHS}, Loss: {running\_loss}")

    torch.save(model.state\_dict(), MODELNAME)

def test():

    model.load\_state\_dict(torch.load(MODELNAME))

    model.eval()

    correct = 0

    total = len(test\_loader.dataset)

    with torch.no\_grad():

        for images, labels in test\_loader:

            images = images.to(DEVICE)

            labels = labels.to(DEVICE)

            outputs = model(images)

            \_, predicted = torch.max(outputs.data, 1)

            correct += (predicted == labels).sum().item()

    print(f'Correct: {correct}')

    print(f'Total: {total}')

    print(f"Accuracy: {100 \* correct / total}%")

train()

test()

**Excution Results:**

Epoch 1/50, Loss: 1001.2247166633606

Epoch 2/50, Loss: 858.4091100692749

Epoch 3/50, Loss: 731.9600386619568

Epoch 4/50, Loss: 640.6352415084839

Epoch 5/50, Loss: 567.7969959378242

Epoch 6/50, Loss: 501.6279647350311

Epoch 7/50, Loss: 446.4034964442253

Epoch 8/50, Loss: 413.7774193882942

Epoch 9/50, Loss: 381.73151218891144

Epoch 10/50, Loss: 351.15896144509315

Epoch 11/50, Loss: 327.7224320471287

Epoch 12/50, Loss: 306.2018339931965

Epoch 13/50, Loss: 286.26026752591133

Epoch 14/50, Loss: 270.93661999702454

Epoch 15/50, Loss: 253.32150445878506

Epoch 16/50, Loss: 241.56750513613224

Epoch 17/50, Loss: 227.0031936466694

Epoch 18/50, Loss: 215.1582535803318

Epoch 19/50, Loss: 203.3438258767128

Epoch 20/50, Loss: 192.63084088265896

Epoch 21/50, Loss: 143.47254614531994

Epoch 22/50, Loss: 134.13038853555918

Epoch 23/50, Loss: 128.9290680065751

Epoch 24/50, Loss: 125.47544783353806

Epoch 25/50, Loss: 122.49192802608013

Epoch 26/50, Loss: 117.81385842710733

Epoch 27/50, Loss: 117.59349628537893

Epoch 28/50, Loss: 112.67623288929462

Epoch 29/50, Loss: 112.06653132289648

Epoch 30/50, Loss: 108.09835635870695

Epoch 31/50, Loss: 107.91248666495085

Epoch 32/50, Loss: 103.83903296664357

Epoch 33/50, Loss: 102.3993116542697

Epoch 34/50, Loss: 100.79486930370331

Epoch 35/50, Loss: 98.0325181260705

Epoch 36/50, Loss: 95.63151026144624

Epoch 37/50, Loss: 94.16126674413681

Epoch 38/50, Loss: 92.07266606017947

Epoch 39/50, Loss: 91.71733235567808

Epoch 40/50, Loss: 89.0015177205205

Epoch 41/50, Loss: 83.85680890642107

Epoch 42/50, Loss: 82.78544940054417

Epoch 43/50, Loss: 83.21741110831499

Epoch 44/50, Loss: 81.17129550129175

Epoch 45/50, Loss: 80.66633601486683

Epoch 46/50, Loss: 81.29912203922868

Epoch 47/50, Loss: 80.95468677207828

Epoch 48/50, Loss: 81.43710922822356

Epoch 49/50, Loss: 80.30242469906807

Epoch 50/50, Loss: 79.96395723614842

Correct: 8625

Total: 10000

Accuracy: 86.25%

**Explanation:**

1. **Data Transformation:**

* **Training Data Transformation:**
  + RandomCrop(32, padding=4): Randomly crops the image to 32x32 pixels with padding of 4 pixels.
  + RandomHorizontalFlip(): Randomly flips the image horizontally with a probability of 0.5.
  + ToTensor(): Converts the image to a PyTorch tensor.
  + Normalize((0.4914, 0.4822, 0.4465), (0.2023, 0.1994, 0.2010)): Normalizes the tensor with the given mean and standard deviation values.
* **Test Data Transformation:**
  + Only ToTensor() and Normalize() are applied since data augmentation is not needed during testing.

