resnet-cifar10

October 3, 2023

1 Training SimpleNN on CIFAR-10

In this project, you will use the SimpleNN model to perform image classification on CIFAR-10. CIFAR-10 orginally contains 60K images from 10 categories. We split it into 45K/5K/10K images to serve as train/valiation/test set. We only release the ground-truth labels of training/validation dataset to you.

1.1 Step 0: Set up the SimpleNN model

As you have practiced to implement simple neural networks in Homework 1, we just prepare the implementation for you.

[ ]:
```
# import necessary dependencies
import argparse
import os, sys
import time
import datetime
from tqdm import tqdm_notebook as tqdm

import torch
import torch.nn as nn
import torch.nn.functional as F
```

[2]:
```
img1 = torch.zeros((128, 8, 99, 99))
img2 = torch.zeros(img1.size())
img3 = torch.cat((img1, img2), 1)
print(img3.size())
```

```
torch.Size([128, 16, 99, 99])
```

[3]:
```
def my_swish(x):
    return torch.sigmoid(x) * x
## implementing swish


class Residual(nn.Module):
    def __init__(self, in_channels, out_channels, stride):
        """
        :param in_channels: number of channels for the input
        :param out_channels: number of channels for the output
        :param stride: stride of first conv2d block
        """
        # conv1: downsamples feature map when stride != 1 (padding = 1 to␣
↪ensure correct shape),
        # batch norm then ReLU
        super(Residual, self).__init__()
        self.conv1 = nn.Sequential(nn.Conv2d(in_channels, out_channels, 3,␣
↪stride=stride, padding=1),
                                   nn.BatchNorm2d(num_features=out_channels,␣
↪eps=1e-05, momentum=0.1),
                                   nn.ReLU())
        # conv2: doesn't downsample feature map since already performed in self.
↪conv1, batch norm,
        # padding=1 to ensure correct output shape (conv2 outputs same shape as␣
↪conv1's output)
        self.conv2 = nn.Sequential(nn.Conv2d(out_channels, out_channels, 3,␣
↪stride=1, padding=1),
                                   nn.BatchNorm2d(num_features=out_channels,␣
↪eps=1e-05, momentum=0.1))
        self.relu2 = nn.ReLU()  # applied after adding x in the forward function

        # in case we need these
        self.in_channels = in_channels
        self.out_channels = out_channels

    def forward(self, x):
        out = self.conv1(x)
        out = self.conv2(out)

        # if number of channels for input/output mismatch, must downsample
        # if x's num_channels doesn't match out's desired n_channels:
        if x.size(dim=1) != self.out_channels:
            x = x[:, :, ::2, ::2]  # downsample feature map by a factor of 2
            # following option (A), concatenate a zeros to increase n_channels␣
↪(bottom-left page776 of IEEE version).
            zzzzz = torch.zeros(x.size(), device=x.device)  # get zeros of same␣
↪size as x
            x_new = torch.cat((x, zzzzz), 1)  # concatenate channels
        else:
            x_new = x

        assert out.size() == x_new.size(), "out.size=={:s} != x.size=={:s}".
↪format(str(list(out.size())), str(list(x.size())))
        return self.relu2(out + x_new)
```

```
class ResNet(nn.Module):
    def __init__(self):
        super(ResNet, self).__init__()
        # implementing ResNet-20, so n=3
        # first conv to get 16-chan output, then (6x conv(in_channel=32), 6x␣
 ↪conv(in_channel=16),
        # 6x conv(in_channel=8)), then average pooling, then FC, finally softmax
        # maintain feature map size, increase n_channels to 16 after 1st␣
 ↪convolution
        self.conv_initial = nn.Sequential(nn.Conv2d(3, 16, 3, stride=1,␣
 ↪padding=1),
                                          nn.BatchNorm2d(num_features=16,␣
 ↪eps=1e-05, momentum=0.1),
                                          nn.ReLU())  # output 32*32 feature map
        # no max pool in this ResNet

        # first six conv layers (ReLU, batchNorm are applied within Residual)
        self.conv32_0 = Residual(in_channels=16, out_channels=16, stride=1) #␣
 ↪output 32*32 feature map
        self.conv32_2 = Residual(in_channels=16, out_channels=16, stride=1) #␣
 ↪output 32*32 feature map
        self.conv32_4 = Residual(in_channels=16, out_channels=16, stride=1) #␣
 ↪output 32*32 feature map

        # feature map is 16*16 after conv16_0
        # next six conv layers, (ReLU, batchNorm are applied within Residual)
        self.conv16_0 = Residual(in_channels=16, out_channels=32, stride=2) #␣
 ↪output 16*16 feature map
        self.conv16_2 = Residual(in_channels=32, out_channels=32, stride=1) #␣
 ↪output 16*16 feature map
        self.conv16_4 = Residual(in_channels=32, out_channels=32, stride=1) #␣
 ↪output 16*16 feature map

        # feature map decreases to 8*8 after conv8_0
        # last six conv layers, (ReLU, batchNorm are applied within Residual)
        self.conv8_0 = Residual(in_channels=32, out_channels=64, stride=2) #␣
 ↪output 8*8 feature map
        self.conv8_2 = Residual(in_channels=64, out_channels=64, stride=1) #␣
 ↪output 8*8 feature map
        self.conv8_4 = Residual(in_channels=64, out_channels=64, stride=1) #␣
 ↪output 8*8 feature map
        # output after self.conv8_4 should be 64*8*8

        # global average pooling (64*(8*8) -> 64*(1))
        self.glbal_avg_pool = nn.AvgPool2d(kernel_size=8)

        # fully-conneted layer, in_channels=64, out_channels=10
        self.fc = nn.Sequential(nn.Linear(64, 10))


    def forward(self, x):
        out = self.conv_initial(x)
        out = self.conv32_0(out)
        out = self.conv32_2(out)
        out = self.conv32_4(out)

        out = self.conv16_0(out)
        out = self.conv16_2(out)
        out = self.conv16_4(out)

        out = self.conv8_0(out)
        out = self.conv8_2(out)
        out = self.conv8_4(out)

        out = self.glbal_avg_pool(out)

        out = out.view(out.size(0), -1)  # flatten before FC
        out = self.fc(out)
        return out
```

1.1.1 Question (a)

Here is a sanity check to verify the implementation of SimpleNN. You need to: 1. Write down your code. 2. **In the PDF report**, give a brief description on how the code helps you know that SimpleNN is implemented correctly.

[4]:
```
#############################################
# your code here
# sanity check for the correctness of ResNet
test_in = torch.rand((1, 3, 32, 32))
test_model = ResNet()
test_out = test_model.forward(test_in)
print(test_out.shape)
#############################################
```

```
torch.Size([1, 10])
```

1.2 Step 1: Set up preprocessing functions

Preprocessing is very important as discussed in the lecture. You will need to write preprocessing functions with the help of *torchvision.transforms* in this step. You can find helpful tutorial/API at here.

1.2.1 Question (b)

For the question, you need to: 1. Complete the preprocessing code below. 2. **In the PDF report**, briefly describe what preprocessing operations you used and what are the purposes of them.

Hint: 1. Only two operations are necessary to complete the basic preprocessing here. 2. The raw input read from the dataset will be PIL images. 3. Data augmentation operations are not mendatory, but feel free to incorporate them if you want. 4. Reference value for mean/std of CIFAR-10 images (assuming the pixel values are within [0,1]): mean (RGB-format): (0.4914, 0.4822, 0.4465), std (RGB-format): (0.2023, 0.1994, 0.2010)

[5]:
```
# useful libraries
import torchvision
import torchvision.transforms as transforms

#############################################
# your code here
# specify preprocessing function ToTensor (PIL to tensor) and Normalise (mean␣
↪becomes 0, stdev becomes 1)
# please see pdf for more details

transform_train =  transforms.Compose([
    transforms.ToTensor(),  ## data augmentation below for training only
    transforms.RandomCrop(size=(32, 32), padding=4, pad_if_needed=False,␣
↪padding_mode='edge'),
    transforms.RandomHorizontalFlip(p=0.5),
    transforms.Normalize((0.4914, 0.4822, 0.4465), (0.2023, 0.1994, 0.2010))
    ])

# validation set: no data augmentation
transform_val = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.4914, 0.4822, 0.4465), (0.2023, 0.1994, 0.2010))
    ])
#############################################
```

1.3 Step 2: Set up dataset and dataloader

1.3.1 Question (c)

Set up the train/val datasets and dataloders that are to be used during the training. Check out the official API for more information about **torch.utils.data.DataLoader**.

