assignment4

June 9, 2023

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.

[23]:
```
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
```

[24]:
```
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.

[25]:
```
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

[26]:
```
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

[27]:
```
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

[28]:
```
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)
```

[29]:
```
# 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.

[30]:
```
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 = 2.3113
Checking accuracy on validation set
Got 103 / 1000 correct (10.30)

Epoch 0, Iteration 100, loss = 1.7587
Checking accuracy on validation set
Got 381 / 1000 correct (38.10)

Epoch 0, Iteration 200, loss = 1.7377
Checking accuracy on validation set
Got 440 / 1000 correct (44.00)

Epoch 0, Iteration 300, loss = 1.5314
Checking accuracy on validation set
Got 469 / 1000 correct (46.90)

Epoch 0, Iteration 400, loss = 1.6199
Checking accuracy on validation set
Got 478 / 1000 correct (47.80)

Epoch 0, Iteration 500, loss = 1.5121
Checking accuracy on validation set
Got 503 / 1000 correct (50.30)
```

```
Epoch 0, Iteration 600, loss = 1.3052
Checking accuracy on validation set
Got 533 / 1000 correct (53.30)

Epoch 0, Iteration 700, loss = 1.3668
Checking accuracy on validation set
Got 503 / 1000 correct (50.30)

Maximum accuracy attained:  53.300000000000004
```

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

[31]:
```
vanillaModel = model
check_accuracy(loader_test, vanillaModel)
```

```
Checking accuracy on test set
Got 5319 / 10000 correct (53.19)
```

[31]: 53.190000000000005

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: Non-local means

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

[32]:
```
# 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, kernel_size =1,␣
 ↪padding = 0, stride =1)
        self.conv_key = nn.Conv2d( in_channels, in_channels, kernel_size =1,␣
 ↪padding = 0, stride =1)
        self.conv_value = nn.Conv2d( in_channels, in_channels, kernel_size =1,␣
 ↪padding = 0, stride =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).reshape(N, C, H*W)
#RuntimeError: shape '[64, 64, 1024]' is invalid for input of size 4734976
        # TODO: Pass the input through conv_key, reshape the output volume to␣
 ↪(N, C, H*W)
        k = self.conv_key(x).reshape(N, C, H*W)

        # TODO: Pass the input through conv_value, reshape the output volume to␣
 ↪(N, C, H*W)
        v = self.conv_value(x).reshape(N, C, H*W)

#         x_r = x.reshape(N, C, H*W)
#         print(q.shape)
#         print(q.transpose(1, 2).shape)
#         print(k.shape)
        # 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

        # Calculate attention scores
        attention_scores = torch.matmul(q.transpose(1, 2), k) / torch.
 ↪sqrt(torch.tensor(C))
        attention_scores = torch.softmax(attention_scores, dim=-1)

#         print(attention_scores.shape)
#         print(v.shape)
        # Apply attention to values
```

```
        attention = torch.matmul(attention_scores, v.transpose(1, 2))

