assignment4

May 28, 2024

1 Assignment 4: Self-Attention for Vision

For this assignment, we're going to implement self-attention blocks in a convolutional neural network for CIFAR-10 Classification.

2 Part I. Preparation

First, we load the CIFAR-10 dataset. This might take a couple minutes the first time you do it, but the files should stay cached after that.

[41]:

```python
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader
from torch.utils.data import sampler

import torchvision.datasets as dset
import torchvision.transforms as T

import numpy as np
```

[42]:

```python
NUM_TRAIN = 49000

# The torchvision.transforms package provides tools for preprocessing data
# and for performing data augmentation; here we set up a transform to
# preprocess the data by subtracting the mean RGB value and dividing by the
# standard deviation of each RGB value; we've hardcoded the mean and std.
transform = T.Compose([
                T.ToTensor(),
                T.Normalize((0.4914, 0.4822, 0.4465), (0.2023, 0.1994, 0.2010))
            ])

# We set up a Dataset object for each split (train / val / test); Datasets load
# training examples one at a time, so we wrap each Dataset in a DataLoader which
# iterates through the Dataset and forms minibatches. We divide the CIFAR-10
# training set into train and val sets by passing a Sampler object to the
# DataLoader telling how it should sample from the underlying Dataset.
cifar10_train = dset.CIFAR10('./data/datasets', train=True, download=True,
                             transform=transform)
loader_train = DataLoader(cifar10_train, batch_size=64,
                          sampler=sampler.SubsetRandomSampler(range(NUM_TRAIN)))

cifar10_val = dset.CIFAR10('./data/datasets', train=True, download=True,
                           transform=transform)
loader_val = DataLoader(cifar10_val, batch_size=64,
                        sampler=sampler.SubsetRandomSampler(range(NUM_TRAIN,␣
 ↪50000)))

cifar10_test = dset.CIFAR10('./data/datasets', train=False, download=True,
                            transform=transform)
loader_test = DataLoader(cifar10_test, batch_size=64)
```

```
Files already downloaded and verified
Files already downloaded and verified
Files already downloaded and verified
```

You have an option to **use GPU by setting the flag to True below**. It is not necessary to use GPU for this assignment. Note that if your computer does not have CUDA enabled, `torch.cuda.is_available()` will return False and this notebook will fallback to CPU mode.

The global variables `dtype` and `device` will control the data types throughout this assignment.

[43]:

```
USE_GPU = True

dtype = torch.float32 # we will be using float throughout this tutorial

if USE_GPU and torch.cuda.is_available():
    device = torch.device('cuda')
else:
    device = torch.device('cpu')

# Constant to control how frequently we print train loss
print_every = 100

print('using device:', device)
```

```
using device: cuda
```

2.1 Flatten Function

[44]:

```
def flatten(x):
    N = x.shape[0] # read in N, C, H, W
    return x.view(N, -1)  # "flatten" the C * H * W values into a single vector␣
 ↪per image

def test_flatten():
    x = torch.arange(12).view(2, 1, 3, 2)
    print('Before flattening: ', x)
    print('After flattening: ', flatten(x))

test_flatten()
```

```
Before flattening:  tensor([[[[ 0,  1],
          [ 2,  3],
          [ 4,  5]]],


        [[[ 6,  7],
          [ 8,  9],
          [10, 11]]]])
After flattening:  tensor([[ 0,  1,  2,  3,  4,  5],
        [ 6,  7,  8,  9, 10, 11]])
```

2.1.1 Check Accuracy Function

[45]:

```
import torch.nn.functional as F  # useful stateless functions
def check_accuracy(loader, model):
    if loader.dataset.train:
        print('Checking accuracy on validation set')
    else:
        print('Checking accuracy on test set')
    num_correct = 0
    num_samples = 0
    model.eval()  # set model to evaluation mode
    with torch.no_grad():
        for x, y in loader:
            x = x.to(device=device, dtype=dtype)  # move to device, e.g. GPU
            y = y.to(device=device, dtype=torch.long)
            scores = model(x)
            _, preds = scores.max(1)
            num_correct += (preds == y).sum()
            num_samples += preds.size(0)
        acc = float(num_correct) / num_samples
        print('Got %d / %d correct (%.2f)' % (num_correct, num_samples, 100 *␣
 ↪acc))
        return 100 * acc
```

2.1.2 Training Loop

[46]:

```
def train(model, optimizer, epochs=1):
    """
    Train a model on CIFAR-10 using the PyTorch Module API.
    Inputs:
    - model: A PyTorch Module giving the model to train.
    - optimizer: An Optimizer object we will use to train the model
    - epochs: (Optional) A Python integer giving the number of epochs to train␣
 ↪for

    Returns: Nothing, but prints model accuracies during training.
    """
    model = model.to(device=device)  # move the model parameters to CPU/GPU
    acc_max = 0
    for e in range(epochs):
        for t, (x, y) in enumerate(loader_train):

            model.train()  # put model to training mode
            x = x.to(device=device, dtype=dtype)  # move to device, e.g. GPU
            y = y.to(device=device, dtype=torch.long)

            scores = model(x)
            loss = F.cross_entropy(scores, y)

            # Zero out all of the gradients for the variables which the␣
 ↪optimizer
            # will update.
            optimizer.zero_grad()

            # This is the backwards pass: compute the gradient of the loss with
            # respect to each  parameter of the model.
            loss.backward()

            # Actually update the parameters of the model using the gradients
            # computed by the backwards pass.
            optimizer.step()

            if t % print_every == 0:
                print('Epoch %d, Iteration %d, loss = %.4f' % (e, t, loss.
 ↪item()))
                acc = check_accuracy(loader_val, model)
                if acc >= acc_max:
                    acc_max = acc
                print()
    print("Maximum accuracy attained: ", acc_max)
```

[47]:
```
# We need to wrap `flatten` function in a module in order to stack it
# in nn.Sequential
class Flatten(nn.Module):
    def forward(self, x):
        return flatten(x)
```

2.2 Vanilla CNN; No Attention

We implement the vanilla architecture for you here. Do not modify the architecture. You will use the same architecture in the following parts. Do not modify the hyper-parameters.

[94]:

```
channel_1 = 64
channel_2 = 32
learning_rate = 1e-3
num_classes = 10

model = nn.Sequential(
    nn.Conv2d(3, channel_1, 3, padding=1, stride=1),
    nn.ReLU(),
    nn.Conv2d(channel_1, channel_2, 3, padding=1),
    nn.ReLU(),
    Flatten(),
    nn.Linear(channel_2*32*32, num_classes),
)

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


train(model, optimizer, epochs=1)
```

```
Epoch 0, Iteration 0, loss = 2.2902
Checking accuracy on validation set
Got 119 / 1000 correct (11.90)

Epoch 0, Iteration 100, loss = 1.5537
Checking accuracy on validation set
Got 451 / 1000 correct (45.10)

Epoch 0, Iteration 200, loss = 1.5256
Checking accuracy on validation set
Got 473 / 1000 correct (47.30)

Epoch 0, Iteration 300, loss = 1.3774
Checking accuracy on validation set
Got 488 / 1000 correct (48.80)

Epoch 0, Iteration 400, loss = 1.0604
Checking accuracy on validation set
Got 535 / 1000 correct (53.50)

Epoch 0, Iteration 500, loss = 1.3256
Checking accuracy on validation set
Got 528 / 1000 correct (52.80)
```

```
Epoch 0, Iteration 600, loss = 1.2408
Checking accuracy on validation set
Got 570 / 1000 correct (57.00)

Epoch 0, Iteration 700, loss = 1.1947
Checking accuracy on validation set
Got 601 / 1000 correct (60.10)

Maximum accuracy attained:  60.099999999999994
```

2.3 Test set – run this only once

Now we test our model on the test set . Think about how this compares to your validation set accuracy. You should be able to see atleast 55% accuracy

[95]:

```
vanillaModel = model
check_accuracy(loader_test, vanillaModel)
```

```
Checking accuracy on test set
Got 5952 / 10000 correct (59.52)
```

[95]:

```
59.519999999999996
```

2.4 Part II Self-Attention

In the next section, you will implement an Attention layer which you will then use within a convnet architecture defined above for cifar 10 classification task.

A self-attention layer is formulated as following:

Input: X of shape $(H \times W, C)$

Query, key, value linear transforms are W_Q, W_K, W_V, of shape (C, C). We implement these linear transforms as 1x1 convolutional layers of the same dimensions.

XW_Q, XW_K, XW_V, represent the output volumes when input X is passed through the transforms.