Here, you need to: 1. Complete the code below.

[6]:
```
# do NOT change these
from tools.dataset import CIFAR10
from torch.utils.data import DataLoader
```

```python
# a few arguments, do NOT change these
DATA_ROOT = "./data"
TRAIN_BATCH_SIZE = 128
VAL_BATCH_SIZE = 100

#############################################
# your code here
# construct dataset
train_set = CIFAR10(
    root=DATA_ROOT,
    mode='train',
    download=True,
    transform=transform_train    # your code
)
val_set = CIFAR10(
    root=DATA_ROOT,
    mode='val',
    download=True,
    transform=transform_val    # your code
)

# construct dataloader
train_loader = DataLoader(
    train_set,
    batch_size=TRAIN_BATCH_SIZE,  # your code
    shuffle=True,     # your code
    num_workers=4
)
val_loader = DataLoader(
    val_set,
    batch_size=VAL_BATCH_SIZE,  # your code
    shuffle=True,     # your code
    num_workers=4
)
#############################################
```

```
Using downloaded and verified file: ./data/cifar10_trainval_F22.zip
Extracting ./data/cifar10_trainval_F22.zip to ./data
Files already downloaded and verified
Using downloaded and verified file: ./data/cifar10_trainval_F22.zip
Extracting ./data/cifar10_trainval_F22.zip to ./data
Files already downloaded and verified
```

1.4 Step 3: Instantiate your SimpleNN model and deploy it to GPU devices.

1.4.1 Question (d)

You may want to deploy your model to GPU device for efficient training. Please assign your model to GPU if possible. If you are training on a machine without GPUs, please deploy your model to CPUs.

Here, you need to: 1. Complete the code below. 2. **In the PDF report**, briefly describe how you verify that your model is indeed deployed on GPU. (Hint: check `nvidia-smi`.)

[7]:
```
# specify the device for computation
#############################################
# your code here

device = 'cuda' if torch.cuda.is_available() else 'cpu'
print(device)
# Construct our model by instantiating the class defined above
model = ResNet()
# Copy to CUDA device. This is very important.
model = model.to(device)
#############################################
```

```
cuda
```

1.5 Step 4: Set up the loss function and optimizer

Loss function/objective function is used to provide "feedback" for the neural networks. Typically, we use multi-class cross-entropy as the loss function for classification models. As for the optimizer, we will use SGD with momentum.

1.5.1 Question (e)

Here, you need to: 1. Set up the cross-entropy loss as the criterion. (Hint: there are implemented functions in **torch.nn**) 2. Specify a SGD optimizer with momentum. (Hint: there are implemented functions in **torch.optim**)

[8]:
```
import torch.nn as nn
import torch.optim as optim




##### my addition
L1_or_L2 = 'L2'
##### end of my addition




# hyperparameters, do NOT change right now
# initial learning rate
```

```
INITIAL_LR = 0.1

# momentum for optimizer
MOMENTUM = 0.9

# L2 regularization strength, 1e-4 in ResNet paper
REG = 1e-4  # 1e-4 is orig reg strength (L2 and L1)

#############################################
# your code here
# create loss function
criterion = nn.CrossEntropyLoss()

# Add optimizer
assert MOMENTUM == 0.9, "momentum isn't 0.9"

# using L2 optimisation
if L1_or_L2 == 'L2':
    optimizer = torch.optim.SGD(model.parameters(), lr=INITIAL_LR,␣
 ↪momentum=MOMENTUM, weight_decay=REG)

# using L1 optimisation, weight_decay=0 (no L2)
if L1_or_L2 == 'L1':
    optimizer = torch.optim.SGD(model.parameters(), lr=INITIAL_LR,␣
 ↪momentum=MOMENTUM, weight_decay=0)

assert L1_or_L2 == 'L1' or L1_or_L2 == 'L2', 'L1_or_L2 should be either str L1␣
 ↪or L2'
#############################################
```

1.6 Step 5: Start the training process.

1.6.1 Question (f)/(g)

Congratulations! You have completed all of the previous steps and it is time to train our neural network.

Here you need to: 1. Complete the training codes. 2. Actually perform the training.

Hint: Training a neural network usually repeats the following 4 steps:

i) Get a batch of data from the dataloader and copy it to your device (GPU).

ii) Do a forward pass to get the outputs from the neural network and compute the loss. Be careful about your inputs to the loss function. Are the inputs required to be the logits or softmax probabilities?)

iii) Do a backward pass (back-propagation) to compute gradients of all weights with respect to the loss.

iiii) Update the model weights with the optimizer.

You will also need to compute the accuracy of training/validation samples to track your model's performance over each epoch (the accuracy should be increasing as you train for more and more epochs).