        # 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)

[33]:
```
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, 1 ,1),
    nn.ReLU(),
    Attention(channel_1),
    nn.ReLU(),
    nn.Conv2d(channel_1, channel_2, 3, 1, 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 = 2.3087
Checking accuracy on validation set
Got 112 / 1000 correct (11.20)

Epoch 0, Iteration 100, loss = 1.5807
Checking accuracy on validation set
Got 402 / 1000 correct (40.20)

Epoch 0, Iteration 200, loss = 1.4572
Checking accuracy on validation set
Got 463 / 1000 correct (46.30)

Epoch 0, Iteration 300, loss = 1.3600
Checking accuracy on validation set
Got 520 / 1000 correct (52.00)
```

```
Epoch 0, Iteration 400, loss = 1.3550
Checking accuracy on validation set
Got 540 / 1000 correct (54.00)

Epoch 0, Iteration 500, loss = 1.4651
Checking accuracy on validation set
Got 557 / 1000 correct (55.70)

Epoch 0, Iteration 600, loss = 1.2878
Checking accuracy on validation set
Got 579 / 1000 correct (57.90)

Epoch 0, Iteration 700, loss = 1.2797
Checking accuracy on validation set
Got 583 / 1000 correct (58.30)

Epoch 1, Iteration 0, loss = 0.8786
Checking accuracy on validation set
Got 597 / 1000 correct (59.70)

Epoch 1, Iteration 100, loss = 1.0277
Checking accuracy on validation set
Got 610 / 1000 correct (61.00)

Epoch 1, Iteration 200, loss = 0.9250
Checking accuracy on validation set
Got 605 / 1000 correct (60.50)

Epoch 1, Iteration 300, loss = 0.8911
Checking accuracy on validation set
Got 608 / 1000 correct (60.80)

Epoch 1, Iteration 400, loss = 1.1953
Checking accuracy on validation set
Got 625 / 1000 correct (62.50)

Epoch 1, Iteration 500, loss = 0.8928
Checking accuracy on validation set
Got 649 / 1000 correct (64.90)

Epoch 1, Iteration 600, loss = 1.0317
Checking accuracy on validation set
Got 639 / 1000 correct (63.90)

Epoch 1, Iteration 700, loss = 0.9095
Checking accuracy on validation set
Got 636 / 1000 correct (63.60)
```

```
Epoch 2, Iteration 0, loss = 0.6403
Checking accuracy on validation set
Got 639 / 1000 correct (63.90)

Epoch 2, Iteration 100, loss = 0.6986
Checking accuracy on validation set
Got 641 / 1000 correct (64.10)

Epoch 2, Iteration 200, loss = 0.7575
Checking accuracy on validation set
Got 637 / 1000 correct (63.70)

Epoch 2, Iteration 300, loss = 0.5882
Checking accuracy on validation set
Got 645 / 1000 correct (64.50)

Epoch 2, Iteration 400, loss = 0.7798
Checking accuracy on validation set
Got 656 / 1000 correct (65.60)

Epoch 2, Iteration 500, loss = 0.7210
Checking accuracy on validation set
Got 641 / 1000 correct (64.10)

Epoch 2, Iteration 600, loss = 0.7281
Checking accuracy on validation set
Got 638 / 1000 correct (63.80)

Epoch 2, Iteration 700, loss = 0.7116
Checking accuracy on validation set
Got 652 / 1000 correct (65.20)

Epoch 3, Iteration 0, loss = 0.4813
Checking accuracy on validation set
Got 645 / 1000 correct (64.50)

Epoch 3, Iteration 100, loss = 0.5037
Checking accuracy on validation set
Got 645 / 1000 correct (64.50)

Epoch 3, Iteration 200, loss = 0.5888
Checking accuracy on validation set
Got 651 / 1000 correct (65.10)

Epoch 3, Iteration 300, loss = 0.6519
Checking accuracy on validation set
Got 654 / 1000 correct (65.40)
```

```
Epoch 3, Iteration 400, loss = 0.6561
Checking accuracy on validation set
Got 645 / 1000 correct (64.50)

Epoch 3, Iteration 500, loss = 0.6153
Checking accuracy on validation set
Got 658 / 1000 correct (65.80)

Epoch 3, Iteration 600, loss = 0.5554
Checking accuracy on validation set
Got 654 / 1000 correct (65.40)

Epoch 3, Iteration 700, loss = 0.5758
Checking accuracy on validation set
Got 649 / 1000 correct (64.90)

Epoch 4, Iteration 0, loss = 0.3022
Checking accuracy on validation set
Got 658 / 1000 correct (65.80)

Epoch 4, Iteration 100, loss = 0.4746
Checking accuracy on validation set
Got 641 / 1000 correct (64.10)

Epoch 4, Iteration 200, loss = 0.4152
Checking accuracy on validation set
Got 643 / 1000 correct (64.30)

Epoch 4, Iteration 300, loss = 0.3042
Checking accuracy on validation set
Got 653 / 1000 correct (65.30)

Epoch 4, Iteration 400, loss = 0.4475
Checking accuracy on validation set
Got 654 / 1000 correct (65.40)

Epoch 4, Iteration 500, loss = 0.6630
Checking accuracy on validation set
Got 631 / 1000 correct (63.10)

Epoch 4, Iteration 600, loss = 0.4523
Checking accuracy on validation set
Got 647 / 1000 correct (64.70)

Epoch 4, Iteration 700, loss = 0.5018
Checking accuracy on validation set
Got 639 / 1000 correct (63.90)
```

```
Epoch 5, Iteration 0, loss = 0.2580
Checking accuracy on validation set
Got 637 / 1000 correct (63.70)

Epoch 5, Iteration 100, loss = 0.1826
Checking accuracy on validation set
Got 639 / 1000 correct (63.90)

Epoch 5, Iteration 200, loss = 0.3418
Checking accuracy on validation set
Got 648 / 1000 correct (64.80)

Epoch 5, Iteration 300, loss = 0.2035
Checking accuracy on validation set
Got 658 / 1000 correct (65.80)

Epoch 5, Iteration 400, loss = 0.3296
Checking accuracy on validation set
Got 659 / 1000 correct (65.90)

Epoch 5, Iteration 500, loss = 0.2472
Checking accuracy on validation set
Got 653 / 1000 correct (65.30)

Epoch 5, Iteration 600, loss = 0.4025
Checking accuracy on validation set
Got 643 / 1000 correct (64.30)

Epoch 5, Iteration 700, loss = 0.3957
Checking accuracy on validation set
Got 625 / 1000 correct (62.50)

Epoch 6, Iteration 0, loss = 0.2108
Checking accuracy on validation set
Got 629 / 1000 correct (62.90)

Epoch 6, Iteration 100, loss = 0.1537
Checking accuracy on validation set
Got 649 / 1000 correct (64.90)

Epoch 6, Iteration 200, loss = 0.1867
Checking accuracy on validation set
Got 653 / 1000 correct (65.30)

Epoch 6, Iteration 300, loss = 0.1373
Checking accuracy on validation set
Got 649 / 1000 correct (64.90)
```

```
Epoch 6, Iteration 400, loss = 0.1010
Checking accuracy on validation set
Got 651 / 1000 correct (65.10)

Epoch 6, Iteration 500, loss = 0.1273
Checking accuracy on validation set
Got 635 / 1000 correct (63.50)

Epoch 6, Iteration 600, loss = 0.2396
Checking accuracy on validation set
Got 626 / 1000 correct (62.60)

Epoch 6, Iteration 700, loss = 0.3079
Checking accuracy on validation set
Got 623 / 1000 correct (62.30)

Epoch 7, Iteration 0, loss = 0.0913
Checking accuracy on validation set
Got 627 / 1000 correct (62.70)

Epoch 7, Iteration 100, loss = 0.1135
Checking accuracy on validation set
Got 634 / 1000 correct (63.40)

Epoch 7, Iteration 200, loss = 0.0582
Checking accuracy on validation set
Got 650 / 1000 correct (65.00)

Epoch 7, Iteration 300, loss = 0.0939
Checking accuracy on validation set
Got 641 / 1000 correct (64.10)

Epoch 7, Iteration 400, loss = 0.0545
Checking accuracy on validation set
Got 641 / 1000 correct (64.10)

Epoch 7, Iteration 500, loss = 0.0855
Checking accuracy on validation set
Got 635 / 1000 correct (63.50)

Epoch 7, Iteration 600, loss = 0.3189
Checking accuracy on validation set
Got 627 / 1000 correct (62.70)

Epoch 7, Iteration 700, loss = 0.1798
Checking accuracy on validation set
Got 628 / 1000 correct (62.80)
```

```
Epoch 8, Iteration 0, loss = 0.1152
Checking accuracy on validation set
Got 628 / 1000 correct (62.80)

Epoch 8, Iteration 100, loss = 0.0234
Checking accuracy on validation set
Got 648 / 1000 correct (64.80)

Epoch 8, Iteration 200, loss = 0.0447
Checking accuracy on validation set
Got 643 / 1000 correct (64.30)

Epoch 8, Iteration 300, loss = 0.1082
Checking accuracy on validation set
Got 639 / 1000 correct (63.90)

Epoch 8, Iteration 400, loss = 0.0725
Checking accuracy on validation set
Got 621 / 1000 correct (62.10)

Epoch 8, Iteration 500, loss = 0.1085
Checking accuracy on validation set
Got 649 / 1000 correct (64.90)

Epoch 8, Iteration 600, loss = 0.0972
Checking accuracy on validation set
Got 624 / 1000 correct (62.40)

Epoch 8, Iteration 700, loss = 0.1193
Checking accuracy on validation set
Got 621 / 1000 correct (62.10)

Epoch 9, Iteration 0, loss = 0.0276
Checking accuracy on validation set
Got 642 / 1000 correct (64.20)

Epoch 9, Iteration 100, loss = 0.0286
Checking accuracy on validation set
Got 633 / 1000 correct (63.30)

Epoch 9, Iteration 200, loss = 0.0481
Checking accuracy on validation set
Got 651 / 1000 correct (65.10)

Epoch 9, Iteration 300, loss = 0.0481
Checking accuracy on validation set
Got 637 / 1000 correct (63.70)
```

```
Epoch 9, Iteration 400, loss = 0.0959
Checking accuracy on validation set
Got 625 / 1000 correct (62.50)

Epoch 9, Iteration 500, loss = 0.0729
Checking accuracy on validation set
Got 629 / 1000 correct (62.90)

Epoch 9, Iteration 600, loss = 0.0539
Checking accuracy on validation set
Got 632 / 1000 correct (63.20)

Epoch 9, Iteration 700, loss = 0.0993
Checking accuracy on validation set
Got 625 / 1000 correct (62.50)

Maximum accuracy attained:  65.9
```

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. *

[34]:
```
earlyAttention = model
check_accuracy(loader_test, earlyAttention)
```

```
Checking accuracy on test set
Got 6243 / 10000 correct (62.43)
```