Self-Attention is given by the formula: $Attention(X) = X + Softmax(\frac{XW_Q(XW_K)^\top}{\sqrt{C}})XW_V$

2.4.1 Inline Question 1: Self-Attention is equivalent to which of the following: (5 points)

1. K-means clustering
2. Non-local means
3. Residual Block
4. Gaussian Blurring

Your Answer: 2

2.4.2 Here you implement the Attention module, and run it in the next section (40 points)

[84]:

```
# Initialize the attention module as a nn.Module subclass
class Attention(nn.Module):
    def __init__(self, in_channels):
        super().__init__()

        # TODO: Implement the Key, Query and Value linear transforms as 1x1␣
 ↪convolutional layers
        # Hint: channel size remains constant throughout
        self.conv_query = nn.Conv2d(in_channels, in_channels, 1)
        self.conv_key = nn.Conv2d(in_channels, in_channels, 1)
        self.conv_value = nn.Conv2d(in_channels, in_channels, 1)

    def forward(self, x):
        N, C, H, W = x.shape

        # TODO: Pass the input through conv_query, reshape the output volume to␣
 ↪(N, C, H*W)
        q = self.conv_query(x)
        q = torch.reshape(q, (N, C, -1))
        # TODO: Pass the input through conv_key, reshape the output volume to␣
 ↪(N, C, H*W)
        k = self.conv_key(x)
        k = torch.reshape(k, (N, C, -1))
        # TODO: Pass the input through conv_value, reshape the output volume to␣
 ↪(N, C, H*W)
        v = self.conv_value(x)
        v = torch.reshape(v, (N, C, -1))
        # TODO: Implement the above formula for attention using q, k, v, C
        # NOTE: The X in the formula is already added for you in the return line
        q = torch.transpose(q, 1, 2)
        att_distrib = torch.matmul(q, k)
        att_distrib = att_distrib / torch.sqrt(torch.tensor([C], device=device))
        att_distrib = torch.softmax(att_distrib, dim=1)
        # print(att_distrib.shape, torch.Tensor([C]))
        attention = torch.matmul(att_distrib, torch.transpose(v, 1, 2)) # (N,␣
 ↪H*W, C)
        attention = torch.transpose(attention, 1, 2) # (N, C, H*W)
        # Reshape the output to (N, C, H, W) before adding to the input volume
        attention = attention.reshape(N, C, H, W)
        return x + attention
```

2.5 Single Attention Block: Early attention; After the first conv layer. (10 points)

[85]:

```
channel_1 = 64
channel_2 = 32
learning_rate = 1e-3

# TODO: Use the above Attention module after the first Convolutional layer.
# Essentially the architecture should be␣
 ↪[Conv->Relu->Attention->Relu->Conv->Relu->Linear]

model = nn.Sequential(
    nn.Conv2d(3, channel_1, 3, padding=1, stride=1),
    nn.ReLU(),
    Attention(channel_1),
    nn.ReLU(),
    nn.Conv2d(channel_1, channel_2, 3, padding=1, stride=1),
    nn.ReLU(),
    Flatten(),
    nn.Linear(channel_2*32*32, num_classes),
)


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

train(model, optimizer, epochs=10)
```

```
Epoch 0, Iteration 0, loss = 2.2777
Checking accuracy on validation set
Got 145 / 1000 correct (14.50)

Epoch 0, Iteration 100, loss = 1.7835
Checking accuracy on validation set
Got 421 / 1000 correct (42.10)

Epoch 0, Iteration 200, loss = 1.3915
Checking accuracy on validation set
Got 451 / 1000 correct (45.10)

Epoch 0, Iteration 300, loss = 1.5871
Checking accuracy on validation set
Got 510 / 1000 correct (51.00)

Epoch 0, Iteration 400, loss = 1.4586
Checking accuracy on validation set
Got 529 / 1000 correct (52.90)

Epoch 0, Iteration 500, loss = 1.1199
```

```
Checking accuracy on validation set
Got 506 / 1000 correct (50.60)

Epoch 0, Iteration 600, loss = 1.1241
Checking accuracy on validation set
Got 541 / 1000 correct (54.10)

Epoch 0, Iteration 700, loss = 1.0821
Checking accuracy on validation set
Got 552 / 1000 correct (55.20)

Epoch 1, Iteration 0, loss = 1.1126
Checking accuracy on validation set
Got 583 / 1000 correct (58.30)

Epoch 1, Iteration 100, loss = 1.2216
Checking accuracy on validation set
Got 580 / 1000 correct (58.00)

Epoch 1, Iteration 200, loss = 1.0497
Checking accuracy on validation set
Got 580 / 1000 correct (58.00)

Epoch 1, Iteration 300, loss = 1.1872
Checking accuracy on validation set
Got 615 / 1000 correct (61.50)

Epoch 1, Iteration 400, loss = 1.0578
Checking accuracy on validation set
Got 609 / 1000 correct (60.90)

Epoch 1, Iteration 500, loss = 0.9512
Checking accuracy on validation set
Got 626 / 1000 correct (62.60)

Epoch 1, Iteration 600, loss = 0.9448
Checking accuracy on validation set
Got 620 / 1000 correct (62.00)

Epoch 1, Iteration 700, loss = 1.0367
Checking accuracy on validation set
Got 620 / 1000 correct (62.00)

Epoch 2, Iteration 0, loss = 0.9432
Checking accuracy on validation set
Got 618 / 1000 correct (61.80)

Epoch 2, Iteration 100, loss = 1.2311
```

```
Checking accuracy on validation set
Got 644 / 1000 correct (64.40)

Epoch 2, Iteration 200, loss = 1.0270
Checking accuracy on validation set
Got 625 / 1000 correct (62.50)

Epoch 2, Iteration 300, loss = 0.6286
Checking accuracy on validation set
Got 636 / 1000 correct (63.60)

Epoch 2, Iteration 400, loss = 0.7940
Checking accuracy on validation set
Got 629 / 1000 correct (62.90)

Epoch 2, Iteration 500, loss = 0.8733
Checking accuracy on validation set
Got 631 / 1000 correct (63.10)

Epoch 2, Iteration 600, loss = 0.7626
Checking accuracy on validation set
Got 656 / 1000 correct (65.60)

Epoch 2, Iteration 700, loss = 0.9407
Checking accuracy on validation set
Got 651 / 1000 correct (65.10)

Epoch 3, Iteration 0, loss = 0.6241
Checking accuracy on validation set
Got 646 / 1000 correct (64.60)

Epoch 3, Iteration 100, loss = 0.5352
Checking accuracy on validation set
Got 649 / 1000 correct (64.90)

Epoch 3, Iteration 200, loss = 0.7777
Checking accuracy on validation set
Got 640 / 1000 correct (64.00)

Epoch 3, Iteration 300, loss = 0.5430
Checking accuracy on validation set
Got 653 / 1000 correct (65.30)

Epoch 3, Iteration 400, loss = 0.7170
Checking accuracy on validation set
Got 648 / 1000 correct (64.80)

Epoch 3, Iteration 500, loss = 0.7274
```

```
Checking accuracy on validation set
Got 655 / 1000 correct (65.50)

Epoch 3, Iteration 600, loss = 0.6654
Checking accuracy on validation set
Got 641 / 1000 correct (64.10)

Epoch 3, Iteration 700, loss = 0.7762
Checking accuracy on validation set
Got 656 / 1000 correct (65.60)

Epoch 4, Iteration 0, loss = 0.4312
Checking accuracy on validation set
Got 668 / 1000 correct (66.80)

Epoch 4, Iteration 100, loss = 0.5915
Checking accuracy on validation set
Got 650 / 1000 correct (65.00)

Epoch 4, Iteration 200, loss = 0.6140
Checking accuracy on validation set
Got 644 / 1000 correct (64.40)

Epoch 4, Iteration 300, loss = 0.6151
Checking accuracy on validation set
Got 640 / 1000 correct (64.00)

Epoch 4, Iteration 400, loss = 0.4157
Checking accuracy on validation set
Got 649 / 1000 correct (64.90)

Epoch 4, Iteration 500, loss = 0.5189
Checking accuracy on validation set
Got 648 / 1000 correct (64.80)

Epoch 4, Iteration 600, loss = 0.8579
Checking accuracy on validation set
Got 653 / 1000 correct (65.30)

Epoch 4, Iteration 700, loss = 0.6421
Checking accuracy on validation set
Got 654 / 1000 correct (65.40)

Epoch 5, Iteration 0, loss = 0.3781
Checking accuracy on validation set
Got 647 / 1000 correct (64.70)

Epoch 5, Iteration 100, loss = 0.4149
```

```
Checking accuracy on validation set
Got 645 / 1000 correct (64.50)

Epoch 5, Iteration 200, loss = 0.5388
Checking accuracy on validation set
Got 637 / 1000 correct (63.70)

Epoch 5, Iteration 300, loss = 0.3386
Checking accuracy on validation set
Got 653 / 1000 correct (65.30)

Epoch 5, Iteration 400, loss = 0.2910
Checking accuracy on validation set
Got 664 / 1000 correct (66.40)

Epoch 5, Iteration 500, loss = 0.4093
Checking accuracy on validation set
Got 630 / 1000 correct (63.00)

Epoch 5, Iteration 600, loss = 0.3622
Checking accuracy on validation set
Got 646 / 1000 correct (64.60)

Epoch 5, Iteration 700, loss = 0.3400
Checking accuracy on validation set
Got 632 / 1000 correct (63.20)

Epoch 6, Iteration 0, loss = 0.1573
Checking accuracy on validation set
Got 650 / 1000 correct (65.00)

Epoch 6, Iteration 100, loss = 0.2719
Checking accuracy on validation set
Got 652 / 1000 correct (65.20)

Epoch 6, Iteration 200, loss = 0.2456
Checking accuracy on validation set
Got 638 / 1000 correct (63.80)

Epoch 6, Iteration 300, loss = 0.2910
Checking accuracy on validation set
Got 647 / 1000 correct (64.70)

Epoch 6, Iteration 400, loss = 0.2731
Checking accuracy on validation set
Got 644 / 1000 correct (64.40)

Epoch 6, Iteration 500, loss = 0.3571
```

```
Checking accuracy on validation set
Got 623 / 1000 correct (62.30)

Epoch 6, Iteration 600, loss = 0.3025
Checking accuracy on validation set
Got 626 / 1000 correct (62.60)

Epoch 6, Iteration 700, loss = 0.2089
Checking accuracy on validation set
Got 622 / 1000 correct (62.20)

Epoch 7, Iteration 0, loss = 0.2481
Checking accuracy on validation set
Got 652 / 1000 correct (65.20)

Epoch 7, Iteration 100, loss = 0.1682
Checking accuracy on validation set
Got 647 / 1000 correct (64.70)

Epoch 7, Iteration 200, loss = 0.2243
Checking accuracy on validation set
Got 650 / 1000 correct (65.00)

Epoch 7, Iteration 300, loss = 0.1850
Checking accuracy on validation set
Got 636 / 1000 correct (63.60)

Epoch 7, Iteration 400, loss = 0.1420
Checking accuracy on validation set
Got 633 / 1000 correct (63.30)

Epoch 7, Iteration 500, loss = 0.1992
Checking accuracy on validation set
Got 638 / 1000 correct (63.80)

Epoch 7, Iteration 600, loss = 0.2456
Checking accuracy on validation set
Got 639 / 1000 correct (63.90)

Epoch 7, Iteration 700, loss = 0.2856
Checking accuracy on validation set
Got 629 / 1000 correct (62.90)

Epoch 8, Iteration 0, loss = 0.0965
Checking accuracy on validation set
Got 628 / 1000 correct (62.80)

Epoch 8, Iteration 100, loss = 0.1769
```

```
Checking accuracy on validation set
Got 644 / 1000 correct (64.40)

Epoch 8, Iteration 200, loss = 0.0863
Checking accuracy on validation set
Got 638 / 1000 correct (63.80)

Epoch 8, Iteration 300, loss = 0.2237
Checking accuracy on validation set
Got 623 / 1000 correct (62.30)

Epoch 8, Iteration 400, loss = 0.3130
Checking accuracy on validation set
Got 617 / 1000 correct (61.70)

Epoch 8, Iteration 500, loss = 0.1709
Checking accuracy on validation set
Got 630 / 1000 correct (63.00)

Epoch 8, Iteration 600, loss = 0.0779
Checking accuracy on validation set
Got 614 / 1000 correct (61.40)

Epoch 8, Iteration 700, loss = 0.2981
Checking accuracy on validation set
Got 627 / 1000 correct (62.70)

Epoch 9, Iteration 0, loss = 0.0837
Checking accuracy on validation set
Got 626 / 1000 correct (62.60)

Epoch 9, Iteration 100, loss = 0.0700
Checking accuracy on validation set
Got 638 / 1000 correct (63.80)

Epoch 9, Iteration 200, loss = 0.1240
Checking accuracy on validation set
Got 634 / 1000 correct (63.40)

Epoch 9, Iteration 300, loss = 0.1387
Checking accuracy on validation set
Got 624 / 1000 correct (62.40)

Epoch 9, Iteration 400, loss = 0.1412
Checking accuracy on validation set
Got 631 / 1000 correct (63.10)

Epoch 9, Iteration 500, loss = 0.1931
```

```
Checking accuracy on validation set
Got 629 / 1000 correct (62.90)

Epoch 9, Iteration 600, loss = 0.1681
Checking accuracy on validation set
Got 630 / 1000 correct (63.00)

Epoch 9, Iteration 700, loss = 0.1472
Checking accuracy on validation set
Got 625 / 1000 correct (62.50)

Maximum accuracy attained:  66.8
```