[9]:
```
# some hyperparameters
# total number of training epochs
EPOCHS = 200

# DECAY_EPOCHS = 4  # disregarded for ResNet
DECAY = 0.1  # large decay, happens at 32k iterations and 48k iterations

# the folder where the trained model is saved
CHECKPOINT_FOLDER = "./saved_model"

# start the training/validation process
# the process should take about 5 minutes on a GTX 1070-Ti
# if the code is written efficiently.
best_val_acc = 0
current_learning_rate = INITIAL_LR



##### my addition
accuracy_arr_train = []
accuracy_arr_val = []
num_iterations = 0
# keeps track of the number of iterations (incremented each mini-batch)
##### end of my addition




print("==> Training starts!")
print("="*50)
for i in range(0, EPOCHS):
    # handle the learning rate scheduler.
    if i == 100 or i == 150:  # at around 32k, 48k iterations
        current_learning_rate = current_learning_rate * DECAY
        for param_group in optimizer.param_groups:
            param_group['lr'] = current_learning_rate
        print("Current learning rate has decayed to %f" %current_learning_rate)
    else:
        print("Current learning rate is %f" %current_learning_rate)
    if num_iterations > 64000:
        print("According to ResNet paper, should terminate")
    #######################
    # your code here
    # switch to train mode
    model.train()

    #######################

    print("Epoch %d, iteration %d:" %(i, num_iterations))
    # this help you compute the training accuracy
    total_examples = 0
    correct_examples = 0

    train_loss = 0 # track training loss if you want

    # Train the model for 1 epoch.
    for batch_idx, (inputs, targets) in enumerate(train_loader):

        ####################################
        # your code here
        # copy inputs to device
        inputs, targets = inputs.to(device), targets.to(device)

        # compute the output and loss
        y_pred = model(inputs)
        loss = criterion(y_pred, targets)



        ## L1 normalisation
##### my addition
        if L1_or_L2 == 'L1':
            for name, param in model.named_parameters():
                if 'bias' not in name:
                    loss += REG * torch.sum(torch.abs(param))
        num_iterations += 1
##### end of my addition




        train_loss += loss

        # zero the gradient
        optimizer.zero_grad()

        # backpropagation
        loss.backward()

        # apply gradient and update the weights
        optimizer.step()

        # count the number of correctly predicted samples in the current batch
        total_examples += len(targets.view(targets.size(0), -1))
        correct_examples += sum(torch.argmax(y_pred, dim=1, keepdim=False) ==␣
 ↪targets)

        ####################################

    avg_loss = train_loss / len(train_loader)
    avg_acc = correct_examples / total_examples
    print("Training loss: %.4f, Training accuracy: %.4f" %(avg_loss, avg_acc))



##### my addition
    accuracy_arr_train.append(avg_acc)
##### end of my addition



    # Validate on the validation dataset
    #######################
    # your code here
    # switch to eval mode
    model.eval()

    #######################

    # this helps you compute the validation accuracy
    total_examples = 0
    correct_examples = 0

    val_loss = 0 # again, track the validation loss if you want

    # disable gradient during validation, which can save GPU memory
    with torch.no_grad():
        for batch_idx, (inputs, targets) in enumerate(val_loader):
            ####################################
            # your code here
            # copy inputs to device
            inputs, targets = inputs.to(device), targets.to(device)

            # compute the output and loss
            y_pred = model(inputs)
            loss = criterion(y_pred, targets)
            val_loss += loss

            # count the number of correctly predicted samples in the current␣
 ↪batch
            total_examples += len(targets.view(targets.size(0), -1))
            correct_examples += sum(torch.argmax(y_pred, dim=1, keepdim=False)␣
 ↪== targets)
            ####################################

    avg_loss = val_loss / len(val_loader)
    avg_acc = correct_examples / total_examples
    print("Validation loss: %.4f, Validation accuracy: %.4f" % (avg_loss,␣
 ↪avg_acc))



##### my addition
    accuracy_arr_val.append(avg_acc)
##### end of my addition



    # save the model checkpoint
    if avg_acc > best_val_acc:
        best_val_acc = avg_acc
        if not os.path.exists(CHECKPOINT_FOLDER):
           os.makedirs(CHECKPOINT_FOLDER)
        print("Saving ...")
        state = {'state_dict': model.state_dict(),
                'epoch': i,
                'lr': current_learning_rate}
        torch.save(state, os.path.join(CHECKPOINT_FOLDER, 'resnet-cifar10.pth'))

    print('')

print("="*50)
print(f"==> Optimization finished! Best validation accuracy: {best_val_acc:.
 ↪4f}")
```

```
==> Training starts!
==================================================
Current learning rate is 0.100000
Epoch 0, iteration 0:
Training loss: 1.6476, Training accuracy: 0.3938
Validation loss: 1.3719, Validation accuracy: 0.4918
```

```
Saving …

Current learning rate is 0.100000
Epoch 1, iteration 352:
Training loss: 1.1589, Training accuracy: 0.5844
Validation loss: 1.0514, Validation accuracy: 0.6222
Saving …

Current learning rate is 0.100000
Epoch 2, iteration 704:
Training loss: 0.9249, Training accuracy: 0.6746
Validation loss: 0.8934, Validation accuracy: 0.6948
Saving …

Current learning rate is 0.100000
Epoch 3, iteration 1056:
Training loss: 0.7827, Training accuracy: 0.7261
Validation loss: 0.7777, Validation accuracy: 0.7262
Saving …

Current learning rate is 0.100000
Epoch 4, iteration 1408:
Training loss: 0.6912, Training accuracy: 0.7585
Validation loss: 0.7061, Validation accuracy: 0.7518
Saving …

Current learning rate is 0.100000
Epoch 5, iteration 1760:
Training loss: 0.6258, Training accuracy: 0.7836
Validation loss: 0.6088, Validation accuracy: 0.7936
Saving …

Current learning rate is 0.100000
Epoch 6, iteration 2112:
Training loss: 0.5774, Training accuracy: 0.7980
Validation loss: 0.6128, Validation accuracy: 0.7936

Current learning rate is 0.100000
Epoch 7, iteration 2464:
Training loss: 0.5428, Training accuracy: 0.8109
Validation loss: 0.5466, Validation accuracy: 0.8134
Saving …

Current learning rate is 0.100000
Epoch 8, iteration 2816:
Training loss: 0.5113, Training accuracy: 0.8211
Validation loss: 0.6246, Validation accuracy: 0.7888
```

```
Current learning rate is 0.100000
Epoch 9, iteration 3168:
Training loss: 0.4843, Training accuracy: 0.8334
Validation loss: 0.4873, Validation accuracy: 0.8278
Saving …

Current learning rate is 0.100000
Epoch 10, iteration 3520:
Training loss: 0.4597, Training accuracy: 0.8394
Validation loss: 0.5629, Validation accuracy: 0.8092

Current learning rate is 0.100000
Epoch 11, iteration 3872:
Training loss: 0.4458, Training accuracy: 0.8474
Validation loss: 0.5632, Validation accuracy: 0.8134

Current learning rate is 0.100000
Epoch 12, iteration 4224:
Training loss: 0.4282, Training accuracy: 0.8507
Validation loss: 0.5131, Validation accuracy: 0.8234

Current learning rate is 0.100000
Epoch 13, iteration 4576:
Training loss: 0.4107, Training accuracy: 0.8583
Validation loss: 0.5392, Validation accuracy: 0.8210

Current learning rate is 0.100000
Epoch 14, iteration 4928:
Training loss: 0.3986, Training accuracy: 0.8629
Validation loss: 0.5402, Validation accuracy: 0.8196

Current learning rate is 0.100000
Epoch 15, iteration 5280:
Training loss: 0.3914, Training accuracy: 0.8659
Validation loss: 0.5059, Validation accuracy: 0.8318
Saving …

Current learning rate is 0.100000
Epoch 16, iteration 5632:
Training loss: 0.3780, Training accuracy: 0.8682
Validation loss: 0.4649, Validation accuracy: 0.8424
Saving …

Current learning rate is 0.100000
Epoch 17, iteration 5984:
Training loss: 0.3703, Training accuracy: 0.8724
Validation loss: 0.6468, Validation accuracy: 0.8032
```

```
Current learning rate is 0.100000
Epoch 18, iteration 6336:
Training loss: 0.3618, Training accuracy: 0.8757
Validation loss: 0.5237, Validation accuracy: 0.8238

Current learning rate is 0.100000
Epoch 19, iteration 6688:
Training loss: 0.3534, Training accuracy: 0.8772
Validation loss: 0.4907, Validation accuracy: 0.8362

Current learning rate is 0.100000
Epoch 20, iteration 7040:
Training loss: 0.3472, Training accuracy: 0.8803
Validation loss: 0.4831, Validation accuracy: 0.8488
Saving …

Current learning rate is 0.100000
Epoch 21, iteration 7392:
Training loss: 0.3412, Training accuracy: 0.8816
Validation loss: 0.4594, Validation accuracy: 0.8468

Current learning rate is 0.100000
Epoch 22, iteration 7744:
Training loss: 0.3308, Training accuracy: 0.8868
Validation loss: 0.4972, Validation accuracy: 0.8378

Current learning rate is 0.100000
Epoch 23, iteration 8096:
Training loss: 0.3245, Training accuracy: 0.8872
Validation loss: 0.4584, Validation accuracy: 0.8500
Saving …

Current learning rate is 0.100000
Epoch 24, iteration 8448:
Training loss: 0.3205, Training accuracy: 0.8876
Validation loss: 0.4552, Validation accuracy: 0.8458

Current learning rate is 0.100000
Epoch 25, iteration 8800:
Training loss: 0.3131, Training accuracy: 0.8914
Validation loss: 0.4629, Validation accuracy: 0.8510
Saving …

Current learning rate is 0.100000
Epoch 26, iteration 9152:
Training loss: 0.3106, Training accuracy: 0.8928