[34]: 62.43

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

[35]:
```
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, 1, 1),
    nn.ReLU(),
    nn.Conv2d(channel_1, channel_2, 3, 1, 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 = 2.2879
Checking accuracy on validation set
Got 135 / 1000 correct (13.50)

Epoch 0, Iteration 100, loss = 1.6502
Checking accuracy on validation set
Got 411 / 1000 correct (41.10)

Epoch 0, Iteration 200, loss = 1.5258
Checking accuracy on validation set
Got 454 / 1000 correct (45.40)

Epoch 0, Iteration 300, loss = 1.6398
Checking accuracy on validation set
Got 470 / 1000 correct (47.00)

Epoch 0, Iteration 400, loss = 1.3946
Checking accuracy on validation set
Got 517 / 1000 correct (51.70)

Epoch 0, Iteration 500, loss = 1.4216
Checking accuracy on validation set
Got 533 / 1000 correct (53.30)

Epoch 0, Iteration 600, loss = 1.4931
Checking accuracy on validation set
Got 519 / 1000 correct (51.90)

Epoch 0, Iteration 700, loss = 1.3623
Checking accuracy on validation set
Got 538 / 1000 correct (53.80)

Epoch 1, Iteration 0, loss = 1.1168
Checking accuracy on validation set
Got 520 / 1000 correct (52.00)
```

```
Epoch 1, Iteration 100, loss = 1.2746
Checking accuracy on validation set
Got 548 / 1000 correct (54.80)

Epoch 1, Iteration 200, loss = 1.4179
Checking accuracy on validation set
Got 531 / 1000 correct (53.10)

Epoch 1, Iteration 300, loss = 1.1626
Checking accuracy on validation set
Got 550 / 1000 correct (55.00)

Epoch 1, Iteration 400, loss = 1.1285
Checking accuracy on validation set
Got 551 / 1000 correct (55.10)

Epoch 1, Iteration 500, loss = 0.9452
Checking accuracy on validation set
Got 570 / 1000 correct (57.00)

Epoch 1, Iteration 600, loss = 1.1317
Checking accuracy on validation set
Got 570 / 1000 correct (57.00)

Epoch 1, Iteration 700, loss = 1.1361
Checking accuracy on validation set
Got 569 / 1000 correct (56.90)

Epoch 2, Iteration 0, loss = 1.0192
Checking accuracy on validation set
Got 589 / 1000 correct (58.90)

Epoch 2, Iteration 100, loss = 0.6741
Checking accuracy on validation set
Got 586 / 1000 correct (58.60)

Epoch 2, Iteration 200, loss = 0.9418
Checking accuracy on validation set
Got 588 / 1000 correct (58.80)

Epoch 2, Iteration 300, loss = 0.7472
Checking accuracy on validation set
Got 592 / 1000 correct (59.20)

Epoch 2, Iteration 400, loss = 1.0388
Checking accuracy on validation set
Got 590 / 1000 correct (59.00)
```

```
Epoch 2, Iteration 500, loss = 0.8401
Checking accuracy on validation set
Got 613 / 1000 correct (61.30)

Epoch 2, Iteration 600, loss = 0.9649
Checking accuracy on validation set
Got 597 / 1000 correct (59.70)

Epoch 2, Iteration 700, loss = 0.8622
Checking accuracy on validation set
Got 625 / 1000 correct (62.50)

Epoch 3, Iteration 0, loss = 0.7525
Checking accuracy on validation set
Got 598 / 1000 correct (59.80)

Epoch 3, Iteration 100, loss = 0.9080
Checking accuracy on validation set
Got 582 / 1000 correct (58.20)

Epoch 3, Iteration 200, loss = 0.7781
Checking accuracy on validation set
Got 591 / 1000 correct (59.10)

Epoch 3, Iteration 300, loss = 0.7300
Checking accuracy on validation set
Got 597 / 1000 correct (59.70)

Epoch 3, Iteration 400, loss = 0.8486
Checking accuracy on validation set
Got 611 / 1000 correct (61.10)

Epoch 3, Iteration 500, loss = 0.7340
Checking accuracy on validation set
Got 604 / 1000 correct (60.40)

Epoch 3, Iteration 600, loss = 0.8991
Checking accuracy on validation set
Got 612 / 1000 correct (61.20)

Epoch 3, Iteration 700, loss = 0.8386
Checking accuracy on validation set
Got 593 / 1000 correct (59.30)

Epoch 4, Iteration 0, loss = 0.5484
Checking accuracy on validation set
Got 614 / 1000 correct (61.40)
```

```
Epoch 4, Iteration 100, loss = 0.6374
Checking accuracy on validation set
Got 616 / 1000 correct (61.60)

Epoch 4, Iteration 200, loss = 0.9365
Checking accuracy on validation set
Got 633 / 1000 correct (63.30)

Epoch 4, Iteration 300, loss = 0.6951
Checking accuracy on validation set
Got 605 / 1000 correct (60.50)

Epoch 4, Iteration 400, loss = 0.6054
Checking accuracy on validation set
Got 612 / 1000 correct (61.20)

Epoch 4, Iteration 500, loss = 0.7699
Checking accuracy on validation set
Got 635 / 1000 correct (63.50)

Epoch 4, Iteration 600, loss = 0.7884
Checking accuracy on validation set
Got 630 / 1000 correct (63.00)

Epoch 4, Iteration 700, loss = 1.0280
Checking accuracy on validation set
Got 624 / 1000 correct (62.40)

Epoch 5, Iteration 0, loss = 0.7358
Checking accuracy on validation set
Got 622 / 1000 correct (62.20)

Epoch 5, Iteration 100, loss = 0.7291
Checking accuracy on validation set
Got 621 / 1000 correct (62.10)

Epoch 5, Iteration 200, loss = 0.5758
Checking accuracy on validation set
Got 620 / 1000 correct (62.00)

Epoch 5, Iteration 300, loss = 0.8560
Checking accuracy on validation set
Got 631 / 1000 correct (63.10)

Epoch 5, Iteration 400, loss = 0.7297
Checking accuracy on validation set
Got 593 / 1000 correct (59.30)
```

```
Epoch 5, Iteration 500, loss = 0.7628
Checking accuracy on validation set
Got 635 / 1000 correct (63.50)

Epoch 5, Iteration 600, loss = 1.0010
Checking accuracy on validation set
Got 626 / 1000 correct (62.60)

Epoch 5, Iteration 700, loss = 0.5614
Checking accuracy on validation set
Got 609 / 1000 correct (60.90)

Epoch 6, Iteration 0, loss = 0.5003
Checking accuracy on validation set
Got 631 / 1000 correct (63.10)

Epoch 6, Iteration 100, loss = 0.6355
Checking accuracy on validation set
Got 639 / 1000 correct (63.90)

Epoch 6, Iteration 200, loss = 0.4645
Checking accuracy on validation set
Got 620 / 1000 correct (62.00)

Epoch 6, Iteration 300, loss = 0.6156
Checking accuracy on validation set
Got 615 / 1000 correct (61.50)

Epoch 6, Iteration 400, loss = 0.5222
Checking accuracy on validation set
Got 612 / 1000 correct (61.20)

Epoch 6, Iteration 500, loss = 0.7358
Checking accuracy on validation set
Got 616 / 1000 correct (61.60)

Epoch 6, Iteration 600, loss = 0.7074
Checking accuracy on validation set
Got 616 / 1000 correct (61.60)

Epoch 6, Iteration 700, loss = 0.6950
Checking accuracy on validation set
Got 612 / 1000 correct (61.20)

Epoch 7, Iteration 0, loss = 0.4905
Checking accuracy on validation set
Got 618 / 1000 correct (61.80)
```

```
Epoch 7, Iteration 100, loss = 0.3307
Checking accuracy on validation set
Got 622 / 1000 correct (62.20)

Epoch 7, Iteration 200, loss = 0.8085
Checking accuracy on validation set
Got 602 / 1000 correct (60.20)

Epoch 7, Iteration 300, loss = 0.6208
Checking accuracy on validation set
Got 615 / 1000 correct (61.50)

Epoch 7, Iteration 400, loss = 0.7416
Checking accuracy on validation set
Got 608 / 1000 correct (60.80)

Epoch 7, Iteration 500, loss = 0.6164
Checking accuracy on validation set
Got 612 / 1000 correct (61.20)

Epoch 7, Iteration 600, loss = 0.4581
Checking accuracy on validation set
Got 614 / 1000 correct (61.40)

Epoch 7, Iteration 700, loss = 0.6923
Checking accuracy on validation set
Got 605 / 1000 correct (60.50)

Epoch 8, Iteration 0, loss = 0.2985
Checking accuracy on validation set
Got 605 / 1000 correct (60.50)

Epoch 8, Iteration 100, loss = 0.5146
Checking accuracy on validation set
Got 599 / 1000 correct (59.90)

Epoch 8, Iteration 200, loss = 0.3899
Checking accuracy on validation set
Got 609 / 1000 correct (60.90)

Epoch 8, Iteration 300, loss = 0.2925
Checking accuracy on validation set
Got 622 / 1000 correct (62.20)

Epoch 8, Iteration 400, loss = 0.5763
Checking accuracy on validation set
Got 601 / 1000 correct (60.10)
```

```
Epoch 8, Iteration 500, loss = 0.5136
Checking accuracy on validation set
Got 616 / 1000 correct (61.60)

Epoch 8, Iteration 600, loss = 0.5568
Checking accuracy on validation set
Got 612 / 1000 correct (61.20)

Epoch 8, Iteration 700, loss = 0.5961
Checking accuracy on validation set
Got 594 / 1000 correct (59.40)

Epoch 9, Iteration 0, loss = 0.4616
Checking accuracy on validation set
Got 590 / 1000 correct (59.00)

Epoch 9, Iteration 100, loss = 0.4840
Checking accuracy on validation set
Got 618 / 1000 correct (61.80)

Epoch 9, Iteration 200, loss = 0.2871
Checking accuracy on validation set
Got 620 / 1000 correct (62.00)

Epoch 9, Iteration 300, loss = 0.4059
Checking accuracy on validation set
Got 596 / 1000 correct (59.60)

Epoch 9, Iteration 400, loss = 0.3399
Checking accuracy on validation set
Got 585 / 1000 correct (58.50)

Epoch 9, Iteration 500, loss = 0.3672
Checking accuracy on validation set
Got 610 / 1000 correct (61.00)

Epoch 9, Iteration 600, loss = 0.4097
Checking accuracy on validation set
Got 606 / 1000 correct (60.60)

Epoch 9, Iteration 700, loss = 0.3299
Checking accuracy on validation set
Got 600 / 1000 correct (60.00)

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.

[36]:
```
lateAttention = model
check_accuracy(loader_test, lateAttention)
```