1. **VGG16 Architecture:**

* **Convolutional Layers**

VGG16 uses a stack of convolutional layers with small receptive fields of 3x3, which are padded to maintain spatial resolution. The convolutional layers are organized into blocks, each followed by a max-pooling layer. The network increases the number of filters as the spatial resolution decreases.

* + **Block 1:**
    - Conv Layer: 3x3, 64 filters, stride 1, padding 1
    - ReLU Activation
    - Conv Layer: 3x3, 64 filters, stride 1, padding 1
    - ReLU Activation
    - Max Pooling: 2x2, stride 2
  + **Block 2:**
    - Conv Layer: 3x3, 128 filters, stride 1, padding 1
    - ReLU Activation
    - Conv Layer: 3x3, 128 filters, stride 1, padding 1
    - ReLU Activation
    - Max Pooling: 2x2, stride 2
  + **Block 3:**
    - Conv Layer: 3x3, 256 filters, stride 1, padding 1
    - ReLU Activation
    - Conv Layer: 3x3, 256 filters, stride 1, padding 1
    - ReLU Activation
    - Conv Layer: 3x3, 256 filters, stride 1, padding 1
    - ReLU Activation
    - Max Pooling: 2x2, stride 2
  + **Block 4:**
    - Conv Layer: 3x3, 512 filters, stride 1, padding 1
    - ReLU Activation
    - Conv Layer: 3x3, 512 filters, stride 1, padding 1
    - ReLU Activation
    - Conv Layer: 3x3, 512 filters, stride 1, padding 1
    - ReLU Activation
    - Max Pooling: 2x2, stride 2
  + **Block 5:**
    - Conv Layer: 3x3, 512 filters, stride 1, padding 1
    - ReLU Activation
    - Conv Layer: 3x3, 512 filters, stride 1, padding 1
    - ReLU Activation
    - Conv Layer: 3x3, 512 filters, stride 1, padding 1
    - ReLU Activation
    - Max Pooling: 2x2, stride 2
* **Fully Connected Layers**

After the convolutional layers, the output is flattened and passed through a series of fully connected (dense) layers.

* + **Flattening:**
    - The output from the final max-pooling layer is flattened into a vector.
  + **Fully Connected Layers:**
    - FC Layer: 4096 neurons
    - ReLU Activation
    - Dropout
    - FC Layer: 4096 neurons
    - ReLU Activation
    - Dropout
    - FC Layer: 10 neurons (number of classes in CIFAR-10)
  + **Output Layer:**
    - The final fully connected layer outputs a vector of size 10, corresponding to the 10 classes in CIFAR-10.

1. **Hyperparameters:**

Hyperparameters are parameters that are set before the training process and determine the network architecture and training behavior.

* **Learning Rate (lr):** 0.0001
  + Determines the step size for the optimizer to adjust weights during training.
* **Batch Size:** 100
  + Number of samples processed before updating the model parameters.
* **Epochs:** 50
  + Number of complete passes through the training dataset.
* **StepLR:** step\_size=20, gamma=0.1
  + Learning rate scheduler that decreases the learning rate by a factor of gamma every step\_size epochs.

1. **Training Process:**

* **Optimizer and Loss Function:**
  + ‘Adam’ optimizer is used for its adaptive learning rate properties.
  + ‘StepLR’ scheduler adjusts the learning rate during training.
  + ‘CrossEntropyLoss’ is used as the loss function for classification.
* **Training Loop:**
  + The model is set to training mode.
  + For each batch, images and labels are moved to the appropriate device (CPU or GPU).
  + Gradients are zeroed, the model makes predictions, loss is computed, and the optimizer updates the model parameters.
  + Learning rate scheduler steps at the end of each epoch.
  + Running loss is printed for monitoring training progress.

1. **Testing Process:**

* **Evaluation Loop:**
  + Model weights are loaded from the saved state.
  + The model is set to evaluation mode.
  + Predictions are made on the test dataset without updating gradients.
  + Accuracy is computed as the percentage of correctly classified samples.