Validation loss: 0.4233, Validation accuracy: 0.8588
Saving …
```

```
Current learning rate is 0.100000
Epoch 27, iteration 9504:
Training loss: 0.2971, Training accuracy: 0.8968
Validation loss: 0.5892, Validation accuracy: 0.8198

Current learning rate is 0.100000
Epoch 28, iteration 9856:
Training loss: 0.3003, Training accuracy: 0.8945
Validation loss: 0.4929, Validation accuracy: 0.8370

Current learning rate is 0.100000
Epoch 29, iteration 10208:
Training loss: 0.2937, Training accuracy: 0.8985
Validation loss: 0.5049, Validation accuracy: 0.8382

Current learning rate is 0.100000
Epoch 30, iteration 10560:
Training loss: 0.2949, Training accuracy: 0.8964
Validation loss: 0.4326, Validation accuracy: 0.8612
Saving …

Current learning rate is 0.100000
Epoch 31, iteration 10912:
Training loss: 0.2948, Training accuracy: 0.8971
Validation loss: 0.4329, Validation accuracy: 0.8570

Current learning rate is 0.100000
Epoch 32, iteration 11264:
Training loss: 0.2886, Training accuracy: 0.8990
Validation loss: 0.6566, Validation accuracy: 0.8082

Current learning rate is 0.100000
Epoch 33, iteration 11616:
Training loss: 0.2834, Training accuracy: 0.9009
Validation loss: 0.4657, Validation accuracy: 0.8558

Current learning rate is 0.100000
Epoch 34, iteration 11968:
Training loss: 0.2819, Training accuracy: 0.9012
Validation loss: 0.4616, Validation accuracy: 0.8544

Current learning rate is 0.100000
Epoch 35, iteration 12320:
Training loss: 0.2778, Training accuracy: 0.9026
Validation loss: 0.4908, Validation accuracy: 0.8464

Current learning rate is 0.100000
```

```
Epoch 36, iteration 12672:
Training loss: 0.2672, Training accuracy: 0.9059
Validation loss: 0.5385, Validation accuracy: 0.8338

Current learning rate is 0.100000
Epoch 37, iteration 13024:
Training loss: 0.2775, Training accuracy: 0.9024
Validation loss: 0.4091, Validation accuracy: 0.8672
Saving …

Current learning rate is 0.100000
Epoch 38, iteration 13376:
Training loss: 0.2672, Training accuracy: 0.9078
Validation loss: 0.4336, Validation accuracy: 0.8624

Current learning rate is 0.100000
Epoch 39, iteration 13728:
Training loss: 0.2627, Training accuracy: 0.9067
Validation loss: 0.4346, Validation accuracy: 0.8574

Current learning rate is 0.100000
Epoch 40, iteration 14080:
Training loss: 0.2597, Training accuracy: 0.9089
Validation loss: 0.4567, Validation accuracy: 0.8488

Current learning rate is 0.100000
Epoch 41, iteration 14432:
Training loss: 0.2596, Training accuracy: 0.9079
Validation loss: 0.5418, Validation accuracy: 0.8350

Current learning rate is 0.100000
Epoch 42, iteration 14784:
Training loss: 0.2607, Training accuracy: 0.9103
Validation loss: 0.4263, Validation accuracy: 0.8648

Current learning rate is 0.100000
Epoch 43, iteration 15136:
Training loss: 0.2519, Training accuracy: 0.9109
Validation loss: 0.4542, Validation accuracy: 0.8566

Current learning rate is 0.100000
Epoch 44, iteration 15488:
Training loss: 0.2496, Training accuracy: 0.9129
Validation loss: 0.5789, Validation accuracy: 0.8372

Current learning rate is 0.100000
Epoch 45, iteration 15840:
Training loss: 0.2551, Training accuracy: 0.9118
```

```
Validation loss: 0.4412, Validation accuracy: 0.8602

Current learning rate is 0.100000
Epoch 46, iteration 16192:
Training loss: 0.2500, Training accuracy: 0.9134
Validation loss: 0.4825, Validation accuracy: 0.8566

Current learning rate is 0.100000
Epoch 47, iteration 16544:
Training loss: 0.2503, Training accuracy: 0.9123
Validation loss: 0.4366, Validation accuracy: 0.8614

Current learning rate is 0.100000
Epoch 48, iteration 16896:
Training loss: 0.2488, Training accuracy: 0.9136
Validation loss: 0.4049, Validation accuracy: 0.8766
Saving …

Current learning rate is 0.100000
Epoch 49, iteration 17248:
Training loss: 0.2387, Training accuracy: 0.9164
Validation loss: 0.3991, Validation accuracy: 0.8666

Current learning rate is 0.100000
Epoch 50, iteration 17600:
Training loss: 0.2408, Training accuracy: 0.9153
Validation loss: 0.4733, Validation accuracy: 0.8516

Current learning rate is 0.100000
Epoch 51, iteration 17952:
Training loss: 0.2386, Training accuracy: 0.9161
Validation loss: 0.4781, Validation accuracy: 0.8494

Current learning rate is 0.100000
Epoch 52, iteration 18304:
Training loss: 0.2369, Training accuracy: 0.9173
Validation loss: 0.5785, Validation accuracy: 0.8228

Current learning rate is 0.100000
Epoch 53, iteration 18656:
Training loss: 0.2353, Training accuracy: 0.9176
Validation loss: 0.4095, Validation accuracy: 0.8636

Current learning rate is 0.100000
Epoch 54, iteration 19008:
Training loss: 0.2348, Training accuracy: 0.9180
Validation loss: 0.4701, Validation accuracy: 0.8624
```

```
Current learning rate is 0.100000
Epoch 55, iteration 19360:
Training loss: 0.2320, Training accuracy: 0.9192
Validation loss: 0.3708, Validation accuracy: 0.8794
Saving …

Current learning rate is 0.100000
Epoch 56, iteration 19712:
Training loss: 0.2373, Training accuracy: 0.9169
Validation loss: 0.4145, Validation accuracy: 0.8702

Current learning rate is 0.100000
Epoch 57, iteration 20064:
Training loss: 0.2289, Training accuracy: 0.9200
Validation loss: 0.4503, Validation accuracy: 0.8612

Current learning rate is 0.100000
Epoch 58, iteration 20416:
Training loss: 0.2309, Training accuracy: 0.9186
Validation loss: 0.4221, Validation accuracy: 0.8668

Current learning rate is 0.100000
Epoch 59, iteration 20768:
Training loss: 0.2253, Training accuracy: 0.9199
Validation loss: 0.3917, Validation accuracy: 0.8824
Saving …

Current learning rate is 0.100000
Epoch 60, iteration 21120:
Training loss: 0.2286, Training accuracy: 0.9192
Validation loss: 0.4906, Validation accuracy: 0.8536

Current learning rate is 0.100000
Epoch 61, iteration 21472:
Training loss: 0.2233, Training accuracy: 0.9223
Validation loss: 0.4167, Validation accuracy: 0.8696

Current learning rate is 0.100000
Epoch 62, iteration 21824:
Training loss: 0.2292, Training accuracy: 0.9190
Validation loss: 0.3802, Validation accuracy: 0.8742

Current learning rate is 0.100000
Epoch 63, iteration 22176:
Training loss: 0.2252, Training accuracy: 0.9218
Validation loss: 0.4094, Validation accuracy: 0.8662

Current learning rate is 0.100000
```

```
Epoch 64, iteration 22528:
Training loss: 0.2232, Training accuracy: 0.9221
Validation loss: 0.3788, Validation accuracy: 0.8808

Current learning rate is 0.100000
Epoch 65, iteration 22880:
Training loss: 0.2257, Training accuracy: 0.9193
Validation loss: 0.4606, Validation accuracy: 0.8618

Current learning rate is 0.100000
Epoch 66, iteration 23232:
Training loss: 0.2196, Training accuracy: 0.9222
Validation loss: 0.4055, Validation accuracy: 0.8708

Current learning rate is 0.100000
Epoch 67, iteration 23584:
Training loss: 0.2228, Training accuracy: 0.9223
Validation loss: 0.4834, Validation accuracy: 0.8498

Current learning rate is 0.100000
Epoch 68, iteration 23936:
Training loss: 0.2179, Training accuracy: 0.9221
Validation loss: 0.4973, Validation accuracy: 0.8464

Current learning rate is 0.100000
Epoch 69, iteration 24288:
Training loss: 0.2129, Training accuracy: 0.9265
Validation loss: 0.4101, Validation accuracy: 0.8666

Current learning rate is 0.100000
Epoch 70, iteration 24640:
Training loss: 0.2230, Training accuracy: 0.9216
Validation loss: 0.4022, Validation accuracy: 0.8674

Current learning rate is 0.100000
Epoch 71, iteration 24992:
Training loss: 0.2128, Training accuracy: 0.9252
Validation loss: 0.4079, Validation accuracy: 0.8678

Current learning rate is 0.100000
Epoch 72, iteration 25344:
Training loss: 0.2199, Training accuracy: 0.9238
Validation loss: 0.3891, Validation accuracy: 0.8724

Current learning rate is 0.100000
Epoch 73, iteration 25696:
Training loss: 0.2165, Training accuracy: 0.9250
Validation loss: 0.4776, Validation accuracy: 0.8600
```

```
Current learning rate is 0.100000
Epoch 74, iteration 26048:
Training loss: 0.2087, Training accuracy: 0.9266
Validation loss: 0.4351, Validation accuracy: 0.8650

Current learning rate is 0.100000
Epoch 75, iteration 26400:
Training loss: 0.2156, Training accuracy: 0.9234
Validation loss: 0.3953, Validation accuracy: 0.8760

Current learning rate is 0.100000
Epoch 76, iteration 26752:
Training loss: 0.2106, Training accuracy: 0.9254
Validation loss: 0.4357, Validation accuracy: 0.8636

Current learning rate is 0.100000
Epoch 77, iteration 27104:
Training loss: 0.2101, Training accuracy: 0.9257
Validation loss: 0.3806, Validation accuracy: 0.8824

Current learning rate is 0.100000
Epoch 78, iteration 27456:
Training loss: 0.2147, Training accuracy: 0.9242
Validation loss: 0.3566, Validation accuracy: 0.8860
Saving …

Current learning rate is 0.100000
Epoch 79, iteration 27808:
Training loss: 0.2064, Training accuracy: 0.9270
Validation loss: 0.4410, Validation accuracy: 0.8692

Current learning rate is 0.100000
Epoch 80, iteration 28160:
Training loss: 0.2137, Training accuracy: 0.9251
Validation loss: 0.4551, Validation accuracy: 0.8612

Current learning rate is 0.100000
Epoch 81, iteration 28512:
Training loss: 0.2063, Training accuracy: 0.9280
Validation loss: 0.3742, Validation accuracy: 0.8852

Current learning rate is 0.100000
Epoch 82, iteration 28864:
Training loss: 0.2073, Training accuracy: 0.9261
Validation loss: 0.4863, Validation accuracy: 0.8530

Current learning rate is 0.100000
```

```
Epoch 83, iteration 29216:
Training loss: 0.2105, Training accuracy: 0.9262