```
Checking accuracy on test set
Got 5974 / 10000 correct (59.74)
```

[36]: 59.74

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:

Self Attention is used for attend to attend to different spatial locations of the same image, enabling the model to attend to the related regions and their interactions. This is also called spatial or intra atttention. One example use of self attention is Image Segmentation where self-attention can be used to capture long range dependencies between different pixels in the image. By attending to individual pixels and gathering information from their neighboring/ related pixels, the model can generate more precise predictions. This application of self-attention aids in accurately delineating object boundaries and enhancing the overall quality of the segmentation results.

On the other hand, Attention is used to attend to the information from one modality and pass on that information to attend to another modality. It allows the model to attend to relevant parts of one modality based on the information from another modality. For example, in image captioning with textual context, attention can be utilized to establish alignment between significant image regions and corresponding words in the accompanying text. The model selectively focuses on specific regions of the image, taking into account the textual context. This allows the model to generate captions that are more contextually relevant and appropriate.

To Differentiate, self-attention in computer vision emphasizes spatial relationships within a single input, enabling the model to concentrate on various regions and their interactions. In contrast, attention is employed to capture connections between distinct modalities, empowering the model to selectively attend to pertinent components of one modality by leveraging information from another modality.

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

[37]:
```
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, 1 ,1),
    nn.ReLU(),
    Attention(channel_1),
    nn.ReLU(),
    nn.Conv2d(channel_1, channel_2, 3, 1 ,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 = 2.3017
Checking accuracy on validation set
Got 133 / 1000 correct (13.30)

Epoch 0, Iteration 100, loss = 1.9606
Checking accuracy on validation set
Got 314 / 1000 correct (31.40)

Epoch 0, Iteration 200, loss = 1.5850
Checking accuracy on validation set
Got 419 / 1000 correct (41.90)

Epoch 0, Iteration 300, loss = 1.3390
Checking accuracy on validation set
Got 424 / 1000 correct (42.40)

Epoch 0, Iteration 400, loss = 1.5136
Checking accuracy on validation set
Got 484 / 1000 correct (48.40)

Epoch 0, Iteration 500, loss = 1.2485
Checking accuracy on validation set
Got 505 / 1000 correct (50.50)

Epoch 0, Iteration 600, loss = 1.3849
Checking accuracy on validation set
Got 510 / 1000 correct (51.00)

Epoch 0, Iteration 700, loss = 1.6672
Checking accuracy on validation set
```

```
Got 536 / 1000 correct (53.60)

Epoch 1, Iteration 0, loss = 1.2340
Checking accuracy on validation set
Got 506 / 1000 correct (50.60)

Epoch 1, Iteration 100, loss = 1.3478
Checking accuracy on validation set
Got 483 / 1000 correct (48.30)

Epoch 1, Iteration 200, loss = 1.3578
Checking accuracy on validation set
Got 538 / 1000 correct (53.80)

Epoch 1, Iteration 300, loss = 1.5439
Checking accuracy on validation set
Got 529 / 1000 correct (52.90)

Epoch 1, Iteration 400, loss = 1.1949
Checking accuracy on validation set
Got 568 / 1000 correct (56.80)

Epoch 1, Iteration 500, loss = 1.3000
Checking accuracy on validation set
Got 564 / 1000 correct (56.40)

Epoch 1, Iteration 600, loss = 1.2228
Checking accuracy on validation set
Got 591 / 1000 correct (59.10)

Epoch 1, Iteration 700, loss = 1.1319
Checking accuracy on validation set
Got 582 / 1000 correct (58.20)

Epoch 2, Iteration 0, loss = 0.8901
Checking accuracy on validation set
Got 608 / 1000 correct (60.80)

Epoch 2, Iteration 100, loss = 1.0949
Checking accuracy on validation set
Got 613 / 1000 correct (61.30)

Epoch 2, Iteration 200, loss = 0.9834
Checking accuracy on validation set
Got 607 / 1000 correct (60.70)

Epoch 2, Iteration 300, loss = 0.9510
Checking accuracy on validation set
```

```
Got 611 / 1000 correct (61.10)

Epoch 2, Iteration 400, loss = 0.9090
Checking accuracy on validation set
Got 600 / 1000 correct (60.00)

Epoch 2, Iteration 500, loss = 1.0913
Checking accuracy on validation set
Got 645 / 1000 correct (64.50)

Epoch 2, Iteration 600, loss = 0.9581
Checking accuracy on validation set
Got 627 / 1000 correct (62.70)

Epoch 2, Iteration 700, loss = 0.8788
Checking accuracy on validation set
Got 661 / 1000 correct (66.10)

Epoch 3, Iteration 0, loss = 0.8825
Checking accuracy on validation set
Got 623 / 1000 correct (62.30)

Epoch 3, Iteration 100, loss = 0.9478
Checking accuracy on validation set
Got 623 / 1000 correct (62.30)

Epoch 3, Iteration 200, loss = 0.7360
Checking accuracy on validation set
Got 617 / 1000 correct (61.70)

Epoch 3, Iteration 300, loss = 0.6984
Checking accuracy on validation set
Got 623 / 1000 correct (62.30)

Epoch 3, Iteration 400, loss = 0.8664
Checking accuracy on validation set
Got 650 / 1000 correct (65.00)

Epoch 3, Iteration 500, loss = 0.6035
Checking accuracy on validation set
Got 634 / 1000 correct (63.40)

Epoch 3, Iteration 600, loss = 0.8371
Checking accuracy on validation set
Got 651 / 1000 correct (65.10)

Epoch 3, Iteration 700, loss = 0.8165
Checking accuracy on validation set
```

```
Got 647 / 1000 correct (64.70)