2.6 Test set – run this only once

Now we test our model on the test set . Think about how this compares to your validation set accuracy. You should see improvement of about 2-3% over the vanilla convnet model. * Use this part to tune your Attention module and then move on to the next parts. *

[86]:

```
earlyAttention = model
check_accuracy(loader_test, earlyAttention)
```

```
Checking accuracy on test set
Got 6112 / 10000 correct (61.12)
```

[86]:

```
61.12
```

2.7 Single Attention Block: Late attention; After the second conv layer. (10 points)

[87]:

```
channel_1 = 64
channel_2 = 32
learning_rate = 1e-3

# TODO: Use the above Attention module after the Second Convolutional layer.
# Essentially the architecture should be␣
 ↪[Conv->Relu->Conv->Relu->Attention->Relu->Linear]

model = nn.Sequential(
    nn.Conv2d(3, channel_1, 3, padding=1, stride=1),
    nn.ReLU(),
    nn.Conv2d(channel_1, channel_2, 3, padding=1, stride=1),
    nn.ReLU(),
    Attention(channel_2),
    nn.ReLU(),
    Flatten(),
    nn.Linear(channel_2*32*32, num_classes),
)
```

```
optimizer = optim.Adam(model.parameters(), lr=learning_rate)

train(model, optimizer, epochs=10)
```

```
Epoch 0, Iteration 0, loss = 2.3091
Checking accuracy on validation set
Got 125 / 1000 correct (12.50)

Epoch 0, Iteration 100, loss = 1.5839
Checking accuracy on validation set
Got 438 / 1000 correct (43.80)

Epoch 0, Iteration 200, loss = 1.9136
Checking accuracy on validation set
Got 497 / 1000 correct (49.70)

Epoch 0, Iteration 300, loss = 1.3352
Checking accuracy on validation set
Got 527 / 1000 correct (52.70)

Epoch 0, Iteration 400, loss = 1.3180
Checking accuracy on validation set
Got 532 / 1000 correct (53.20)

Epoch 0, Iteration 500, loss = 1.3627
Checking accuracy on validation set
Got 532 / 1000 correct (53.20)

Epoch 0, Iteration 600, loss = 1.4136
Checking accuracy on validation set
Got 554 / 1000 correct (55.40)

Epoch 0, Iteration 700, loss = 1.3577
Checking accuracy on validation set
Got 567 / 1000 correct (56.70)

Epoch 1, Iteration 0, loss = 1.3397
Checking accuracy on validation set
Got 557 / 1000 correct (55.70)

Epoch 1, Iteration 100, loss = 1.0281
Checking accuracy on validation set
Got 587 / 1000 correct (58.70)

Epoch 1, Iteration 200, loss = 0.8183
Checking accuracy on validation set
```

```
Got 574 / 1000 correct (57.40)

Epoch 1, Iteration 300, loss = 0.9360
Checking accuracy on validation set
Got 586 / 1000 correct (58.60)

Epoch 1, Iteration 400, loss = 0.9947
Checking accuracy on validation set
Got 602 / 1000 correct (60.20)

Epoch 1, Iteration 500, loss = 0.8387
Checking accuracy on validation set
Got 607 / 1000 correct (60.70)

Epoch 1, Iteration 600, loss = 0.8237
Checking accuracy on validation set
Got 594 / 1000 correct (59.40)

Epoch 1, Iteration 700, loss = 0.9452
Checking accuracy on validation set
Got 585 / 1000 correct (58.50)

Epoch 2, Iteration 0, loss = 1.2590
Checking accuracy on validation set
Got 610 / 1000 correct (61.00)

Epoch 2, Iteration 100, loss = 0.9441
Checking accuracy on validation set
Got 589 / 1000 correct (58.90)

Epoch 2, Iteration 200, loss = 1.1428
Checking accuracy on validation set
Got 598 / 1000 correct (59.80)

Epoch 2, Iteration 300, loss = 0.7899
Checking accuracy on validation set
Got 585 / 1000 correct (58.50)

Epoch 2, Iteration 400, loss = 0.9110
Checking accuracy on validation set
Got 617 / 1000 correct (61.70)

Epoch 2, Iteration 500, loss = 0.9792
Checking accuracy on validation set
Got 605 / 1000 correct (60.50)

Epoch 2, Iteration 600, loss = 1.2413
Checking accuracy on validation set
```

```
Got 620 / 1000 correct (62.00)

Epoch 2, Iteration 700, loss = 0.9184
Checking accuracy on validation set
Got 630 / 1000 correct (63.00)

Epoch 3, Iteration 0, loss = 0.7756
Checking accuracy on validation set
Got 619 / 1000 correct (61.90)

Epoch 3, Iteration 100, loss = 0.7184
Checking accuracy on validation set
Got 624 / 1000 correct (62.40)

Epoch 3, Iteration 200, loss = 0.6576
Checking accuracy on validation set
Got 612 / 1000 correct (61.20)

Epoch 3, Iteration 300, loss = 0.7145
Checking accuracy on validation set
Got 611 / 1000 correct (61.10)

Epoch 3, Iteration 400, loss = 0.9338
Checking accuracy on validation set
Got 632 / 1000 correct (63.20)

Epoch 3, Iteration 500, loss = 0.9933
Checking accuracy on validation set
Got 630 / 1000 correct (63.00)

Epoch 3, Iteration 600, loss = 0.5983
Checking accuracy on validation set
Got 618 / 1000 correct (61.80)

Epoch 3, Iteration 700, loss = 0.7197
Checking accuracy on validation set
Got 619 / 1000 correct (61.90)

Epoch 4, Iteration 0, loss = 0.6874
Checking accuracy on validation set
Got 619 / 1000 correct (61.90)

Epoch 4, Iteration 100, loss = 0.5170
Checking accuracy on validation set
Got 616 / 1000 correct (61.60)

Epoch 4, Iteration 200, loss = 0.4866
Checking accuracy on validation set
```

```
Got 613 / 1000 correct (61.30)

Epoch 4, Iteration 300, loss = 1.1615
Checking accuracy on validation set
Got 611 / 1000 correct (61.10)

Epoch 4, Iteration 400, loss = 0.8170
Checking accuracy on validation set
Got 624 / 1000 correct (62.40)

Epoch 4, Iteration 500, loss = 1.0931
Checking accuracy on validation set
Got 595 / 1000 correct (59.50)

Epoch 4, Iteration 600, loss = 0.7278
Checking accuracy on validation set
Got 639 / 1000 correct (63.90)

Epoch 4, Iteration 700, loss = 0.5860
Checking accuracy on validation set
Got 622 / 1000 correct (62.20)

Epoch 5, Iteration 0, loss = 0.4848
Checking accuracy on validation set
Got 619 / 1000 correct (61.90)

Epoch 5, Iteration 100, loss = 0.4473
Checking accuracy on validation set
Got 610 / 1000 correct (61.00)

Epoch 5, Iteration 200, loss = 0.4268
Checking accuracy on validation set
Got 612 / 1000 correct (61.20)

Epoch 5, Iteration 300, loss = 0.6967
Checking accuracy on validation set
Got 616 / 1000 correct (61.60)

Epoch 5, Iteration 400, loss = 0.7446
Checking accuracy on validation set
Got 608 / 1000 correct (60.80)

Epoch 5, Iteration 500, loss = 0.5288
Checking accuracy on validation set
Got 618 / 1000 correct (61.80)

Epoch 5, Iteration 600, loss = 0.8227
Checking accuracy on validation set
```

```
Got 612 / 1000 correct (61.20)

Epoch 5, Iteration 700, loss = 0.4823
Checking accuracy on validation set
Got 625 / 1000 correct (62.50)

Epoch 6, Iteration 0, loss = 0.5073
Checking accuracy on validation set
Got 629 / 1000 correct (62.90)

Epoch 6, Iteration 100, loss = 0.4172
Checking accuracy on validation set
Got 612 / 1000 correct (61.20)

Epoch 6, Iteration 200, loss = 0.2946
Checking accuracy on validation set
Got 617 / 1000 correct (61.70)

Epoch 6, Iteration 300, loss = 0.3361
Checking accuracy on validation set
Got 607 / 1000 correct (60.70)

Epoch 6, Iteration 400, loss = 0.5107
Checking accuracy on validation set
Got 590 / 1000 correct (59.00)

Epoch 6, Iteration 500, loss = 0.4370
Checking accuracy on validation set
Got 618 / 1000 correct (61.80)

Epoch 6, Iteration 600, loss = 0.6390
Checking accuracy on validation set
Got 597 / 1000 correct (59.70)

Epoch 6, Iteration 700, loss = 0.4263
Checking accuracy on validation set
Got 599 / 1000 correct (59.90)

Epoch 7, Iteration 0, loss = 0.3595
Checking accuracy on validation set
Got 595 / 1000 correct (59.50)

Epoch 7, Iteration 100, loss = 0.2347
Checking accuracy on validation set
Got 600 / 1000 correct (60.00)

Epoch 7, Iteration 200, loss = 0.4223
Checking accuracy on validation set
```

```
Got 598 / 1000 correct (59.80)

Epoch 7, Iteration 300, loss = 0.2486
Checking accuracy on validation set
Got 592 / 1000 correct (59.20)

Epoch 7, Iteration 400, loss = 0.5701
Checking accuracy on validation set
Got 587 / 1000 correct (58.70)

Epoch 7, Iteration 500, loss = 0.3920
Checking accuracy on validation set
Got 591 / 1000 correct (59.10)

Epoch 7, Iteration 600, loss = 0.6804
Checking accuracy on validation set
Got 607 / 1000 correct (60.70)

Epoch 7, Iteration 700, loss = 0.4118
Checking accuracy on validation set
Got 595 / 1000 correct (59.50)

Epoch 8, Iteration 0, loss = 0.3907
Checking accuracy on validation set
Got 614 / 1000 correct (61.40)

Epoch 8, Iteration 100, loss = 0.2254
Checking accuracy on validation set
Got 596 / 1000 correct (59.60)

Epoch 8, Iteration 200, loss = 0.2503
Checking accuracy on validation set
Got 598 / 1000 correct (59.80)

Epoch 8, Iteration 300, loss = 0.3949
Checking accuracy on validation set
Got 578 / 1000 correct (57.80)

Epoch 8, Iteration 400, loss = 0.3272
Checking accuracy on validation set
Got 581 / 1000 correct (58.10)

Epoch 8, Iteration 500, loss = 0.4342
Checking accuracy on validation set
Got 594 / 1000 correct (59.40)

Epoch 8, Iteration 600, loss = 0.3267
Checking accuracy on validation set
```

```
Got 591 / 1000 correct (59.10)

Epoch 8, Iteration 700, loss = 0.4204
Checking accuracy on validation set
Got 602 / 1000 correct (60.20)

Epoch 9, Iteration 0, loss = 0.2279
Checking accuracy on validation set
Got 575 / 1000 correct (57.50)

Epoch 9, Iteration 100, loss = 0.2174
Checking accuracy on validation set
Got 583 / 1000 correct (58.30)

Epoch 9, Iteration 200, loss = 0.1031
Checking accuracy on validation set
Got 584 / 1000 correct (58.40)

Epoch 9, Iteration 300, loss = 0.2592
Checking accuracy on validation set
Got 583 / 1000 correct (58.30)

Epoch 9, Iteration 400, loss = 0.2464
Checking accuracy on validation set
Got 578 / 1000 correct (57.80)

Epoch 9, Iteration 500, loss = 0.2619
Checking accuracy on validation set
Got 567 / 1000 correct (56.70)

Epoch 9, Iteration 600, loss = 0.3589
Checking accuracy on validation set
Got 572 / 1000 correct (57.20)

Epoch 9, Iteration 700, loss = 0.1551
Checking accuracy on validation set
Got 577 / 1000 correct (57.70)

Maximum accuracy attained:  63.9
```