Validation loss: 0.4233, Validation accuracy: 0.8662

Current learning rate is 0.100000
Epoch 84, iteration 29568:
Training loss: 0.2086, Training accuracy: 0.9265
Validation loss: 0.5409, Validation accuracy: 0.8494

Current learning rate is 0.100000
Epoch 85, iteration 29920:
Training loss: 0.2063, Training accuracy: 0.9283
Validation loss: 0.4195, Validation accuracy: 0.8720

Current learning rate is 0.100000
Epoch 86, iteration 30272:
Training loss: 0.2083, Training accuracy: 0.9280
Validation loss: 0.4213, Validation accuracy: 0.8740

Current learning rate is 0.100000
Epoch 87, iteration 30624:
Training loss: 0.2039, Training accuracy: 0.9287
Validation loss: 0.4776, Validation accuracy: 0.8560

Current learning rate is 0.100000
Epoch 88, iteration 30976:
Training loss: 0.2027, Training accuracy: 0.9280
Validation loss: 0.4101, Validation accuracy: 0.8642

Current learning rate is 0.100000
Epoch 89, iteration 31328:
Training loss: 0.2036, Training accuracy: 0.9279
Validation loss: 0.3694, Validation accuracy: 0.8840

Current learning rate is 0.100000
Epoch 90, iteration 31680:
Training loss: 0.2117, Training accuracy: 0.9248
Validation loss: 0.4063, Validation accuracy: 0.8722

Current learning rate is 0.100000
Epoch 91, iteration 32032:
Training loss: 0.2066, Training accuracy: 0.9287
Validation loss: 0.4331, Validation accuracy: 0.8708

Current learning rate is 0.100000
Epoch 92, iteration 32384:
Training loss: 0.2039, Training accuracy: 0.9279
Validation loss: 0.3904, Validation accuracy: 0.8714
```

```
Current learning rate is 0.100000
Epoch 93, iteration 32736:
Training loss: 0.2016, Training accuracy: 0.9287
Validation loss: 0.4854, Validation accuracy: 0.8492

Current learning rate is 0.100000
Epoch 94, iteration 33088:
Training loss: 0.2007, Training accuracy: 0.9298
Validation loss: 0.3460, Validation accuracy: 0.8874
Saving …

Current learning rate is 0.100000
Epoch 95, iteration 33440:
Training loss: 0.1992, Training accuracy: 0.9312
Validation loss: 0.4527, Validation accuracy: 0.8542

Current learning rate is 0.100000
Epoch 96, iteration 33792:
Training loss: 0.2019, Training accuracy: 0.9288
Validation loss: 0.4418, Validation accuracy: 0.8680

Current learning rate is 0.100000
Epoch 97, iteration 34144:
Training loss: 0.1990, Training accuracy: 0.9310
Validation loss: 0.4205, Validation accuracy: 0.8710

Current learning rate is 0.100000
Epoch 98, iteration 34496:
Training loss: 0.2033, Training accuracy: 0.9297
Validation loss: 0.5346, Validation accuracy: 0.8458

Current learning rate is 0.100000
Epoch 99, iteration 34848:
Training loss: 0.1949, Training accuracy: 0.9316
Validation loss: 0.3870, Validation accuracy: 0.8798

Current learning rate has decayed to 0.010000
Epoch 100, iteration 35200:
Training loss: 0.1179, Training accuracy: 0.9612
Validation loss: 0.2675, Validation accuracy: 0.9126
Saving …

Current learning rate is 0.010000
Epoch 101, iteration 35552:
Training loss: 0.0881, Training accuracy: 0.9712
Validation loss: 0.2584, Validation accuracy: 0.9194
Saving …
```

```
Current learning rate is 0.010000
Epoch 102, iteration 35904:
Training loss: 0.0832, Training accuracy: 0.9722
Validation loss: 0.2571, Validation accuracy: 0.9196
Saving …

Current learning rate is 0.010000
Epoch 103, iteration 36256:
Training loss: 0.0737, Training accuracy: 0.9754
Validation loss: 0.2595, Validation accuracy: 0.9194

Current learning rate is 0.010000
Epoch 104, iteration 36608:
Training loss: 0.0675, Training accuracy: 0.9772
Validation loss: 0.2608, Validation accuracy: 0.9198
Saving …

Current learning rate is 0.010000
Epoch 105, iteration 36960:
Training loss: 0.0649, Training accuracy: 0.9783
Validation loss: 0.2576, Validation accuracy: 0.9214
Saving …

Current learning rate is 0.010000
Epoch 106, iteration 37312:
Training loss: 0.0615, Training accuracy: 0.9800
Validation loss: 0.2634, Validation accuracy: 0.9210

Current learning rate is 0.010000
Epoch 107, iteration 37664:
Training loss: 0.0587, Training accuracy: 0.9808
Validation loss: 0.2650, Validation accuracy: 0.9198

Current learning rate is 0.010000
Epoch 108, iteration 38016:
Training loss: 0.0541, Training accuracy: 0.9828
Validation loss: 0.2631, Validation accuracy: 0.9242
Saving …

Current learning rate is 0.010000
Epoch 109, iteration 38368:
Training loss: 0.0521, Training accuracy: 0.9828
Validation loss: 0.2634, Validation accuracy: 0.9246
Saving …

Current learning rate is 0.010000
Epoch 110, iteration 38720:
```

```
Training loss: 0.0508, Training accuracy: 0.9836
Validation loss: 0.2702, Validation accuracy: 0.9214

Current learning rate is 0.010000
Epoch 111, iteration 39072:
Training loss: 0.0489, Training accuracy: 0.9842
Validation loss: 0.2731, Validation accuracy: 0.9208

Current learning rate is 0.010000
Epoch 112, iteration 39424:
Training loss: 0.0480, Training accuracy: 0.9847
Validation loss: 0.2804, Validation accuracy: 0.9204

Current learning rate is 0.010000
Epoch 113, iteration 39776:
Training loss: 0.0461, Training accuracy: 0.9849
Validation loss: 0.2775, Validation accuracy: 0.9226

Current learning rate is 0.010000
Epoch 114, iteration 40128:
Training loss: 0.0448, Training accuracy: 0.9856
Validation loss: 0.2796, Validation accuracy: 0.9222

Current learning rate is 0.010000
Epoch 115, iteration 40480:
Training loss: 0.0435, Training accuracy: 0.9859
Validation loss: 0.2835, Validation accuracy: 0.9222

Current learning rate is 0.010000
Epoch 116, iteration 40832:
Training loss: 0.0403, Training accuracy: 0.9870
Validation loss: 0.2848, Validation accuracy: 0.9218

Current learning rate is 0.010000
Epoch 117, iteration 41184:
Training loss: 0.0417, Training accuracy: 0.9862
Validation loss: 0.2824, Validation accuracy: 0.9224

Current learning rate is 0.010000
Epoch 118, iteration 41536:
Training loss: 0.0397, Training accuracy: 0.9871
Validation loss: 0.2835, Validation accuracy: 0.9226

Current learning rate is 0.010000
Epoch 119, iteration 41888:
Training loss: 0.0366, Training accuracy: 0.9884
Validation loss: 0.2846, Validation accuracy: 0.9214
```

```
Current learning rate is 0.010000
Epoch 120, iteration 42240:
Training loss: 0.0369, Training accuracy: 0.9880
Validation loss: 0.2934, Validation accuracy: 0.9202

Current learning rate is 0.010000
Epoch 121, iteration 42592:
Training loss: 0.0364, Training accuracy: 0.9883
Validation loss: 0.2893, Validation accuracy: 0.9238

Current learning rate is 0.010000
Epoch 122, iteration 42944:
Training loss: 0.0357, Training accuracy: 0.9884
Validation loss: 0.3048, Validation accuracy: 0.9206

Current learning rate is 0.010000
Epoch 123, iteration 43296:
Training loss: 0.0345, Training accuracy: 0.9886
Validation loss: 0.3016, Validation accuracy: 0.9194

Current learning rate is 0.010000
Epoch 124, iteration 43648:
Training loss: 0.0331, Training accuracy: 0.9894
Validation loss: 0.3049, Validation accuracy: 0.9214

Current learning rate is 0.010000
Epoch 125, iteration 44000:
Training loss: 0.0302, Training accuracy: 0.9905
Validation loss: 0.3049, Validation accuracy: 0.9206

Current learning rate is 0.010000
Epoch 126, iteration 44352:
Training loss: 0.0325, Training accuracy: 0.9900
Validation loss: 0.3062, Validation accuracy: 0.9206

Current learning rate is 0.010000
Epoch 127, iteration 44704:
Training loss: 0.0311, Training accuracy: 0.9906
Validation loss: 0.3127, Validation accuracy: 0.9216

Current learning rate is 0.010000
Epoch 128, iteration 45056:
Training loss: 0.0308, Training accuracy: 0.9902
Validation loss: 0.3168, Validation accuracy: 0.9182

Current learning rate is 0.010000
Epoch 129, iteration 45408:
Training loss: 0.0301, Training accuracy: 0.9905
```

```
Validation loss: 0.2979, Validation accuracy: 0.9202

Current learning rate is 0.010000
Epoch 130, iteration 45760:
Training loss: 0.0303, Training accuracy: 0.9903
Validation loss: 0.3077, Validation accuracy: 0.9224

Current learning rate is 0.010000
Epoch 131, iteration 46112:
Training loss: 0.0282, Training accuracy: 0.9911
Validation loss: 0.3087, Validation accuracy: 0.9234

Current learning rate is 0.010000
Epoch 132, iteration 46464:
Training loss: 0.0270, Training accuracy: 0.9916
Validation loss: 0.3152, Validation accuracy: 0.9232

Current learning rate is 0.010000
Epoch 133, iteration 46816:
Training loss: 0.0270, Training accuracy: 0.9917
Validation loss: 0.3243, Validation accuracy: 0.9198

Current learning rate is 0.010000
Epoch 134, iteration 47168:
Training loss: 0.0261, Training accuracy: 0.9916
Validation loss: 0.3148, Validation accuracy: 0.9218

Current learning rate is 0.010000
Epoch 135, iteration 47520:
Training loss: 0.0262, Training accuracy: 0.9922
Validation loss: 0.3246, Validation accuracy: 0.9196

Current learning rate is 0.010000
Epoch 136, iteration 47872:
Training loss: 0.0261, Training accuracy: 0.9919
Validation loss: 0.3225, Validation accuracy: 0.9192

Current learning rate is 0.010000
Epoch 137, iteration 48224:
Training loss: 0.0258, Training accuracy: 0.9920
Validation loss: 0.3324, Validation accuracy: 0.9186

Current learning rate is 0.010000
Epoch 138, iteration 48576:
Training loss: 0.0251, Training accuracy: 0.9919
Validation loss: 0.3259, Validation accuracy: 0.9206

Current learning rate is 0.010000
```

```
Epoch 139, iteration 48928:
Training loss: 0.0228, Training accuracy: 0.9931
Validation loss: 0.3299, Validation accuracy: 0.9162

Current learning rate is 0.010000