Epoch 4, Iteration 0, loss = 0.6053
Checking accuracy on validation set
Got 647 / 1000 correct (64.70)

Epoch 4, Iteration 100, loss = 0.5280
Checking accuracy on validation set
Got 639 / 1000 correct (63.90)

Epoch 4, Iteration 200, loss = 0.7052
Checking accuracy on validation set
Got 642 / 1000 correct (64.20)

Epoch 4, Iteration 300, loss = 0.5191
Checking accuracy on validation set
Got 636 / 1000 correct (63.60)

Epoch 4, Iteration 400, loss = 0.6710
Checking accuracy on validation set
Got 626 / 1000 correct (62.60)

Epoch 4, Iteration 500, loss = 0.5947
Checking accuracy on validation set
Got 635 / 1000 correct (63.50)

Epoch 4, Iteration 600, loss = 0.7311
Checking accuracy on validation set
Got 630 / 1000 correct (63.00)

Epoch 4, Iteration 700, loss = 0.6020
Checking accuracy on validation set
Got 640 / 1000 correct (64.00)

Epoch 5, Iteration 0, loss = 0.5500
Checking accuracy on validation set
Got 627 / 1000 correct (62.70)

Epoch 5, Iteration 100, loss = 0.4306
Checking accuracy on validation set
Got 628 / 1000 correct (62.80)

Epoch 5, Iteration 200, loss = 0.6201
Checking accuracy on validation set
Got 645 / 1000 correct (64.50)

Epoch 5, Iteration 300, loss = 0.4431
Checking accuracy on validation set
```

```
Got 640 / 1000 correct (64.00)

Epoch 5, Iteration 400, loss = 0.3817
Checking accuracy on validation set
Got 637 / 1000 correct (63.70)

Epoch 5, Iteration 500, loss = 0.3450
Checking accuracy on validation set
Got 622 / 1000 correct (62.20)

Epoch 5, Iteration 600, loss = 0.6369
Checking accuracy on validation set
Got 646 / 1000 correct (64.60)

Epoch 5, Iteration 700, loss = 0.6463
Checking accuracy on validation set
Got 641 / 1000 correct (64.10)

Epoch 6, Iteration 0, loss = 0.3517
Checking accuracy on validation set
Got 643 / 1000 correct (64.30)

Epoch 6, Iteration 100, loss = 0.2337
Checking accuracy on validation set
Got 624 / 1000 correct (62.40)

Epoch 6, Iteration 200, loss = 0.4153
Checking accuracy on validation set
Got 627 / 1000 correct (62.70)

Epoch 6, Iteration 300, loss = 0.4165
Checking accuracy on validation set
Got 647 / 1000 correct (64.70)

Epoch 6, Iteration 400, loss = 0.4186
Checking accuracy on validation set
Got 636 / 1000 correct (63.60)

Epoch 6, Iteration 500, loss = 0.5516
Checking accuracy on validation set
Got 639 / 1000 correct (63.90)

Epoch 6, Iteration 600, loss = 0.2363
Checking accuracy on validation set
Got 639 / 1000 correct (63.90)

Epoch 6, Iteration 700, loss = 0.6781
Checking accuracy on validation set
```

```
Got 624 / 1000 correct (62.40)

Epoch 7, Iteration 0, loss = 0.2058
Checking accuracy on validation set
Got 639 / 1000 correct (63.90)

Epoch 7, Iteration 100, loss = 0.3069
Checking accuracy on validation set
Got 648 / 1000 correct (64.80)

Epoch 7, Iteration 200, loss = 0.2424
Checking accuracy on validation set
Got 641 / 1000 correct (64.10)

Epoch 7, Iteration 300, loss = 0.3960
Checking accuracy on validation set
Got 626 / 1000 correct (62.60)

Epoch 7, Iteration 400, loss = 0.2174
Checking accuracy on validation set
Got 629 / 1000 correct (62.90)

Epoch 7, Iteration 500, loss = 0.2432
Checking accuracy on validation set
Got 629 / 1000 correct (62.90)

Epoch 7, Iteration 600, loss = 0.6479
Checking accuracy on validation set
Got 637 / 1000 correct (63.70)

Epoch 7, Iteration 700, loss = 0.4796
Checking accuracy on validation set
Got 636 / 1000 correct (63.60)

Epoch 8, Iteration 0, loss = 0.1350
Checking accuracy on validation set
Got 616 / 1000 correct (61.60)

Epoch 8, Iteration 100, loss = 0.1545
Checking accuracy on validation set
Got 634 / 1000 correct (63.40)

Epoch 8, Iteration 200, loss = 0.2062
Checking accuracy on validation set
Got 637 / 1000 correct (63.70)

Epoch 8, Iteration 300, loss = 0.1565
Checking accuracy on validation set
```

```
Got 646 / 1000 correct (64.60)

Epoch 8, Iteration 400, loss = 0.2364
Checking accuracy on validation set
Got 636 / 1000 correct (63.60)

Epoch 8, Iteration 500, loss = 0.2539
Checking accuracy on validation set
Got 634 / 1000 correct (63.40)

Epoch 8, Iteration 600, loss = 0.3119
Checking accuracy on validation set
Got 623 / 1000 correct (62.30)

Epoch 8, Iteration 700, loss = 0.2701
Checking accuracy on validation set
Got 624 / 1000 correct (62.40)

Epoch 9, Iteration 0, loss = 0.2171
Checking accuracy on validation set
Got 621 / 1000 correct (62.10)

Epoch 9, Iteration 100, loss = 0.1329
Checking accuracy on validation set
Got 647 / 1000 correct (64.70)

Epoch 9, Iteration 200, loss = 0.1171
Checking accuracy on validation set
Got 636 / 1000 correct (63.60)

Epoch 9, Iteration 300, loss = 0.1061
Checking accuracy on validation set
Got 624 / 1000 correct (62.40)

Epoch 9, Iteration 400, loss = 0.1479
Checking accuracy on validation set
Got 619 / 1000 correct (61.90)

Epoch 9, Iteration 500, loss = 0.1091
Checking accuracy on validation set
Got 630 / 1000 correct (63.00)

Epoch 9, Iteration 600, loss = 0.2144
Checking accuracy on validation set
Got 627 / 1000 correct (62.70)

Epoch 9, Iteration 700, loss = 0.2449
Checking accuracy on validation set
```

```
Got 621 / 1000 correct (62.10)

Maximum accuracy attained:  66.10000000000001
```

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.

[38]:
```
vanillaModel = model
check_accuracy(loader_test, vanillaModel)
```

```
Checking accuracy on test set
Got 5997 / 10000 correct (59.97)
```

[38]:
```
59.97
```

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.

[46]:
```
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.