2.8 Test set – run this only once

Now we test our model on the test set . Think about how this compares to your validation set accuracy.

[88]:

```
lateAttention = model
check_accuracy(loader_test, lateAttention)
```

```
Checking accuracy on test set
Got 5831 / 10000 correct (58.31)
```

[88]:
```
58.309999999999995
```

2.8.1 Inline Question 2: Provide one example each of usage of self-attention and attention in computer vision. Explain the difference between the two. (5 points)

Your Answer: * Usage of self-attention: The Vision Transformer (ViT) leverages self-attention to process images as sequences of patches, enabling it to capture long-range dependencies and interactions between different parts of the image. This approach provides an alternative to traditional CNNs in object classification, offering competitive performance on various computer vision tasks.

- Usage of attention: The Squeeze-and-Excitation Network (SENet) enhances CNNs by recalibrating channel-wise feature responses using attention method. This allows the network to focus on the most relevant parts of the input sequence, improving performance and global feature representation.
- Differences:
  - Self-attention is typically part of a layer (like in ViT) capturing non-local dependencies by computing relationships between all pairs of elements in an input. It can be used very frequently in a network. It models within-sequence relationships mainly. The weights are computed from the input sequence directly, and hence different inputs correspond to different weights. It improves performance by capturing dependencies within the input sequence more effectively by doing dot-products using queries and keys for the weights.
  - Attention is typically an individual module that concatenates encoder and decoder, serving the function of capturing the sequence-to-sequence relationships in the network. It's often used only a few times to concatenate two modules (like an encoder and a decoder) in a network. And hence, it models sequence-to-sequence relationships mainly. The weights in the MLP are learnt, and hence different inputs correspond to the same group of learnt weights. Attention improves performance by emphasizing important features based on the learnt weights indicating relative importance.