Epoch 140, iteration 49280:
Training loss: 0.0250, Training accuracy: 0.9920
Validation loss: 0.3190, Validation accuracy: 0.9198

Current learning rate is 0.010000
Epoch 141, iteration 49632:
Training loss: 0.0244, Training accuracy: 0.9926
Validation loss: 0.3194, Validation accuracy: 0.9230

Current learning rate is 0.010000
Epoch 142, iteration 49984:
Training loss: 0.0229, Training accuracy: 0.9931
Validation loss: 0.3348, Validation accuracy: 0.9188

Current learning rate is 0.010000
Epoch 143, iteration 50336:
Training loss: 0.0230, Training accuracy: 0.9930
Validation loss: 0.3246, Validation accuracy: 0.9204

Current learning rate is 0.010000
Epoch 144, iteration 50688:
Training loss: 0.0243, Training accuracy: 0.9924
Validation loss: 0.3263, Validation accuracy: 0.9188

Current learning rate is 0.010000
Epoch 145, iteration 51040:
Training loss: 0.0227, Training accuracy: 0.9930
Validation loss: 0.3307, Validation accuracy: 0.9244

Current learning rate is 0.010000
Epoch 146, iteration 51392:
Training loss: 0.0230, Training accuracy: 0.9927
Validation loss: 0.3273, Validation accuracy: 0.9220

Current learning rate is 0.010000
Epoch 147, iteration 51744:
Training loss: 0.0197, Training accuracy: 0.9945
Validation loss: 0.3294, Validation accuracy: 0.9212

Current learning rate is 0.010000
Epoch 148, iteration 52096:
Training loss: 0.0220, Training accuracy: 0.9935
Validation loss: 0.3320, Validation accuracy: 0.9206
```

```
Current learning rate is 0.010000
Epoch 149, iteration 52448:
Training loss: 0.0211, Training accuracy: 0.9936
Validation loss: 0.3333, Validation accuracy: 0.9228

Current learning rate has decayed to 0.001000
Epoch 150, iteration 52800:
Training loss: 0.0177, Training accuracy: 0.9950
Validation loss: 0.3251, Validation accuracy: 0.9240

Current learning rate is 0.001000
Epoch 151, iteration 53152:
Training loss: 0.0180, Training accuracy: 0.9947
Validation loss: 0.3276, Validation accuracy: 0.9240

Current learning rate is 0.001000
Epoch 152, iteration 53504:
Training loss: 0.0165, Training accuracy: 0.9958
Validation loss: 0.3291, Validation accuracy: 0.9238

Current learning rate is 0.001000
Epoch 153, iteration 53856:
Training loss: 0.0159, Training accuracy: 0.9956
Validation loss: 0.3281, Validation accuracy: 0.9234

Current learning rate is 0.001000
Epoch 154, iteration 54208:
Training loss: 0.0161, Training accuracy: 0.9956
Validation loss: 0.3278, Validation accuracy: 0.9216

Current learning rate is 0.001000
Epoch 155, iteration 54560:
Training loss: 0.0155, Training accuracy: 0.9958
Validation loss: 0.3247, Validation accuracy: 0.9232

Current learning rate is 0.001000
Epoch 156, iteration 54912:
Training loss: 0.0151, Training accuracy: 0.9959
Validation loss: 0.3251, Validation accuracy: 0.9230

Current learning rate is 0.001000
Epoch 157, iteration 55264:
Training loss: 0.0151, Training accuracy: 0.9957
Validation loss: 0.3280, Validation accuracy: 0.9230

Current learning rate is 0.001000
Epoch 158, iteration 55616:
```

```
Training loss: 0.0148, Training accuracy: 0.9961
Validation loss: 0.3258, Validation accuracy: 0.9232

Current learning rate is 0.001000
Epoch 159, iteration 55968:
Training loss: 0.0146, Training accuracy: 0.9960
Validation loss: 0.3260, Validation accuracy: 0.9242

Current learning rate is 0.001000
Epoch 160, iteration 56320:
Training loss: 0.0151, Training accuracy: 0.9960
Validation loss: 0.3232, Validation accuracy: 0.9246

Current learning rate is 0.001000
Epoch 161, iteration 56672:
Training loss: 0.0138, Training accuracy: 0.9965
Validation loss: 0.3270, Validation accuracy: 0.9248
Saving …

Current learning rate is 0.001000
Epoch 162, iteration 57024:
Training loss: 0.0151, Training accuracy: 0.9960
Validation loss: 0.3233, Validation accuracy: 0.9242

Current learning rate is 0.001000
Epoch 163, iteration 57376:
Training loss: 0.0141, Training accuracy: 0.9964
Validation loss: 0.3228, Validation accuracy: 0.9248

Current learning rate is 0.001000
Epoch 164, iteration 57728:
Training loss: 0.0140, Training accuracy: 0.9963
Validation loss: 0.3243, Validation accuracy: 0.9232

Current learning rate is 0.001000
Epoch 165, iteration 58080:
Training loss: 0.0138, Training accuracy: 0.9964
Validation loss: 0.3230, Validation accuracy: 0.9242

Current learning rate is 0.001000
Epoch 166, iteration 58432:
Training loss: 0.0140, Training accuracy: 0.9969
Validation loss: 0.3270, Validation accuracy: 0.9234

Current learning rate is 0.001000
Epoch 167, iteration 58784:
Training loss: 0.0140, Training accuracy: 0.9961
Validation loss: 0.3245, Validation accuracy: 0.9236
```

```
Current learning rate is 0.001000
Epoch 168, iteration 59136:
Training loss: 0.0136, Training accuracy: 0.9966
Validation loss: 0.3261, Validation accuracy: 0.9234

Current learning rate is 0.001000
Epoch 169, iteration 59488:
Training loss: 0.0137, Training accuracy: 0.9964
Validation loss: 0.3295, Validation accuracy: 0.9234

Current learning rate is 0.001000
Epoch 170, iteration 59840:
Training loss: 0.0143, Training accuracy: 0.9960
Validation loss: 0.3267, Validation accuracy: 0.9228

Current learning rate is 0.001000
Epoch 171, iteration 60192:
Training loss: 0.0137, Training accuracy: 0.9964
Validation loss: 0.3258, Validation accuracy: 0.9246

Current learning rate is 0.001000
Epoch 172, iteration 60544:
Training loss: 0.0136, Training accuracy: 0.9965
Validation loss: 0.3266, Validation accuracy: 0.9262
Saving …

Current learning rate is 0.001000
Epoch 173, iteration 60896:
Training loss: 0.0139, Training accuracy: 0.9966
Validation loss: 0.3246, Validation accuracy: 0.9248

Current learning rate is 0.001000
Epoch 174, iteration 61248:
Training loss: 0.0142, Training accuracy: 0.9963
Validation loss: 0.3244, Validation accuracy: 0.9232

Current learning rate is 0.001000
Epoch 175, iteration 61600:
Training loss: 0.0139, Training accuracy: 0.9963
Validation loss: 0.3284, Validation accuracy: 0.9244

Current learning rate is 0.001000
Epoch 176, iteration 61952:
Training loss: 0.0130, Training accuracy: 0.9967
Validation loss: 0.3248, Validation accuracy: 0.9236

Current learning rate is 0.001000
```

```
Epoch 177, iteration 62304:
Training loss: 0.0134, Training accuracy: 0.9966
Validation loss: 0.3245, Validation accuracy: 0.9244

Current learning rate is 0.001000
Epoch 178, iteration 62656:
Training loss: 0.0125, Training accuracy: 0.9969
Validation loss: 0.3213, Validation accuracy: 0.9256

Current learning rate is 0.001000
Epoch 179, iteration 63008:
Training loss: 0.0131, Training accuracy: 0.9968
Validation loss: 0.3227, Validation accuracy: 0.9232

Current learning rate is 0.001000
Epoch 180, iteration 63360:
Training loss: 0.0145, Training accuracy: 0.9962
Validation loss: 0.3243, Validation accuracy: 0.9234

Current learning rate is 0.001000
Epoch 181, iteration 63712:
Training loss: 0.0125, Training accuracy: 0.9971
Validation loss: 0.3239, Validation accuracy: 0.9238

Current learning rate is 0.001000
According to ResNet paper, should terminate
Epoch 182, iteration 64064:
Training loss: 0.0132, Training accuracy: 0.9967
Validation loss: 0.3255, Validation accuracy: 0.9244

Current learning rate is 0.001000
According to ResNet paper, should terminate
Epoch 183, iteration 64416:
Training loss: 0.0135, Training accuracy: 0.9967
Validation loss: 0.3251, Validation accuracy: 0.9232

Current learning rate is 0.001000
According to ResNet paper, should terminate
Epoch 184, iteration 64768:
Training loss: 0.0129, Training accuracy: 0.9970
Validation loss: 0.3238, Validation accuracy: 0.9256

Current learning rate is 0.001000
According to ResNet paper, should terminate
Epoch 185, iteration 65120:
Training loss: 0.0134, Training accuracy: 0.9967
Validation loss: 0.3285, Validation accuracy: 0.9242
```

```
Current learning rate is 0.001000
According to ResNet paper, should terminate
Epoch 186, iteration 65472:
Training loss: 0.0128, Training accuracy: 0.9967
Validation loss: 0.3256, Validation accuracy: 0.9236

Current learning rate is 0.001000
According to ResNet paper, should terminate
Epoch 187, iteration 65824:
Training loss: 0.0124, Training accuracy: 0.9968
Validation loss: 0.3255, Validation accuracy: 0.9242

Current learning rate is 0.001000
According to ResNet paper, should terminate
Epoch 188, iteration 66176:
Training loss: 0.0129, Training accuracy: 0.9966
Validation loss: 0.3293, Validation accuracy: 0.9238

Current learning rate is 0.001000
According to ResNet paper, should terminate
Epoch 189, iteration 66528:
Training loss: 0.0125, Training accuracy: 0.9968
Validation loss: 0.3251, Validation accuracy: 0.9240

Current learning rate is 0.001000
According to ResNet paper, should terminate
Epoch 190, iteration 66880:
Training loss: 0.0123, Training accuracy: 0.9974
Validation loss: 0.3264, Validation accuracy: 0.9238

Current learning rate is 0.001000
According to ResNet paper, should terminate
Epoch 191, iteration 67232:
Training loss: 0.0129, Training accuracy: 0.9970
Validation loss: 0.3271, Validation accuracy: 0.9230

Current learning rate is 0.001000
According to ResNet paper, should terminate
Epoch 192, iteration 67584:
Training loss: 0.0124, Training accuracy: 0.9970
Validation loss: 0.3276, Validation accuracy: 0.9250

Current learning rate is 0.001000
According to ResNet paper, should terminate
Epoch 193, iteration 67936:
Training loss: 0.0128, Training accuracy: 0.9968
Validation loss: 0.3283, Validation accuracy: 0.9242
```