[47]:
```
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 = 4.5005
Checking accuracy on validation set
Got 119 / 1000 correct (11.90)

Epoch 0, Iteration 100, loss = 1.5761
Checking accuracy on validation set
Got 396 / 1000 correct (39.60)

Epoch 0, Iteration 200, loss = 1.4433
Checking accuracy on validation set
Got 406 / 1000 correct (40.60)

Epoch 0, Iteration 300, loss = 1.2680
Checking accuracy on validation set
Got 455 / 1000 correct (45.50)

Epoch 0, Iteration 400, loss = 0.9880
Checking accuracy on validation set
Got 500 / 1000 correct (50.00)

Epoch 0, Iteration 500, loss = 1.3753
Checking accuracy on validation set
Got 485 / 1000 correct (48.50)

Epoch 0, Iteration 600, loss = 0.9280
Checking accuracy on validation set
Got 567 / 1000 correct (56.70)

Epoch 0, Iteration 700, loss = 1.1729
Checking accuracy on validation set
Got 523 / 1000 correct (52.30)

Epoch 1, Iteration 0, loss = 0.9300
Checking accuracy on validation set
Got 577 / 1000 correct (57.70)
```

```
Epoch 1, Iteration 100, loss = 1.1176
Checking accuracy on validation set
Got 581 / 1000 correct (58.10)

Epoch 1, Iteration 200, loss = 1.0988
Checking accuracy on validation set
Got 596 / 1000 correct (59.60)

Epoch 1, Iteration 300, loss = 0.9804
Checking accuracy on validation set
Got 620 / 1000 correct (62.00)

Epoch 1, Iteration 400, loss = 1.1027
Checking accuracy on validation set
Got 623 / 1000 correct (62.30)

Epoch 1, Iteration 500, loss = 0.9551
Checking accuracy on validation set
Got 596 / 1000 correct (59.60)

Epoch 1, Iteration 600, loss = 0.9800
Checking accuracy on validation set
Got 619 / 1000 correct (61.90)

Epoch 1, Iteration 700, loss = 1.0259
Checking accuracy on validation set
Got 669 / 1000 correct (66.90)

Epoch 2, Iteration 0, loss = 0.7928
Checking accuracy on validation set
Got 643 / 1000 correct (64.30)

Epoch 2, Iteration 100, loss = 0.8086
Checking accuracy on validation set
Got 597 / 1000 correct (59.70)

Epoch 2, Iteration 200, loss = 0.7073
Checking accuracy on validation set
Got 667 / 1000 correct (66.70)

Epoch 2, Iteration 300, loss = 0.9120
Checking accuracy on validation set
Got 664 / 1000 correct (66.40)

Epoch 2, Iteration 400, loss = 1.0415
Checking accuracy on validation set
Got 670 / 1000 correct (67.00)
```

```
Epoch 2, Iteration 500, loss = 0.9040
Checking accuracy on validation set
Got 655 / 1000 correct (65.50)

Epoch 2, Iteration 600, loss = 1.0012
Checking accuracy on validation set
Got 644 / 1000 correct (64.40)

Epoch 2, Iteration 700, loss = 0.7503
Checking accuracy on validation set
Got 678 / 1000 correct (67.80)

Epoch 3, Iteration 0, loss = 0.6246
Checking accuracy on validation set
Got 660 / 1000 correct (66.00)

Epoch 3, Iteration 100, loss = 0.9770
Checking accuracy on validation set
Got 722 / 1000 correct (72.20)

Epoch 3, Iteration 200, loss = 0.8190
Checking accuracy on validation set
Got 633 / 1000 correct (63.30)

Epoch 3, Iteration 300, loss = 0.9217
Checking accuracy on validation set
Got 717 / 1000 correct (71.70)

Epoch 3, Iteration 400, loss = 0.6896
Checking accuracy on validation set
Got 716 / 1000 correct (71.60)

Epoch 3, Iteration 500, loss = 0.8062
Checking accuracy on validation set
Got 699 / 1000 correct (69.90)

Epoch 3, Iteration 600, loss = 0.7674
Checking accuracy on validation set
Got 696 / 1000 correct (69.60)

Epoch 3, Iteration 700, loss = 0.8552
Checking accuracy on validation set
Got 714 / 1000 correct (71.40)

Epoch 4, Iteration 0, loss = 0.6897
Checking accuracy on validation set
Got 716 / 1000 correct (71.60)
```

```
Epoch 4, Iteration 100, loss = 0.5356
Checking accuracy on validation set
Got 712 / 1000 correct (71.20)

Epoch 4, Iteration 200, loss = 0.6174
Checking accuracy on validation set
Got 728 / 1000 correct (72.80)

Epoch 4, Iteration 300, loss = 0.7274
Checking accuracy on validation set
Got 697 / 1000 correct (69.70)

Epoch 4, Iteration 400, loss = 0.9215
Checking accuracy on validation set
Got 724 / 1000 correct (72.40)

Epoch 4, Iteration 500, loss = 0.5413
Checking accuracy on validation set
Got 715 / 1000 correct (71.50)

Epoch 4, Iteration 600, loss = 0.5304
Checking accuracy on validation set
Got 741 / 1000 correct (74.10)

Epoch 4, Iteration 700, loss = 0.9532
Checking accuracy on validation set
Got 731 / 1000 correct (73.10)

Epoch 5, Iteration 0, loss = 0.7480
Checking accuracy on validation set
Got 724 / 1000 correct (72.40)

Epoch 5, Iteration 100, loss = 0.5889
Checking accuracy on validation set
Got 737 / 1000 correct (73.70)

Epoch 5, Iteration 200, loss = 0.8247
Checking accuracy on validation set
Got 738 / 1000 correct (73.80)

Epoch 5, Iteration 300, loss = 0.3807
Checking accuracy on validation set
Got 738 / 1000 correct (73.80)

Epoch 5, Iteration 400, loss = 0.7233
Checking accuracy on validation set
Got 744 / 1000 correct (74.40)
```

```
Epoch 5, Iteration 500, loss = 0.5695
Checking accuracy on validation set
Got 718 / 1000 correct (71.80)

Epoch 5, Iteration 600, loss = 0.7948
Checking accuracy on validation set
Got 718 / 1000 correct (71.80)

Epoch 5, Iteration 700, loss = 0.6396
Checking accuracy on validation set
Got 735 / 1000 correct (73.50)

Epoch 6, Iteration 0, loss = 0.7415
Checking accuracy on validation set
Got 734 / 1000 correct (73.40)

Epoch 6, Iteration 100, loss = 0.5042
Checking accuracy on validation set
Got 741 / 1000 correct (74.10)

Epoch 6, Iteration 200, loss = 0.5784
Checking accuracy on validation set
Got 749 / 1000 correct (74.90)

Epoch 6, Iteration 300, loss = 0.6435
Checking accuracy on validation set
Got 724 / 1000 correct (72.40)

Epoch 6, Iteration 400, loss = 0.5283
Checking accuracy on validation set
Got 745 / 1000 correct (74.50)

Epoch 6, Iteration 500, loss = 0.6991
Checking accuracy on validation set
Got 739 / 1000 correct (73.90)

Epoch 6, Iteration 600, loss = 0.4693
Checking accuracy on validation set
Got 727 / 1000 correct (72.70)

Epoch 6, Iteration 700, loss = 0.5063
Checking accuracy on validation set
Got 732 / 1000 correct (73.20)

Epoch 7, Iteration 0, loss = 0.5561
Checking accuracy on validation set
Got 749 / 1000 correct (74.90)
```

```
Epoch 7, Iteration 100, loss = 0.5698
Checking accuracy on validation set
Got 752 / 1000 correct (75.20)

Epoch 7, Iteration 200, loss = 0.9448
Checking accuracy on validation set
Got 751 / 1000 correct (75.10)

Epoch 7, Iteration 300, loss = 0.5782
Checking accuracy on validation set
Got 747 / 1000 correct (74.70)

Epoch 7, Iteration 400, loss = 0.5310
Checking accuracy on validation set
Got 734 / 1000 correct (73.40)

Epoch 7, Iteration 500, loss = 0.5911
Checking accuracy on validation set
Got 740 / 1000 correct (74.00)

Epoch 7, Iteration 600, loss = 0.5704
Checking accuracy on validation set
Got 754 / 1000 correct (75.40)

Epoch 7, Iteration 700, loss = 0.5725
Checking accuracy on validation set
Got 754 / 1000 correct (75.40)

Epoch 8, Iteration 0, loss = 0.4119
Checking accuracy on validation set
Got 745 / 1000 correct (74.50)

Epoch 8, Iteration 100, loss = 0.3801
Checking accuracy on validation set
Got 764 / 1000 correct (76.40)

Epoch 8, Iteration 200, loss = 0.5309
Checking accuracy on validation set
Got 753 / 1000 correct (75.30)

Epoch 8, Iteration 300, loss = 0.4212
Checking accuracy on validation set
Got 741 / 1000 correct (74.10)

Epoch 8, Iteration 400, loss = 0.4169
Checking accuracy on validation set
Got 739 / 1000 correct (73.90)
```

```
Epoch 8, Iteration 500, loss = 0.3725
Checking accuracy on validation set
Got 751 / 1000 correct (75.10)

Epoch 8, Iteration 600, loss = 0.5556
Checking accuracy on validation set
Got 737 / 1000 correct (73.70)

Epoch 8, Iteration 700, loss = 0.3776
Checking accuracy on validation set
Got 760 / 1000 correct (76.00)

Epoch 9, Iteration 0, loss = 0.4804
Checking accuracy on validation set
Got 745 / 1000 correct (74.50)

Epoch 9, Iteration 100, loss = 0.3719
Checking accuracy on validation set
Got 769 / 1000 correct (76.90)

Epoch 9, Iteration 200, loss = 0.4680
Checking accuracy on validation set
Got 766 / 1000 correct (76.60)

Epoch 9, Iteration 300, loss = 0.4127
Checking accuracy on validation set
Got 758 / 1000 correct (75.80)

Epoch 9, Iteration 400, loss = 0.3090
Checking accuracy on validation set
Got 747 / 1000 correct (74.70)

Epoch 9, Iteration 500, loss = 0.1964
Checking accuracy on validation set
Got 758 / 1000 correct (75.80)

Epoch 9, Iteration 600, loss = 0.3004
Checking accuracy on validation set
Got 742 / 1000 correct (74.20)

Epoch 9, Iteration 700, loss = 0.5556
Checking accuracy on validation set
Got 753 / 1000 correct (75.30)

Maximum accuracy attained:  76.9
Checking accuracy on test set
Got 7463 / 10000 correct (74.63)
```

[47]: 74.63

[48]:
```
Resnet = model
check_accuracy(loader_test, Resnet)
```

```
Checking accuracy on test set
Got 7463 / 10000 correct (74.63)
```

[48]: 74.63

2.14 Resnet with Attention (5 points)

[42]:
```
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)

        self.attention = Attention(128)


    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 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)
```

[43]:
```
## Resnet with Attention
learning_rate = 1e-3

# TODO: Use the above Attention module after the 2nd resnet block i.e. after␣
 ↪self.layer2.

model = ResNet10()

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

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

```
Epoch 0, Iteration 0, loss = 4.6657
Checking accuracy on validation set
Got 119 / 1000 correct (11.90)
```

```
Epoch 0, Iteration 100, loss = 1.4973
Checking accuracy on validation set
Got 401 / 1000 correct (40.10)

Epoch 0, Iteration 200, loss = 1.6365
Checking accuracy on validation set
Got 419 / 1000 correct (41.90)

Epoch 0, Iteration 300, loss = 1.2889
Checking accuracy on validation set
Got 507 / 1000 correct (50.70)

Epoch 0, Iteration 400, loss = 1.3445
Checking accuracy on validation set
Got 506 / 1000 correct (50.60)

Epoch 0, Iteration 500, loss = 1.1996
Checking accuracy on validation set
Got 562 / 1000 correct (56.20)

Epoch 0, Iteration 600, loss = 1.1652
Checking accuracy on validation set
Got 595 / 1000 correct (59.50)

Epoch 0, Iteration 700, loss = 1.0385
Checking accuracy on validation set
Got 586 / 1000 correct (58.60)

Epoch 1, Iteration 0, loss = 0.8241
Checking accuracy on validation set
Got 584 / 1000 correct (58.40)

Epoch 1, Iteration 100, loss = 1.0126
Checking accuracy on validation set
Got 649 / 1000 correct (64.90)

Epoch 1, Iteration 200, loss = 0.9641
Checking accuracy on validation set
Got 652 / 1000 correct (65.20)

Epoch 1, Iteration 300, loss = 0.7813
Checking accuracy on validation set
Got 649 / 1000 correct (64.90)

Epoch 1, Iteration 400, loss = 0.9810
Checking accuracy on validation set
Got 652 / 1000 correct (65.20)
```

```
Epoch 1, Iteration 500, loss = 0.8898
Checking accuracy on validation set
Got 649 / 1000 correct (64.90)

Epoch 1, Iteration 600, loss = 0.9529
Checking accuracy on validation set
Got 699 / 1000 correct (69.90)

Epoch 1, Iteration 700, loss = 0.6324
Checking accuracy on validation set
Got 688 / 1000 correct (68.80)

Epoch 2, Iteration 0, loss = 0.7298
Checking accuracy on validation set
Got 692 / 1000 correct (69.20)

Epoch 2, Iteration 100, loss = 0.6861
Checking accuracy on validation set
Got 713 / 1000 correct (71.30)

Epoch 2, Iteration 200, loss = 0.9921
Checking accuracy on validation set
Got 693 / 1000 correct (69.30)

Epoch 2, Iteration 300, loss = 0.5949
Checking accuracy on validation set
Got 705 / 1000 correct (70.50)

Epoch 2, Iteration 400, loss = 0.8917
Checking accuracy on validation set
Got 716 / 1000 correct (71.60)

Epoch 2, Iteration 500, loss = 0.9522
Checking accuracy on validation set
Got 695 / 1000 correct (69.50)

Epoch 2, Iteration 600, loss = 0.6782
Checking accuracy on validation set
Got 717 / 1000 correct (71.70)

Epoch 2, Iteration 700, loss = 0.6623
Checking accuracy on validation set
Got 685 / 1000 correct (68.50)

Epoch 3, Iteration 0, loss = 0.5418
Checking accuracy on validation set
Got 711 / 1000 correct (71.10)
```

```
Epoch 3, Iteration 100, loss = 0.6220
Checking accuracy on validation set
Got 744 / 1000 correct (74.40)

Epoch 3, Iteration 200, loss = 0.6327
Checking accuracy on validation set
Got 719 / 1000 correct (71.90)

Epoch 3, Iteration 300, loss = 0.6662
Checking accuracy on validation set
Got 734 / 1000 correct (73.40)

Epoch 3, Iteration 400, loss = 0.5752
Checking accuracy on validation set
Got 729 / 1000 correct (72.90)

Epoch 3, Iteration 500, loss = 0.6846
Checking accuracy on validation set
Got 714 / 1000 correct (71.40)

Epoch 3, Iteration 600, loss = 0.5660
Checking accuracy on validation set
Got 735 / 1000 correct (73.50)

Epoch 3, Iteration 700, loss = 0.6668
Checking accuracy on validation set
Got 747 / 1000 correct (74.70)

Epoch 4, Iteration 0, loss = 0.5166
Checking accuracy on validation set
Got 756 / 1000 correct (75.60)

Epoch 4, Iteration 100, loss = 0.5784
Checking accuracy on validation set
Got 756 / 1000 correct (75.60)

Epoch 4, Iteration 200, loss = 0.4132
Checking accuracy on validation set
Got 748 / 1000 correct (74.80)

Epoch 4, Iteration 300, loss = 0.3618
Checking accuracy on validation set
Got 752 / 1000 correct (75.20)

Epoch 4, Iteration 400, loss = 0.4773
Checking accuracy on validation set
Got 769 / 1000 correct (76.90)
```

```
Epoch 4, Iteration 500, loss = 0.5796
Checking accuracy on validation set
Got 753 / 1000 correct (75.30)

Epoch 4, Iteration 600, loss = 0.5089
Checking accuracy on validation set
Got 750 / 1000 correct (75.00)

Epoch 4, Iteration 700, loss = 0.7006
Checking accuracy on validation set
Got 756 / 1000 correct (75.60)

Epoch 5, Iteration 0, loss = 0.4568
Checking accuracy on validation set
Got 755 / 1000 correct (75.50)

Epoch 5, Iteration 100, loss = 0.3074
Checking accuracy on validation set
Got 777 / 1000 correct (77.70)

Epoch 5, Iteration 200, loss = 0.3748
Checking accuracy on validation set
Got 776 / 1000 correct (77.60)

Epoch 5, Iteration 300, loss = 0.2326
Checking accuracy on validation set
Got 744 / 1000 correct (74.40)

Epoch 5, Iteration 400, loss = 0.2989
Checking accuracy on validation set
Got 764 / 1000 correct (76.40)

Epoch 5, Iteration 500, loss = 0.5952
Checking accuracy on validation set
Got 763 / 1000 correct (76.30)

Epoch 5, Iteration 600, loss = 0.6404
Checking accuracy on validation set
Got 770 / 1000 correct (77.00)

Epoch 5, Iteration 700, loss = 0.4440
Checking accuracy on validation set
Got 779 / 1000 correct (77.90)

Epoch 6, Iteration 0, loss = 0.2613
Checking accuracy on validation set
Got 778 / 1000 correct (77.80)
```

```
Epoch 6, Iteration 100, loss = 0.2353
Checking accuracy on validation set
Got 776 / 1000 correct (77.60)

Epoch 6, Iteration 200, loss = 0.2555
Checking accuracy on validation set
Got 758 / 1000 correct (75.80)

Epoch 6, Iteration 300, loss = 0.2340
Checking accuracy on validation set
Got 769 / 1000 correct (76.90)

Epoch 6, Iteration 400, loss = 0.2990
Checking accuracy on validation set
Got 777 / 1000 correct (77.70)

Epoch 6, Iteration 500, loss = 0.2496
Checking accuracy on validation set
Got 764 / 1000 correct (76.40)

Epoch 6, Iteration 600, loss = 0.3840
Checking accuracy on validation set
Got 772 / 1000 correct (77.20)

Epoch 6, Iteration 700, loss = 0.6152
Checking accuracy on validation set
Got 755 / 1000 correct (75.50)

Epoch 7, Iteration 0, loss = 0.2148
Checking accuracy on validation set
Got 779 / 1000 correct (77.90)

Epoch 7, Iteration 100, loss = 0.1413
Checking accuracy on validation set
Got 783 / 1000 correct (78.30)

Epoch 7, Iteration 200, loss = 0.1523
Checking accuracy on validation set
Got 789 / 1000 correct (78.90)

Epoch 7, Iteration 300, loss = 0.1511
Checking accuracy on validation set
Got 772 / 1000 correct (77.20)

Epoch 7, Iteration 400, loss = 0.1843
Checking accuracy on validation set
Got 769 / 1000 correct (76.90)
```

```
Epoch 7, Iteration 500, loss = 0.2356
Checking accuracy on validation set
Got 761 / 1000 correct (76.10)

Epoch 7, Iteration 600, loss = 0.3964
Checking accuracy on validation set
Got 756 / 1000 correct (75.60)

Epoch 7, Iteration 700, loss = 0.3712
Checking accuracy on validation set
Got 768 / 1000 correct (76.80)

Epoch 8, Iteration 0, loss = 0.3197
Checking accuracy on validation set
Got 743 / 1000 correct (74.30)

Epoch 8, Iteration 100, loss = 0.2343
Checking accuracy on validation set
Got 782 / 1000 correct (78.20)

Epoch 8, Iteration 200, loss = 0.1345
Checking accuracy on validation set
Got 778 / 1000 correct (77.80)

Epoch 8, Iteration 300, loss = 0.2005
Checking accuracy on validation set
Got 785 / 1000 correct (78.50)

Epoch 8, Iteration 400, loss = 0.2382
Checking accuracy on validation set
Got 772 / 1000 correct (77.20)

Epoch 8, Iteration 500, loss = 0.1123
Checking accuracy on validation set
Got 761 / 1000 correct (76.10)

Epoch 8, Iteration 600, loss = 0.1827
Checking accuracy on validation set
Got 766 / 1000 correct (76.60)

Epoch 8, Iteration 700, loss = 0.2097
Checking accuracy on validation set
Got 767 / 1000 correct (76.70)

Epoch 9, Iteration 0, loss = 0.1006
Checking accuracy on validation set
Got 763 / 1000 correct (76.30)
```

```
Epoch 9, Iteration 100, loss = 0.0827
Checking accuracy on validation set
Got 781 / 1000 correct (78.10)

Epoch 9, Iteration 200, loss = 0.1909
Checking accuracy on validation set
Got 779 / 1000 correct (77.90)

Epoch 9, Iteration 300, loss = 0.0390
Checking accuracy on validation set
Got 788 / 1000 correct (78.80)

Epoch 9, Iteration 400, loss = 0.0821
Checking accuracy on validation set
Got 759 / 1000 correct (75.90)

Epoch 9, Iteration 500, loss = 0.0380
Checking accuracy on validation set
Got 761 / 1000 correct (76.10)

Epoch 9, Iteration 600, loss = 0.2338
Checking accuracy on validation set
Got 765 / 1000 correct (76.50)

Epoch 9, Iteration 700, loss = 0.3154
Checking accuracy on validation set
Got 760 / 1000 correct (76.00)

Maximum accuracy attained:  78.9
```

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.

[45]:
```
AttentionResnet = model
check_accuracy(loader_test, AttentionResnet)
```

```
Checking accuracy on test set
Got 7391 / 10000 correct (73.91)
```

[45]: 73.91

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 (75.9%, 75.79%, 74.01)
2. Vanilla Resnet (74.63%, 75.98%, 74.94%)
3. Single Attention Block: Early attention; After the first conv layer (63.3%, 61.84%, 62.43%)
4. Double Attention Blocks (62.57%, 62.49%, 61.97%)
5. Single Attention Block: Late attention; After the second conv layer (59.08%, 59.89%, 59.74%)
6. Vanilla CNN (53.8%, 52.77%, 53.3%)

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

Your Answer: Attention helps in attending and understanding the different features in the model, thus gives a better rating using attention. Resnet gives a better prediction as they have a residual block that helps in improving the performance. As the number of attention block increases, the model performance doesn't have a greater improvement. The rankings can be explained based on the characteristics of each model. Resnet with Attention ranks first due to its combination of Resnet architecture, which addresses the vanishing gradient problem, and attention mechanisms, which allow the model to focus on important features. This combination results in improved performance. Vanilla Resnet comes in second as it utilizes residual blocks to enable deeper networks and learn complex representations. The single attention block with early attention ranks third, capturing important features at an early stage and providing a moderate performance boost. The double attention blocks follow, offering some improvement but not significantly more than a single attention block. The single attention block with late attention ranks fifth, capturing higher-level features but with a less pronounced impact on performance. Finally, the vanilla CNN without attention or residual connections ranks last, as it lacks the enhancements needed to capture complex patterns effectively. Overall, the rankings highlight the benefits of attention mechanisms and the importance of model architecture in achieving higher performance.