2.9 Double Attention Blocks: After conv layers 1 and 2 (10 points)

[89]:
```python
channel_1 = 64
channel_2 = 32
learning_rate = 1e-3

# TODO: Use the above Attention module after the Second Convolutional layer.
# Essentially the architecture should be [Conv->Relu->Attention->Relu->Conv->Relu->Attention->Relu->Linear]

model = nn.Sequential(
    nn.Conv2d(3, channel_1, 3, padding=1, stride=1),
    nn.ReLU(),
    Attention(channel_1),
    nn.ReLU(),
    nn.Conv2d(channel_1, channel_2, 3, padding=1, stride=1),
    nn.ReLU(),
    Attention(channel_2),
    nn.ReLU(),
    Flatten(),
    nn.Linear(channel_2*32*32, num_classes),
)

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

train(model, optimizer, epochs=10)
```

```
Epoch 0, Iteration 0, loss = 2.2909
Checking accuracy on validation set
Got 109 / 1000 correct (10.90)

Epoch 0, Iteration 100, loss = 1.9164
Checking accuracy on validation set
Got 322 / 1000 correct (32.20)

Epoch 0, Iteration 200, loss = 1.5259
Checking accuracy on validation set
Got 450 / 1000 correct (45.00)

Epoch 0, Iteration 300, loss = 1.3118
Checking accuracy on validation set
Got 467 / 1000 correct (46.70)

Epoch 0, Iteration 400, loss = 1.4503
Checking accuracy on validation set
Got 506 / 1000 correct (50.60)

Epoch 0, Iteration 500, loss = 1.3227
Checking accuracy on validation set
Got 515 / 1000 correct (51.50)

Epoch 0, Iteration 600, loss = 1.2445
Checking accuracy on validation set
Got 524 / 1000 correct (52.40)

Epoch 0, Iteration 700, loss = 1.3021
Checking accuracy on validation set
Got 532 / 1000 correct (53.20)

Epoch 1, Iteration 0, loss = 1.2930
Checking accuracy on validation set
```

```
Got 530 / 1000 correct (53.00)

Epoch 1, Iteration 100, loss = 1.3517
Checking accuracy on validation set
Got 542 / 1000 correct (54.20)

Epoch 1, Iteration 200, loss = 1.2945
Checking accuracy on validation set
Got 567 / 1000 correct (56.70)

Epoch 1, Iteration 300, loss = 1.3331
Checking accuracy on validation set
Got 538 / 1000 correct (53.80)

Epoch 1, Iteration 400, loss = 1.0128
Checking accuracy on validation set
Got 551 / 1000 correct (55.10)

Epoch 1, Iteration 500, loss = 1.3041
Checking accuracy on validation set
Got 530 / 1000 correct (53.00)

Epoch 1, Iteration 600, loss = 1.2753
Checking accuracy on validation set
Got 551 / 1000 correct (55.10)

Epoch 1, Iteration 700, loss = 1.1456
Checking accuracy on validation set
Got 576 / 1000 correct (57.60)

Epoch 2, Iteration 0, loss = 1.1693
Checking accuracy on validation set
Got 576 / 1000 correct (57.60)

Epoch 2, Iteration 100, loss = 1.0134
Checking accuracy on validation set
Got 587 / 1000 correct (58.70)

Epoch 2, Iteration 200, loss = 1.1225
Checking accuracy on validation set
Got 584 / 1000 correct (58.40)

Epoch 2, Iteration 300, loss = 1.3498
Checking accuracy on validation set
Got 588 / 1000 correct (58.80)

Epoch 2, Iteration 400, loss = 1.2768
Checking accuracy on validation set
```

```
Got 596 / 1000 correct (59.60)

Epoch 2, Iteration 500, loss = 1.1820
Checking accuracy on validation set
Got 604 / 1000 correct (60.40)

Epoch 2, Iteration 600, loss = 1.4075
Checking accuracy on validation set
Got 617 / 1000 correct (61.70)

Epoch 2, Iteration 700, loss = 1.0997
Checking accuracy on validation set
Got 598 / 1000 correct (59.80)

Epoch 3, Iteration 0, loss = 0.9658
Checking accuracy on validation set
Got 604 / 1000 correct (60.40)

Epoch 3, Iteration 100, loss = 0.7417
Checking accuracy on validation set
Got 604 / 1000 correct (60.40)

Epoch 3, Iteration 200, loss = 0.9589
Checking accuracy on validation set
Got 589 / 1000 correct (58.90)

Epoch 3, Iteration 300, loss = 1.1804
Checking accuracy on validation set
Got 595 / 1000 correct (59.50)

Epoch 3, Iteration 400, loss = 1.1039
Checking accuracy on validation set
Got 622 / 1000 correct (62.20)

Epoch 3, Iteration 500, loss = 1.0229
Checking accuracy on validation set
Got 593 / 1000 correct (59.30)

Epoch 3, Iteration 600, loss = 0.9299
Checking accuracy on validation set
Got 612 / 1000 correct (61.20)

Epoch 3, Iteration 700, loss = 1.3686
Checking accuracy on validation set
Got 614 / 1000 correct (61.40)

Epoch 4, Iteration 0, loss = 0.6392
Checking accuracy on validation set
```

```
Got 607 / 1000 correct (60.70)

Epoch 4, Iteration 100, loss = 0.8663
Checking accuracy on validation set
Got 588 / 1000 correct (58.80)

Epoch 4, Iteration 200, loss = 0.8482
Checking accuracy on validation set
Got 597 / 1000 correct (59.70)

Epoch 4, Iteration 300, loss = 0.8884
Checking accuracy on validation set
Got 590 / 1000 correct (59.00)

Epoch 4, Iteration 400, loss = 0.9475
Checking accuracy on validation set
Got 612 / 1000 correct (61.20)

Epoch 4, Iteration 500, loss = 1.0487
Checking accuracy on validation set
Got 615 / 1000 correct (61.50)

Epoch 4, Iteration 600, loss = 0.7489
Checking accuracy on validation set
Got 617 / 1000 correct (61.70)

Epoch 4, Iteration 700, loss = 0.8002
Checking accuracy on validation set
Got 596 / 1000 correct (59.60)

Epoch 5, Iteration 0, loss = 0.5266
Checking accuracy on validation set
Got 615 / 1000 correct (61.50)

Epoch 5, Iteration 100, loss = 0.6291
Checking accuracy on validation set
Got 620 / 1000 correct (62.00)

Epoch 5, Iteration 200, loss = 0.6703
Checking accuracy on validation set
Got 615 / 1000 correct (61.50)

Epoch 5, Iteration 300, loss = 0.5306
Checking accuracy on validation set
Got 592 / 1000 correct (59.20)

Epoch 5, Iteration 400, loss = 0.9859
Checking accuracy on validation set
```

```
Got 611 / 1000 correct (61.10)

Epoch 5, Iteration 500, loss = 0.8990
Checking accuracy on validation set
Got 626 / 1000 correct (62.60)

Epoch 5, Iteration 600, loss = 0.7123
Checking accuracy on validation set
Got 637 / 1000 correct (63.70)

Epoch 5, Iteration 700, loss = 0.8093
Checking accuracy on validation set
Got 613 / 1000 correct (61.30)

Epoch 6, Iteration 0, loss = 0.4293
Checking accuracy on validation set
Got 633 / 1000 correct (63.30)

Epoch 6, Iteration 100, loss = 0.6259
Checking accuracy on validation set
Got 606 / 1000 correct (60.60)

Epoch 6, Iteration 200, loss = 0.5768
Checking accuracy on validation set
Got 621 / 1000 correct (62.10)

Epoch 6, Iteration 300, loss = 0.4714
Checking accuracy on validation set
Got 612 / 1000 correct (61.20)

Epoch 6, Iteration 400, loss = 0.4936
Checking accuracy on validation set
Got 615 / 1000 correct (61.50)

Epoch 6, Iteration 500, loss = 0.8145
Checking accuracy on validation set
Got 616 / 1000 correct (61.60)

Epoch 6, Iteration 600, loss = 0.5466
Checking accuracy on validation set
Got 601 / 1000 correct (60.10)

Epoch 6, Iteration 700, loss = 0.6788
Checking accuracy on validation set
Got 615 / 1000 correct (61.50)

Epoch 7, Iteration 0, loss = 0.5111
Checking accuracy on validation set
```

```
Got 618 / 1000 correct (61.80)

Epoch 7, Iteration 100, loss = 0.3314
Checking accuracy on validation set
Got 602 / 1000 correct (60.20)

Epoch 7, Iteration 200, loss = 0.4677
Checking accuracy on validation set
Got 594 / 1000 correct (59.40)

Epoch 7, Iteration 300, loss = 0.4808
Checking accuracy on validation set
Got 608 / 1000 correct (60.80)

Epoch 7, Iteration 400, loss = 0.4332
Checking accuracy on validation set
Got 607 / 1000 correct (60.70)

Epoch 7, Iteration 500, loss = 0.5851
Checking accuracy on validation set
Got 599 / 1000 correct (59.90)

Epoch 7, Iteration 600, loss = 0.4782
Checking accuracy on validation set
Got 617 / 1000 correct (61.70)

Epoch 7, Iteration 700, loss = 0.4484
Checking accuracy on validation set
Got 614 / 1000 correct (61.40)

Epoch 8, Iteration 0, loss = 0.2564
Checking accuracy on validation set
Got 606 / 1000 correct (60.60)

Epoch 8, Iteration 100, loss = 0.2846
Checking accuracy on validation set
Got 610 / 1000 correct (61.00)

Epoch 8, Iteration 200, loss = 0.3426
Checking accuracy on validation set
Got 599 / 1000 correct (59.90)

Epoch 8, Iteration 300, loss = 0.3699
Checking accuracy on validation set
Got 589 / 1000 correct (58.90)

Epoch 8, Iteration 400, loss = 0.2428
Checking accuracy on validation set
```

```
Got 605 / 1000 correct (60.50)

Epoch 8, Iteration 500, loss = 0.3250
Checking accuracy on validation set
Got 595 / 1000 correct (59.50)

Epoch 8, Iteration 600, loss = 0.4352
Checking accuracy on validation set
Got 587 / 1000 correct (58.70)

Epoch 8, Iteration 700, loss = 0.5713
Checking accuracy on validation set
Got 570 / 1000 correct (57.00)

Epoch 9, Iteration 0, loss = 0.2609
Checking accuracy on validation set
Got 594 / 1000 correct (59.40)

Epoch 9, Iteration 100, loss = 0.1942
Checking accuracy on validation set
Got 594 / 1000 correct (59.40)

Epoch 9, Iteration 200, loss = 0.2212
Checking accuracy on validation set
Got 589 / 1000 correct (58.90)

Epoch 9, Iteration 300, loss = 0.2576
Checking accuracy on validation set
Got 584 / 1000 correct (58.40)

Epoch 9, Iteration 400, loss = 0.2013
Checking accuracy on validation set
Got 601 / 1000 correct (60.10)

Epoch 9, Iteration 500, loss = 0.4081
Checking accuracy on validation set
Got 586 / 1000 correct (58.60)

Epoch 9, Iteration 600, loss = 0.4593
Checking accuracy on validation set
Got 584 / 1000 correct (58.40)

Epoch 9, Iteration 700, loss = 0.3191
Checking accuracy on validation set
Got 593 / 1000 correct (59.30)

Maximum accuracy attained:  63.7
```

2.10 Test set – run this only once

Now we test our model on the test set . Think about how this compares to your validation set accuracy.

[90]:
```
vanillaModel = model
check_accuracy(loader_test, vanillaModel)
```

```
Checking accuracy on test set
Got 5862 / 10000 correct (58.62)
```

[90]:
```
58.620000000000005
```

2.11 Resnet with Attention

Now we will experiment with applying attention within the Resnet10 architecture that we implemented in Homework 2. Please note that for a deeper model such as Resnet we do not expect significant improvements in performance with Attention

2.12 Vanilla Resnet, No Attention

The architecture for Resnet is given below, please train it and evaluate it on the test set.

[57]:
```
import torch
import torch.nn as nn

class ResNet(nn.Module):

    def __init__(self, block, layers, img_channels=3, num_classes=100,␣
 ↪batchnorm=False):
        super(ResNet, self).__init__() #layers = [1, 1, 1, 1]
        self.in_channels = 64
        self.conv1 = nn.Conv2d(img_channels, 64, kernel_size=7, stride=2,␣
 ↪padding=3)
        self.bn1 = nn.BatchNorm2d(64)
        self.relu = nn.ReLU()
        self.maxpool = nn.MaxPool2d(kernel_size=3, stride=2, padding=1)
        self.batchnorm = batchnorm
        self.layer1 = self.make_layer(block, layers[0], out_channels=64,␣
 ↪stride=1, batchnorm=batchnorm)
        self.layer2 = self.make_layer(block, layers[1], out_channels=128,␣
 ↪stride=1, batchnorm=batchnorm)
        self.layer3 = self.make_layer(block, layers[2], out_channels=256,␣
 ↪stride=1, batchnorm=batchnorm)
        self.layer4 = self.make_layer(block, layers[3], out_channels=512,␣
 ↪stride=2, batchnorm=batchnorm)

        self.averagepool = nn.AdaptiveAvgPool2d((1, 1))
        self.fc = nn.Linear(512, num_classes)


    def forward(self, x):

        x = self.conv1(x)
        if self.batchnorm:
            x = self.bn1(x)
        x = self.relu(x)
        x = self.maxpool(x)
        x = self.layer1(x)
        x = self.layer2(x)
        x = self.layer3(x)
        x = self.layer4(x)
        x = self.averagepool(x)
        x = x.reshape(x.shape[0], -1)
        x = x.reshape(x.shape[0], -1)
        x = self.fc(x)

        return x




    def make_layer(self, block, num_blocks, out_channels, stride,␣
 ↪batchnorm=False):
        downsampler = None
        layers = []
        if stride != 1 or self.in_channels != out_channels:
            downsampler = nn.Sequential(nn.Conv2d(self.in_channels,␣
 ↪out_channels, kernel_size = 1, stride = stride), nn.
 ↪BatchNorm2d(out_channels))

        layers.append(block(self.in_channels, out_channels, downsampler,␣
 ↪stride, batchnorm=batchnorm))

        self.in_channels = out_channels

        for i in range(num_blocks - 1):
            layers.append(block(self.in_channels, out_channels))


        return nn.Sequential(*layers)

class block(nn.Module):
    def __init__(self, in_channels, out_channels, downsampler = None, stride =␣
↪1, batchnorm=False):

        super(block, self).__init__()
        self.conv1 = nn.Conv2d(in_channels, out_channels, kernel_size = 3,␣
↪padding = 2)
        self.bn1 = nn.BatchNorm2d(out_channels)
        self.conv2 = nn.Conv2d(out_channels, out_channels, kernel_size = 3,␣
↪stride = stride)
        self.bn2 = nn.BatchNorm2d(out_channels)
        self.downsampler = downsampler
        self.relu = nn.ReLU()
        self.batchnorm = batchnorm


    def forward(self, x):

        residual = x
        x = self.conv1(x)
        if self.batchnorm:
            x = self.bn1(x)
        x = self.relu(x)
        x = self.conv2(x)
        if self.batchnorm:
            x = self.bn2(x)
        x = self.relu(x)

        if self.downsampler:
            residual = self.downsampler(residual)

        return self.relu(residual + x)



def ResNet10(num_classes = 100, batchnorm= False):

    return ResNet(block, [1, 1, 1, 1], num_classes=num_classes,␣
↪batchnorm=batchnorm)
```

2.13 Test set – run this only once

Now we test our model on the test set . Think about how this compares to your validation set accuracy.

[58]:

```
learning_rate = 1e-3

model = ResNet10()
```

```
optimizer = optim.Adam(model.parameters(), lr=learning_rate)

train(model, optimizer, epochs=10)

vanillaResnet = model
check_accuracy(loader_test, vanillaResnet)
```

```
Epoch 0, Iteration 0, loss = 4.7750
Checking accuracy on validation set
Got 105 / 1000 correct (10.50)

Epoch 0, Iteration 100, loss = 1.5069
Checking accuracy on validation set
Got 395 / 1000 correct (39.50)

Epoch 0, Iteration 200, loss = 1.3592
Checking accuracy on validation set
Got 449 / 1000 correct (44.90)

Epoch 0, Iteration 300, loss = 1.4675
Checking accuracy on validation set
Got 432 / 1000 correct (43.20)

Epoch 0, Iteration 400, loss = 1.2042
Checking accuracy on validation set
Got 471 / 1000 correct (47.10)

Epoch 0, Iteration 500, loss = 1.1853
Checking accuracy on validation set
Got 528 / 1000 correct (52.80)

Epoch 0, Iteration 600, loss = 1.1258
Checking accuracy on validation set
Got 571 / 1000 correct (57.10)

Epoch 0, Iteration 700, loss = 1.0300
Checking accuracy on validation set
Got 573 / 1000 correct (57.30)

Epoch 1, Iteration 0, loss = 1.0037
Checking accuracy on validation set
Got 577 / 1000 correct (57.70)

Epoch 1, Iteration 100, loss = 0.8076
Checking accuracy on validation set
Got 528 / 1000 correct (52.80)
```

```
Epoch 1, Iteration 200, loss = 0.9683
Checking accuracy on validation set
Got 623 / 1000 correct (62.30)

Epoch 1, Iteration 300, loss = 1.2502
Checking accuracy on validation set
Got 538 / 1000 correct (53.80)

Epoch 1, Iteration 400, loss = 1.0313
Checking accuracy on validation set
Got 621 / 1000 correct (62.10)

Epoch 1, Iteration 500, loss = 1.0359
Checking accuracy on validation set
Got 612 / 1000 correct (61.20)

Epoch 1, Iteration 600, loss = 1.1694
Checking accuracy on validation set
Got 628 / 1000 correct (62.80)

Epoch 1, Iteration 700, loss = 1.0434
Checking accuracy on validation set
Got 621 / 1000 correct (62.10)

Epoch 2, Iteration 0, loss = 0.9030
Checking accuracy on validation set
Got 631 / 1000 correct (63.10)

Epoch 2, Iteration 100, loss = 1.0406
Checking accuracy on validation set
Got 609 / 1000 correct (60.90)

Epoch 2, Iteration 200, loss = 0.7680
Checking accuracy on validation set
Got 640 / 1000 correct (64.00)

Epoch 2, Iteration 300, loss = 0.8748
Checking accuracy on validation set
Got 665 / 1000 correct (66.50)

Epoch 2, Iteration 400, loss = 0.9845
Checking accuracy on validation set
Got 658 / 1000 correct (65.80)

Epoch 2, Iteration 500, loss = 0.9294
Checking accuracy on validation set
Got 634 / 1000 correct (63.40)
```

```
Epoch 2, Iteration 600, loss = 1.1083
Checking accuracy on validation set
Got 662 / 1000 correct (66.20)

Epoch 2, Iteration 700, loss = 0.6598
Checking accuracy on validation set
Got 697 / 1000 correct (69.70)

Epoch 3, Iteration 0, loss = 0.8251
Checking accuracy on validation set
Got 674 / 1000 correct (67.40)

Epoch 3, Iteration 100, loss = 0.5967
Checking accuracy on validation set
Got 692 / 1000 correct (69.20)

Epoch 3, Iteration 200, loss = 0.7270
Checking accuracy on validation set
Got 680 / 1000 correct (68.00)

Epoch 3, Iteration 300, loss = 1.1079
Checking accuracy on validation set
Got 676 / 1000 correct (67.60)

Epoch 3, Iteration 400, loss = 0.6674
Checking accuracy on validation set
Got 704 / 1000 correct (70.40)

Epoch 3, Iteration 500, loss = 0.6718
Checking accuracy on validation set
Got 706 / 1000 correct (70.60)

Epoch 3, Iteration 600, loss = 0.8633
Checking accuracy on validation set
Got 707 / 1000 correct (70.70)

Epoch 3, Iteration 700, loss = 0.8723
Checking accuracy on validation set
Got 718 / 1000 correct (71.80)

Epoch 4, Iteration 0, loss = 0.6791
Checking accuracy on validation set
Got 683 / 1000 correct (68.30)

Epoch 4, Iteration 100, loss = 0.6927
Checking accuracy on validation set
Got 709 / 1000 correct (70.90)
```

```
Epoch 4, Iteration 200, loss = 0.5078
Checking accuracy on validation set
Got 706 / 1000 correct (70.60)

Epoch 4, Iteration 300, loss = 0.6036
Checking accuracy on validation set
Got 736 / 1000 correct (73.60)

Epoch 4, Iteration 400, loss = 0.7852
Checking accuracy on validation set
Got 712 / 1000 correct (71.20)

Epoch 4, Iteration 500, loss = 0.8443
Checking accuracy on validation set
Got 713 / 1000 correct (71.30)

Epoch 4, Iteration 600, loss = 0.6034
Checking accuracy on validation set
Got 730 / 1000 correct (73.00)

Epoch 4, Iteration 700, loss = 0.7621
Checking accuracy on validation set
Got 730 / 1000 correct (73.00)

Epoch 5, Iteration 0, loss = 0.5981
Checking accuracy on validation set
Got 726 / 1000 correct (72.60)

Epoch 5, Iteration 100, loss = 0.9903
Checking accuracy on validation set
Got 730 / 1000 correct (73.00)

Epoch 5, Iteration 200, loss = 0.6806
Checking accuracy on validation set
Got 739 / 1000 correct (73.90)

Epoch 5, Iteration 300, loss = 0.4643
Checking accuracy on validation set
Got 703 / 1000 correct (70.30)

Epoch 5, Iteration 400, loss = 0.6257
Checking accuracy on validation set
Got 756 / 1000 correct (75.60)

Epoch 5, Iteration 500, loss = 0.6085
Checking accuracy on validation set
Got 716 / 1000 correct (71.60)
```

```
Epoch 5, Iteration 600, loss = 0.4997
Checking accuracy on validation set
Got 727 / 1000 correct (72.70)

Epoch 5, Iteration 700, loss = 0.6201
Checking accuracy on validation set
Got 742 / 1000 correct (74.20)

Epoch 6, Iteration 0, loss = 0.5664
Checking accuracy on validation set
Got 735 / 1000 correct (73.50)

Epoch 6, Iteration 100, loss = 0.5474
Checking accuracy on validation set
Got 730 / 1000 correct (73.00)

Epoch 6, Iteration 200, loss = 0.6548
Checking accuracy on validation set
Got 735 / 1000 correct (73.50)

Epoch 6, Iteration 300, loss = 0.4773
Checking accuracy on validation set
Got 739 / 1000 correct (73.90)

Epoch 6, Iteration 400, loss = 0.4326
Checking accuracy on validation set
Got 736 / 1000 correct (73.60)

Epoch 6, Iteration 500, loss = 0.5066
Checking accuracy on validation set
Got 761 / 1000 correct (76.10)

Epoch 6, Iteration 600, loss = 0.4803
Checking accuracy on validation set
Got 750 / 1000 correct (75.00)

Epoch 6, Iteration 700, loss = 0.5982
Checking accuracy on validation set
Got 747 / 1000 correct (74.70)

Epoch 7, Iteration 0, loss = 0.5280
Checking accuracy on validation set
Got 742 / 1000 correct (74.20)

Epoch 7, Iteration 100, loss = 0.5425
Checking accuracy on validation set
Got 729 / 1000 correct (72.90)
```

```
Epoch 7, Iteration 200, loss = 0.2875
Checking accuracy on validation set
Got 764 / 1000 correct (76.40)

Epoch 7, Iteration 300, loss = 0.4327
Checking accuracy on validation set
Got 743 / 1000 correct (74.30)

Epoch 7, Iteration 400, loss = 0.4404
Checking accuracy on validation set
Got 734 / 1000 correct (73.40)

Epoch 7, Iteration 500, loss = 0.6656
Checking accuracy on validation set
Got 752 / 1000 correct (75.20)

Epoch 7, Iteration 600, loss = 0.7700
Checking accuracy on validation set
Got 762 / 1000 correct (76.20)

Epoch 7, Iteration 700, loss = 0.3530
Checking accuracy on validation set
Got 753 / 1000 correct (75.30)

Epoch 8, Iteration 0, loss = 0.4891
Checking accuracy on validation set
Got 745 / 1000 correct (74.50)

Epoch 8, Iteration 100, loss = 0.3552
Checking accuracy on validation set
Got 743 / 1000 correct (74.30)

Epoch 8, Iteration 200, loss = 0.5656
Checking accuracy on validation set
Got 738 / 1000 correct (73.80)

Epoch 8, Iteration 300, loss = 0.6213
Checking accuracy on validation set
Got 756 / 1000 correct (75.60)

Epoch 8, Iteration 400, loss = 0.6020
Checking accuracy on validation set
Got 733 / 1000 correct (73.30)

Epoch 8, Iteration 500, loss = 0.2505
Checking accuracy on validation set
Got 753 / 1000 correct (75.30)
```

```
Epoch 8, Iteration 600, loss = 0.5816
Checking accuracy on validation set
Got 770 / 1000 correct (77.00)

Epoch 8, Iteration 700, loss = 0.2314
Checking accuracy on validation set
Got 752 / 1000 correct (75.20)

Epoch 9, Iteration 0, loss = 0.4711
Checking accuracy on validation set
Got 761 / 1000 correct (76.10)

Epoch 9, Iteration 100, loss = 0.3250
Checking accuracy on validation set
Got 754 / 1000 correct (75.40)

Epoch 9, Iteration 200, loss = 0.3695
Checking accuracy on validation set
Got 762 / 1000 correct (76.20)

Epoch 9, Iteration 300, loss = 0.4426
Checking accuracy on validation set
Got 762 / 1000 correct (76.20)

Epoch 9, Iteration 400, loss = 0.3072
Checking accuracy on validation set
Got 764 / 1000 correct (76.40)

Epoch 9, Iteration 500, loss = 0.4218
Checking accuracy on validation set
Got 770 / 1000 correct (77.00)

Epoch 9, Iteration 600, loss = 0.4738
Checking accuracy on validation set
Got 769 / 1000 correct (76.90)

Epoch 9, Iteration 700, loss = 0.4158
Checking accuracy on validation set
Got 743 / 1000 correct (74.30)

Maximum accuracy attained:  77.0
Checking accuracy on test set
Got 7488 / 10000 correct (74.88)
```

```
[58]: 74.88
```

2.14 Resnet with Attention (5 points)

[92]:

```
## Resnet with Attention

learning_rate = 1e-3

# TODO: Use the above Attention module after the 2nd resnet block i.e. after␣
 ↪self.layer2.
class ResNetAttention(nn.Module):

    def __init__(self, block, layers, img_channels=3, num_classes=100,␣
 ↪batchnorm=False):
        super(ResNetAttention, self).__init__() #layers = [1, 1, 1, 1]
        self.in_channels = 64
        self.conv1 = nn.Conv2d(img_channels, 64, kernel_size=7, stride=2,␣
 ↪padding=3)
        self.bn1 = nn.BatchNorm2d(64)
        self.relu = nn.ReLU()
        self.maxpool = nn.MaxPool2d(kernel_size=3, stride=2, padding=1)
        self.batchnorm = batchnorm
        self.layer1 = self.make_layer(block, layers[0], out_channels=64,␣
 ↪stride=1, batchnorm=batchnorm)
        self.layer2 = self.make_layer(block, layers[1], out_channels=128,␣
 ↪stride=1, batchnorm=batchnorm)
        self.attention = Attention(128)
        self.layer3 = self.make_layer(block, layers[2], out_channels=256,␣
 ↪stride=1, batchnorm=batchnorm)
        self.layer4 = self.make_layer(block, layers[3], out_channels=512,␣
 ↪stride=2, batchnorm=batchnorm)

        self.averagepool = nn.AdaptiveAvgPool2d((1, 1))
        self.fc = nn.Linear(512, num_classes)


    def forward(self, x):

        x = self.conv1(x)
        if self.batchnorm:
            x = self.bn1(x)
        x = self.relu(x)
        x = self.maxpool(x)
        x = self.layer1(x)
        x = self.layer2(x)
        x = self.attention(x)
        x = self.layer3(x)
        x = self.layer4(x)
        x = self.averagepool(x)
        x = x.reshape(x.shape[0], -1)
        x = x.reshape(x.shape[0], -1)
        x = self.fc(x)

        return x

    def make_layer(self, block, num_blocks, out_channels, stride,␣
 ↪batchnorm=False):
        downsampler = None
        layers = []
        if stride != 1 or self.in_channels != out_channels:
            downsampler = nn.Sequential(nn.Conv2d(self.in_channels,␣
 ↪out_channels, kernel_size = 1, stride = stride), nn.
 ↪BatchNorm2d(out_channels))

        layers.append(block(self.in_channels, out_channels, downsampler,␣
 ↪stride, batchnorm=batchnorm))

        self.in_channels = out_channels

        for i in range(num_blocks - 1):
            layers.append(block(self.in_channels, out_channels))


        return nn.Sequential(*layers)

class ResNetAttention10block(nn.Module):

    def __init__(self, in_channels, out_channels, downsampler = None, stride =␣
 ↪1, batchnorm=False):

        super(ResNetAttention10block, self).__init__()
        self.conv1 = nn.Conv2d(in_channels, out_channels, kernel_size = 3,␣
 ↪padding = 2)
        self.bn1 = nn.BatchNorm2d(out_channels)
        self.conv2 = nn.Conv2d(out_channels, out_channels, kernel_size = 3,␣
 ↪stride = stride)
        self.bn2 = nn.BatchNorm2d(out_channels)
        self.downsampler = downsampler
        self.relu = nn.ReLU()
        self.batchnorm = batchnorm


    def forward(self, x):

        residual = x
        x = self.conv1(x)
        if self.batchnorm:
            x = self.bn1(x)
        x = self.relu(x)
        x = self.conv2(x)
        if self.batchnorm:
            x = self.bn2(x)
        x = self.relu(x)

        if self.downsampler:
            residual = self.downsampler(residual)

        return self.relu(residual + x)

def ResNetAttention10(num_classes = 100, batchnorm= False):

    return ResNetAttention(ResNetAttention10block, [1, 1, 1, 1],␣
 ↪num_classes=num_classes, batchnorm=batchnorm)

model = ResNetAttention10()

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

train(model, optimizer, epochs=10)
```

```
Epoch 0, Iteration 0, loss = 4.6545
Checking accuracy on validation set
Got 98 / 1000 correct (9.80)

Epoch 0, Iteration 100, loss = 1.7813
Checking accuracy on validation set
Got 301 / 1000 correct (30.10)

Epoch 0, Iteration 200, loss = 1.5791
Checking accuracy on validation set
Got 466 / 1000 correct (46.60)

Epoch 0, Iteration 300, loss = 1.3994
Checking accuracy on validation set
Got 482 / 1000 correct (48.20)

Epoch 0, Iteration 400, loss = 1.3925
Checking accuracy on validation set
Got 496 / 1000 correct (49.60)

Epoch 0, Iteration 500, loss = 1.2355
Checking accuracy on validation set
```

```
Got 516 / 1000 correct (51.60)

Epoch 0, Iteration 600, loss = 1.5204
Checking accuracy on validation set
Got 507 / 1000 correct (50.70)

Epoch 0, Iteration 700, loss = 1.3769
Checking accuracy on validation set
Got 584 / 1000 correct (58.40)

Epoch 1, Iteration 0, loss = 0.8449
Checking accuracy on validation set
Got 538 / 1000 correct (53.80)

Epoch 1, Iteration 100, loss = 1.1106
Checking accuracy on validation set
Got 593 / 1000 correct (59.30)

Epoch 1, Iteration 200, loss = 1.0164
Checking accuracy on validation set
Got 590 / 1000 correct (59.00)

Epoch 1, Iteration 300, loss = 1.0507
Checking accuracy on validation set
Got 581 / 1000 correct (58.10)

Epoch 1, Iteration 400, loss = 1.0679
Checking accuracy on validation set
Got 615 / 1000 correct (61.50)

Epoch 1, Iteration 500, loss = 1.1800
Checking accuracy on validation set
Got 641 / 1000 correct (64.10)

Epoch 1, Iteration 600, loss = 1.2314
Checking accuracy on validation set
Got 656 / 1000 correct (65.60)

Epoch 1, Iteration 700, loss = 0.9316
Checking accuracy on validation set
Got 683 / 1000 correct (68.30)

Epoch 2, Iteration 0, loss = 0.7903
Checking accuracy on validation set
Got 647 / 1000 correct (64.70)

Epoch 2, Iteration 100, loss = 1.0066
Checking accuracy on validation set
```

```
Got 663 / 1000 correct (66.30)

Epoch 2, Iteration 200, loss = 0.8983
Checking accuracy on validation set
Got 652 / 1000 correct (65.20)

Epoch 2, Iteration 300, loss = 0.8269
Checking accuracy on validation set
Got 653 / 1000 correct (65.30)

Epoch 2, Iteration 400, loss = 0.7509
Checking accuracy on validation set
Got 665 / 1000 correct (66.50)

Epoch 2, Iteration 500, loss = 0.9236
Checking accuracy on validation set
Got 653 / 1000 correct (65.30)

Epoch 2, Iteration 600, loss = 0.8746
Checking accuracy on validation set
Got 670 / 1000 correct (67.00)

Epoch 2, Iteration 700, loss = 0.6206
Checking accuracy on validation set
Got 702 / 1000 correct (70.20)

Epoch 3, Iteration 0, loss = 0.6253
Checking accuracy on validation set
Got 698 / 1000 correct (69.80)

Epoch 3, Iteration 100, loss = 0.7086
Checking accuracy on validation set
Got 731 / 1000 correct (73.10)

Epoch 3, Iteration 200, loss = 0.7628
Checking accuracy on validation set
Got 714 / 1000 correct (71.40)

Epoch 3, Iteration 300, loss = 0.5393
Checking accuracy on validation set
Got 720 / 1000 correct (72.00)

Epoch 3, Iteration 400, loss = 0.7262
Checking accuracy on validation set
Got 713 / 1000 correct (71.30)

Epoch 3, Iteration 500, loss = 0.8739
Checking accuracy on validation set
```

```
Got 723 / 1000 correct (72.30)

Epoch 3, Iteration 600, loss = 0.7431
Checking accuracy on validation set
Got 726 / 1000 correct (72.60)

Epoch 3, Iteration 700, loss = 0.7139
Checking accuracy on validation set
Got 708 / 1000 correct (70.80)

Epoch 4, Iteration 0, loss = 0.4678
Checking accuracy on validation set
Got 729 / 1000 correct (72.90)

Epoch 4, Iteration 100, loss = 0.4696
Checking accuracy on validation set
Got 739 / 1000 correct (73.90)

Epoch 4, Iteration 200, loss = 0.4469
Checking accuracy on validation set
Got 740 / 1000 correct (74.00)

Epoch 4, Iteration 300, loss = 0.4950
Checking accuracy on validation set
Got 723 / 1000 correct (72.30)

Epoch 4, Iteration 400, loss = 0.7305
Checking accuracy on validation set
Got 714 / 1000 correct (71.40)

Epoch 4, Iteration 500, loss = 0.4393
Checking accuracy on validation set
Got 729 / 1000 correct (72.90)

Epoch 4, Iteration 600, loss = 0.6803
Checking accuracy on validation set
Got 747 / 1000 correct (74.70)

Epoch 4, Iteration 700, loss = 0.7277
Checking accuracy on validation set
Got 739 / 1000 correct (73.90)

Epoch 5, Iteration 0, loss = 0.3798
Checking accuracy on validation set
Got 743 / 1000 correct (74.30)

Epoch 5, Iteration 100, loss = 0.6093
Checking accuracy on validation set
```

```
Got 735 / 1000 correct (73.50)

Epoch 5, Iteration 200, loss = 0.4920
Checking accuracy on validation set
Got 709 / 1000 correct (70.90)

Epoch 5, Iteration 300, loss = 0.6187
Checking accuracy on validation set
Got 748 / 1000 correct (74.80)

Epoch 5, Iteration 400, loss = 0.4713
Checking accuracy on validation set
Got 765 / 1000 correct (76.50)

Epoch 5, Iteration 500, loss = 0.6515
Checking accuracy on validation set
Got 768 / 1000 correct (76.80)

Epoch 5, Iteration 600, loss = 0.6915
Checking accuracy on validation set
Got 765 / 1000 correct (76.50)

Epoch 5, Iteration 700, loss = 0.5748
Checking accuracy on validation set
Got 756 / 1000 correct (75.60)

Epoch 6, Iteration 0, loss = 0.5636
Checking accuracy on validation set
Got 762 / 1000 correct (76.20)

Epoch 6, Iteration 100, loss = 0.4709
Checking accuracy on validation set
Got 760 / 1000 correct (76.00)

Epoch 6, Iteration 200, loss = 0.2997
Checking accuracy on validation set
Got 774 / 1000 correct (77.40)

Epoch 6, Iteration 300, loss = 0.3733
Checking accuracy on validation set
Got 773 / 1000 correct (77.30)

Epoch 6, Iteration 400, loss = 0.4140
Checking accuracy on validation set
Got 738 / 1000 correct (73.80)

Epoch 6, Iteration 500, loss = 0.7153
Checking accuracy on validation set
```

```
Got 751 / 1000 correct (75.10)

Epoch 6, Iteration 600, loss = 0.5156
Checking accuracy on validation set
Got 768 / 1000 correct (76.80)

Epoch 6, Iteration 700, loss = 0.4074
Checking accuracy on validation set
Got 767 / 1000 correct (76.70)

Epoch 7, Iteration 0, loss = 0.2979
Checking accuracy on validation set
Got 777 / 1000 correct (77.70)

Epoch 7, Iteration 100, loss = 0.3108
Checking accuracy on validation set
Got 779 / 1000 correct (77.90)

Epoch 7, Iteration 200, loss = 0.3539
Checking accuracy on validation set
Got 769 / 1000 correct (76.90)

Epoch 7, Iteration 300, loss = 0.5528
Checking accuracy on validation set
Got 760 / 1000 correct (76.00)

Epoch 7, Iteration 400, loss = 0.4538
Checking accuracy on validation set
Got 779 / 1000 correct (77.90)

Epoch 7, Iteration 500, loss = 0.6501
Checking accuracy on validation set
Got 759 / 1000 correct (75.90)

Epoch 7, Iteration 600, loss = 0.2808
Checking accuracy on validation set
Got 785 / 1000 correct (78.50)

Epoch 7, Iteration 700, loss = 0.4682
Checking accuracy on validation set
Got 755 / 1000 correct (75.50)

Epoch 8, Iteration 0, loss = 0.1710
Checking accuracy on validation set
Got 772 / 1000 correct (77.20)

Epoch 8, Iteration 100, loss = 0.4158
Checking accuracy on validation set
```

```
Got 777 / 1000 correct (77.70)

Epoch 8, Iteration 200, loss = 0.2259
Checking accuracy on validation set
Got 784 / 1000 correct (78.40)

Epoch 8, Iteration 300, loss = 0.4438
Checking accuracy on validation set
Got 784 / 1000 correct (78.40)

Epoch 8, Iteration 400, loss = 0.4175
Checking accuracy on validation set
Got 776 / 1000 correct (77.60)

Epoch 8, Iteration 500, loss = 0.2937
Checking accuracy on validation set
Got 770 / 1000 correct (77.00)

Epoch 8, Iteration 600, loss = 0.3422
Checking accuracy on validation set
Got 754 / 1000 correct (75.40)

Epoch 8, Iteration 700, loss = 0.2597
Checking accuracy on validation set
Got 780 / 1000 correct (78.00)

Epoch 9, Iteration 0, loss = 0.4149
Checking accuracy on validation set
Got 776 / 1000 correct (77.60)

Epoch 9, Iteration 100, loss = 0.3305
Checking accuracy on validation set
Got 769 / 1000 correct (76.90)

Epoch 9, Iteration 200, loss = 0.3941
Checking accuracy on validation set
Got 753 / 1000 correct (75.30)

Epoch 9, Iteration 300, loss = 0.2413
Checking accuracy on validation set
Got 761 / 1000 correct (76.10)

Epoch 9, Iteration 400, loss = 0.2372
Checking accuracy on validation set
Got 778 / 1000 correct (77.80)

Epoch 9, Iteration 500, loss = 0.4171
Checking accuracy on validation set
```

```
Got 761 / 1000 correct (76.10)

Epoch 9, Iteration 600, loss = 0.4070
Checking accuracy on validation set
Got 780 / 1000 correct (78.00)

Epoch 9, Iteration 700, loss = 0.2234
Checking accuracy on validation set
Got 786 / 1000 correct (78.60)

Maximum accuracy attained:  78.60000000000001
```

2.15 Test set – run this only once

Now we test our model on the test set . Think about how this compares to your validation set accuracy.

[93]:

```
AttentionResnet = model
check_accuracy(loader_test, AttentionResnet)
```

```
Checking accuracy on test set
Got 7701 / 10000 correct (77.01)
```

[93]:

```
77.01
```

2.16 Inline Question 3: Rank the above models based on their performance on test dataset (15 points)

(You are encouraged to run each of the experiments (training) at least 3 times to get an average estimate)

Report the test accuracies alongside the model names. For example, 1. Vanilla CNN (57.45%, 57.99%).. etc

1. Resnet with attention (75.98%, 76.34%, 77.01%)
2. Vanilla Resnet (74.88%, 74.43%, 73.52%)
3. CNN with early attention (60.60%, 62.32%, 61.12%)
4. CNN with double attention blocks (58.31%, 60.24%, 58.62%)
5. Vanilla CNN (59.51%, 56.98%, 59.52%)
6. CNN with late attention (56.37%, 57.68%, 58.30%)

2.16.1 Bonus Question (Ungraded): Can you give a possible explanation that supports the rankings?

Your Answer:

Sometimes Resnet with attention performs worse than Resnet vanilla. This is because the sampling efficiency decreases as we use Resnet with attention instead of Resnet, making the network require more epochs to train. After increasing the number of training epochs, the performance of Resnet with attention stably gets better than the vanilla one.