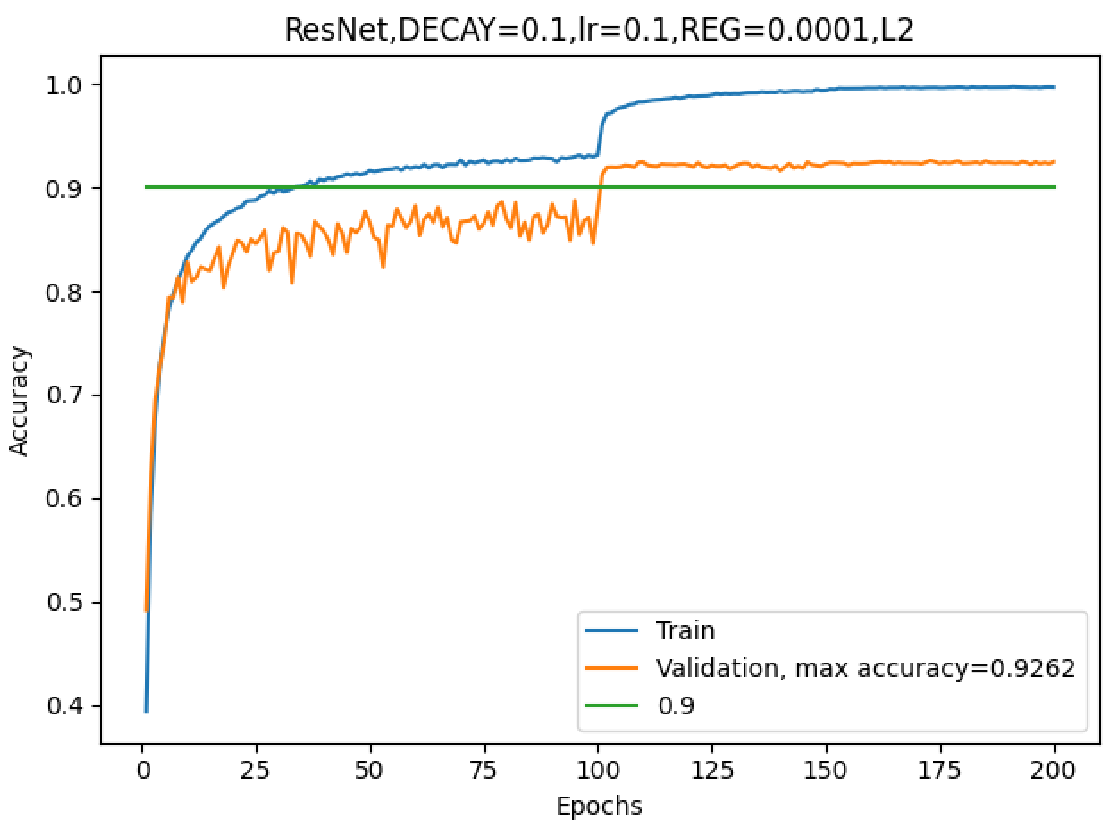
```
Current learning rate is 0.001000
According to ResNet paper, should terminate
Epoch 194, iteration 68288:
Training loss: 0.0135, Training accuracy: 0.9966
Validation loss: 0.3266, Validation accuracy: 0.9224

Current learning rate is 0.001000
According to ResNet paper, should terminate
Epoch 195, iteration 68640:
Training loss: 0.0129, Training accuracy: 0.9968
Validation loss: 0.3264, Validation accuracy: 0.9246

Current learning rate is 0.001000
According to ResNet paper, should terminate
Epoch 196, iteration 68992:
Training loss: 0.0133, Training accuracy: 0.9965
Validation loss: 0.3283, Validation accuracy: 0.9230

Current learning rate is 0.001000
According to ResNet paper, should terminate
Epoch 197, iteration 69344:
Training loss: 0.0124, Training accuracy: 0.9970
Validation loss: 0.3274, Validation accuracy: 0.9240

Current learning rate is 0.001000
According to ResNet paper, should terminate
Epoch 198, iteration 69696:
Training loss: 0.0123, Training accuracy: 0.9971
Validation loss: 0.3286, Validation accuracy: 0.9230

Current learning rate is 0.001000
According to ResNet paper, should terminate
Epoch 199, iteration 70048:
Training loss: 0.0121, Training accuracy: 0.9969
Validation loss: 0.3340, Validation accuracy: 0.9248

==================================================
==> Optimization finished! Best validation accuracy: 0.9262
```

2 Bonus: with learning rate decay

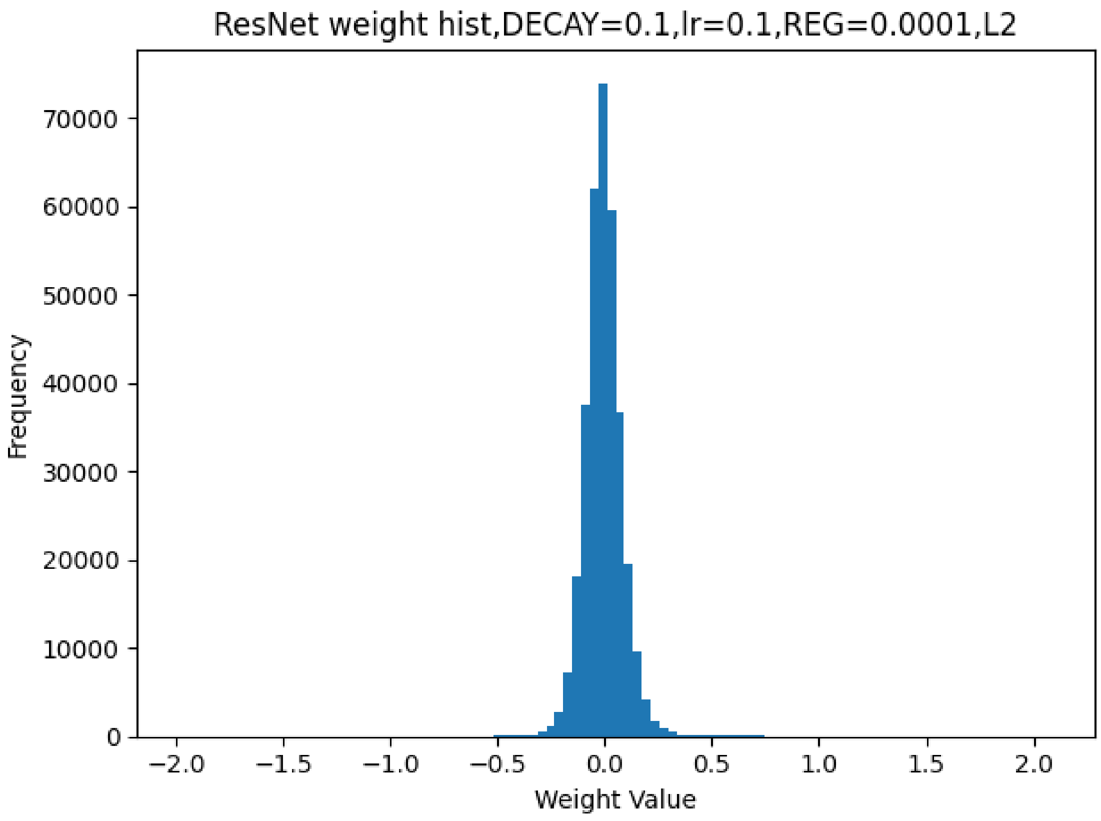
The following code can help you adjust the learning rate during training. You need to figure out how to incorporate this code into your training loop.

```
    if i % DECAY_EPOCHS == 0 and i != 0:
        current_learning_rate = current_learning_rate * DECAY
        for param_group in optimizer.param_groups:
            param_group['lr'] = current_learning_rate
        print("Current learning rate has decayed to %f" %current_learning_rate)
```

[21]:
```
##### plotting accuracy vs epoch


import matplotlib.pyplot as plt
import numpy as np


def my_to_np(lst):
    return np.array([lst[i].detach().cpu().numpy().flatten() for i in␣
 ↪range(len(lst))])


save = 'y'

fig, ax = plt.subplots(1, 1)
xx = np.linspace(1, EPOCHS, EPOCHS)
## don't call my_to_np(lst) if plotting raises an error
ax.plot(xx, my_to_np(accuracy_arr_train), label='Train')
ax.plot(xx, my_to_np(accuracy_arr_val), label='Validation, max accuracy={:.4f}'.
 ↪format(best_val_acc))
ax.plot(xx, np.linspace(0.9, 0.9, EPOCHS), label='0.9')
title_ = 'ResNet,DECAY={:g},lr={:g},REG={:g},{:s}'.format(DECAY, INITIAL_LR,␣
 ↪REG, L1_or_L2)
if DECAY == 1:
    title_ = 'No Learning Rate Decay'
ax.set_xlabel('Epochs')
ax.set_ylabel('Accuracy')
ax.legend()
ax.set_title(title_)
fig.tight_layout()
if save == 'y':
    plt.savefig('q3_aug_BN_ReLU_lr{:g}_{:s}_{:g}.pdf'.format(INITIAL_LR,␣
 ↪L1_or_L2, REG),
                dpi=500, bbox_inches='tight')
```



[23]:
```
##### plotting weights distribution

save = 'y'
fig, ax = plt.subplots(1, 1)

weights_list = []

for name, module in model.named_modules():
    if isinstance(module, nn.Conv2d) or isinstance(module, nn.Linear):
        weights_list.extend(list((module.weight.cpu().detach().numpy()).
 ↪flatten()))

ax.hist(weights_list, bins=100)
ax.set_title(str("ResNet weight hist,DECAY={:g},lr={:g},REG={:g},{:s}".
 ↪format(DECAY, INITIAL_LR, REG, L1_or_L2)))

ax.set_xlabel('Weight Value')
ax.set_ylabel('Frequency')
fig.tight_layout()
if save == 'y':
    plt.savefig('q3weights_aug_BN_ReLU_lr{:g}_{:s}_{:g}.pdf'.format(INITIAL_LR,␣
↪L1_or_L2, REG),
                dpi=500, bbox_inches='tight')
```



[ ]: