assignment4

May 22, 2022

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.

[1]:

```
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
```

[2]:

```
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.

[3]:

```
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

[4]:

```
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

[5]:
```
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

[6]:
```
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)
```

[7]:

```
# 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.

[12]:

```
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=10)
```

```
Epoch 0, Iteration 0, loss = 2.3160
Checking accuracy on validation set
Got 147 / 1000 correct (14.70)

Epoch 0, Iteration 100, loss = 1.7426
Checking accuracy on validation set
Got 411 / 1000 correct (41.10)

Epoch 0, Iteration 200, loss = 1.6827
Checking accuracy on validation set
Got 447 / 1000 correct (44.70)

Epoch 0, Iteration 300, loss = 1.2107
Checking accuracy on validation set
Got 489 / 1000 correct (48.90)

Epoch 0, Iteration 400, loss = 1.1421
Checking accuracy on validation set
Got 530 / 1000 correct (53.00)

Epoch 0, Iteration 500, loss = 1.3717
Checking accuracy on validation set
Got 549 / 1000 correct (54.90)
```

```
Epoch 0, Iteration 600, loss = 1.2229
Checking accuracy on validation set
Got 556 / 1000 correct (55.60)

Epoch 0, Iteration 700, loss = 1.2221
Checking accuracy on validation set
Got 564 / 1000 correct (56.40)

Epoch 1, Iteration 0, loss = 0.9969
Checking accuracy on validation set
Got 573 / 1000 correct (57.30)

Epoch 1, Iteration 100, loss = 1.1605
Checking accuracy on validation set
Got 596 / 1000 correct (59.60)

Epoch 1, Iteration 200, loss = 1.1781
Checking accuracy on validation set
Got 572 / 1000 correct (57.20)

Epoch 1, Iteration 300, loss = 1.1549
Checking accuracy on validation set
Got 581 / 1000 correct (58.10)

Epoch 1, Iteration 400, loss = 1.0037
Checking accuracy on validation set
Got 609 / 1000 correct (60.90)

Epoch 1, Iteration 500, loss = 0.9147
Checking accuracy on validation set
Got 589 / 1000 correct (58.90)

Epoch 1, Iteration 600, loss = 0.9904
Checking accuracy on validation set
Got 612 / 1000 correct (61.20)

Epoch 1, Iteration 700, loss = 1.1474
Checking accuracy on validation set
Got 603 / 1000 correct (60.30)

Epoch 2, Iteration 0, loss = 0.8510
Checking accuracy on validation set
Got 584 / 1000 correct (58.40)

Epoch 2, Iteration 100, loss = 0.9505
Checking accuracy on validation set
Got 608 / 1000 correct (60.80)
```

```
Epoch 2, Iteration 200, loss = 0.9030
Checking accuracy on validation set
Got 596 / 1000 correct (59.60)

Epoch 2, Iteration 300, loss = 0.6908
Checking accuracy on validation set
Got 613 / 1000 correct (61.30)

Epoch 2, Iteration 400, loss = 1.0968
Checking accuracy on validation set
Got 623 / 1000 correct (62.30)

Epoch 2, Iteration 500, loss = 1.1535
Checking accuracy on validation set
Got 631 / 1000 correct (63.10)

Epoch 2, Iteration 600, loss = 1.0932
Checking accuracy on validation set
Got 625 / 1000 correct (62.50)

Epoch 2, Iteration 700, loss = 1.1419
Checking accuracy on validation set
Got 607 / 1000 correct (60.70)

Epoch 3, Iteration 0, loss = 0.8151
Checking accuracy on validation set
Got 616 / 1000 correct (61.60)

Epoch 3, Iteration 100, loss = 1.1970
Checking accuracy on validation set
Got 629 / 1000 correct (62.90)

Epoch 3, Iteration 200, loss = 0.7049
Checking accuracy on validation set
Got 617 / 1000 correct (61.70)

Epoch 3, Iteration 300, loss = 0.6545
Checking accuracy on validation set
Got 626 / 1000 correct (62.60)

Epoch 3, Iteration 400, loss = 0.9516
Checking accuracy on validation set
Got 599 / 1000 correct (59.90)

Epoch 3, Iteration 500, loss = 0.9490
Checking accuracy on validation set
Got 612 / 1000 correct (61.20)
```

```
Epoch 3, Iteration 600, loss = 1.1471
Checking accuracy on validation set
Got 613 / 1000 correct (61.30)

Epoch 3, Iteration 700, loss = 0.9088
Checking accuracy on validation set
Got 623 / 1000 correct (62.30)

Epoch 4, Iteration 0, loss = 0.6591
Checking accuracy on validation set
Got 619 / 1000 correct (61.90)

Epoch 4, Iteration 100, loss = 0.7624
Checking accuracy on validation set
Got 626 / 1000 correct (62.60)

Epoch 4, Iteration 200, loss = 0.7869
Checking accuracy on validation set
Got 619 / 1000 correct (61.90)

Epoch 4, Iteration 300, loss = 0.6548
Checking accuracy on validation set
Got 619 / 1000 correct (61.90)

Epoch 4, Iteration 400, loss = 0.8292
Checking accuracy on validation set
Got 641 / 1000 correct (64.10)

Epoch 4, Iteration 500, loss = 0.7641
Checking accuracy on validation set
Got 632 / 1000 correct (63.20)

Epoch 4, Iteration 600, loss = 0.8133
Checking accuracy on validation set
Got 637 / 1000 correct (63.70)

Epoch 4, Iteration 700, loss = 0.7195
Checking accuracy on validation set
Got 616 / 1000 correct (61.60)

Epoch 5, Iteration 0, loss = 0.7579
Checking accuracy on validation set
Got 598 / 1000 correct (59.80)

Epoch 5, Iteration 100, loss = 0.4717
Checking accuracy on validation set
Got 612 / 1000 correct (61.20)
```

```
Epoch 5, Iteration 200, loss = 0.5900
Checking accuracy on validation set
Got 610 / 1000 correct (61.00)

Epoch 5, Iteration 300, loss = 0.5545
Checking accuracy on validation set
Got 614 / 1000 correct (61.40)

Epoch 5, Iteration 400, loss = 0.6165
Checking accuracy on validation set
Got 603 / 1000 correct (60.30)

Epoch 5, Iteration 500, loss = 0.6379
Checking accuracy on validation set
Got 610 / 1000 correct (61.00)

Epoch 5, Iteration 600, loss = 0.6086
Checking accuracy on validation set
Got 610 / 1000 correct (61.00)

Epoch 5, Iteration 700, loss = 1.1581
Checking accuracy on validation set
Got 603 / 1000 correct (60.30)

Epoch 6, Iteration 0, loss = 0.5652
Checking accuracy on validation set
Got 616 / 1000 correct (61.60)

Epoch 6, Iteration 100, loss = 0.5739
Checking accuracy on validation set
Got 605 / 1000 correct (60.50)

Epoch 6, Iteration 200, loss = 0.5644
Checking accuracy on validation set
Got 629 / 1000 correct (62.90)

Epoch 6, Iteration 300, loss = 0.3756
Checking accuracy on validation set
Got 608 / 1000 correct (60.80)

Epoch 6, Iteration 400, loss = 0.5809
Checking accuracy on validation set
Got 610 / 1000 correct (61.00)

Epoch 6, Iteration 500, loss = 0.4685
Checking accuracy on validation set
Got 618 / 1000 correct (61.80)
```

```
Epoch 6, Iteration 600, loss = 0.5972
Checking accuracy on validation set
Got 611 / 1000 correct (61.10)

Epoch 6, Iteration 700, loss = 0.7350
Checking accuracy on validation set
Got 619 / 1000 correct (61.90)

Epoch 7, Iteration 0, loss = 0.3977
Checking accuracy on validation set
Got 607 / 1000 correct (60.70)

Epoch 7, Iteration 100, loss = 0.4126
Checking accuracy on validation set
Got 605 / 1000 correct (60.50)

Epoch 7, Iteration 200, loss = 0.6383
Checking accuracy on validation set
Got 588 / 1000 correct (58.80)

Epoch 7, Iteration 300, loss = 0.3596
Checking accuracy on validation set
Got 608 / 1000 correct (60.80)

Epoch 7, Iteration 400, loss = 0.5627
Checking accuracy on validation set
Got 609 / 1000 correct (60.90)

Epoch 7, Iteration 500, loss = 0.4669
Checking accuracy on validation set
Got 596 / 1000 correct (59.60)

Epoch 7, Iteration 600, loss = 0.3887
Checking accuracy on validation set
Got 622 / 1000 correct (62.20)

Epoch 7, Iteration 700, loss = 0.4287
Checking accuracy on validation set
Got 612 / 1000 correct (61.20)

Epoch 8, Iteration 0, loss = 0.4137
Checking accuracy on validation set
Got 593 / 1000 correct (59.30)

Epoch 8, Iteration 100, loss = 0.6287
Checking accuracy on validation set
Got 609 / 1000 correct (60.90)
```

```
Epoch 8, Iteration 200, loss = 0.2699
Checking accuracy on validation set
Got 600 / 1000 correct (60.00)

Epoch 8, Iteration 300, loss = 0.4436
Checking accuracy on validation set
Got 604 / 1000 correct (60.40)

Epoch 8, Iteration 400, loss = 0.5692
Checking accuracy on validation set
Got 605 / 1000 correct (60.50)

Epoch 8, Iteration 500, loss = 0.3970
Checking accuracy on validation set
Got 603 / 1000 correct (60.30)

Epoch 8, Iteration 600, loss = 0.4447
Checking accuracy on validation set
Got 618 / 1000 correct (61.80)

Epoch 8, Iteration 700, loss = 0.2976
Checking accuracy on validation set
Got 588 / 1000 correct (58.80)

Epoch 9, Iteration 0, loss = 0.3414
Checking accuracy on validation set
Got 604 / 1000 correct (60.40)

Epoch 9, Iteration 100, loss = 0.2387
Checking accuracy on validation set
Got 599 / 1000 correct (59.90)

Epoch 9, Iteration 200, loss = 0.3102
Checking accuracy on validation set
Got 596 / 1000 correct (59.60)

Epoch 9, Iteration 300, loss = 0.3547
Checking accuracy on validation set
Got 609 / 1000 correct (60.90)

Epoch 9, Iteration 400, loss = 0.3677
Checking accuracy on validation set
Got 604 / 1000 correct (60.40)

Epoch 9, Iteration 500, loss = 0.2807
Checking accuracy on validation set
Got 594 / 1000 correct (59.40)
```

```
Epoch 9, Iteration 600, loss = 0.4219
Checking accuracy on validation set
Got 590 / 1000 correct (59.00)

Epoch 9, Iteration 700, loss = 0.7132
Checking accuracy on validation set
Got 590 / 1000 correct (59.00)

Maximum accuracy attained:  64.1
```

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

[13]:

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

```
Checking accuracy on test set
Got 5798 / 10000 correct (57.98)
```

[13]:

```
57.98
```

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: Self-attention is equivalent to non-local means. The non-local means algorithm replaces the value of a pixel by an average of a selection of other pixels values, while self-attention is an attention mechanism relating different positions of a single sequence in order to compute a representation of the same sequence. K-means is a method of vector quantization that aims to partition n observations into k clusters in which each observation belongs to the cluster with the nearest mean. A residual block is a stack of layers set in such a way that the output of a layer is taken and added to another layer deeper in the block, but it includes no attention. Guassian blurring is to blur an image by Gaussian function.

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

[60]:

```
# 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).reshape(N, C, H*W)
        # 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)
        # 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
        temp = torch.matmul(q, torch.transpose(k, 1, 2))/(np.sqrt(C))
        attention = torch.matmul(F.softmax(temp, dim=-1), v)
        # 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)

[77]:

```
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),
    nn.ReLU(),
    Flatten(),
    nn.Linear(channel_2*32*32, 10),
)

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

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

```
Epoch 0, Iteration 0, loss = 2.3018
Checking accuracy on validation set
Got 135 / 1000 correct (13.50)

Epoch 0, Iteration 100, loss = 1.4160
Checking accuracy on validation set
Got 385 / 1000 correct (38.50)

Epoch 0, Iteration 200, loss = 1.4014
Checking accuracy on validation set
Got 468 / 1000 correct (46.80)

Epoch 0, Iteration 300, loss = 1.4034
Checking accuracy on validation set
Got 533 / 1000 correct (53.30)

Epoch 0, Iteration 400, loss = 1.5503
Checking accuracy on validation set
Got 555 / 1000 correct (55.50)

Epoch 0, Iteration 500, loss = 1.1840
Checking accuracy on validation set
Got 564 / 1000 correct (56.40)

Epoch 0, Iteration 600, loss = 1.1146
Checking accuracy on validation set
Got 581 / 1000 correct (58.10)
```

```
Epoch 0, Iteration 700, loss = 1.0500
Checking accuracy on validation set
Got 612 / 1000 correct (61.20)

Epoch 1, Iteration 0, loss = 1.1237
Checking accuracy on validation set
Got 624 / 1000 correct (62.40)

Epoch 1, Iteration 100, loss = 1.0338
Checking accuracy on validation set
Got 602 / 1000 correct (60.20)

Epoch 1, Iteration 200, loss = 1.1228
Checking accuracy on validation set
Got 624 / 1000 correct (62.40)

Epoch 1, Iteration 300, loss = 1.2680
Checking accuracy on validation set
Got 618 / 1000 correct (61.80)

Epoch 1, Iteration 400, loss = 1.1750
Checking accuracy on validation set
Got 631 / 1000 correct (63.10)

Epoch 1, Iteration 500, loss = 0.9403
Checking accuracy on validation set
Got 619 / 1000 correct (61.90)

Epoch 1, Iteration 600, loss = 1.0290
Checking accuracy on validation set
Got 649 / 1000 correct (64.90)

Epoch 1, Iteration 700, loss = 0.9683
Checking accuracy on validation set
Got 637 / 1000 correct (63.70)

Epoch 2, Iteration 0, loss = 0.9890
Checking accuracy on validation set
Got 649 / 1000 correct (64.90)

Epoch 2, Iteration 100, loss = 1.1009
Checking accuracy on validation set
Got 638 / 1000 correct (63.80)

Epoch 2, Iteration 200, loss = 0.7541
Checking accuracy on validation set
Got 644 / 1000 correct (64.40)
```

```
Epoch 2, Iteration 300, loss = 0.8767
Checking accuracy on validation set
Got 647 / 1000 correct (64.70)

Epoch 2, Iteration 400, loss = 0.9723
Checking accuracy on validation set
Got 644 / 1000 correct (64.40)

Epoch 2, Iteration 500, loss = 1.0035
Checking accuracy on validation set
Got 646 / 1000 correct (64.60)

Epoch 2, Iteration 600, loss = 0.6993
Checking accuracy on validation set
Got 648 / 1000 correct (64.80)

Epoch 2, Iteration 700, loss = 0.8933
Checking accuracy on validation set
Got 667 / 1000 correct (66.70)

Epoch 3, Iteration 0, loss = 0.5578
Checking accuracy on validation set
Got 672 / 1000 correct (67.20)

Epoch 3, Iteration 100, loss = 0.5692
Checking accuracy on validation set
Got 675 / 1000 correct (67.50)

Epoch 3, Iteration 200, loss = 0.4396
Checking accuracy on validation set
Got 669 / 1000 correct (66.90)

Epoch 3, Iteration 300, loss = 0.5937
Checking accuracy on validation set
Got 669 / 1000 correct (66.90)

Epoch 3, Iteration 400, loss = 0.6461
Checking accuracy on validation set
Got 665 / 1000 correct (66.50)

Epoch 3, Iteration 500, loss = 0.5879
Checking accuracy on validation set
Got 655 / 1000 correct (65.50)

Epoch 3, Iteration 600, loss = 0.7000
Checking accuracy on validation set
Got 644 / 1000 correct (64.40)
```

```
Epoch 3, Iteration 700, loss = 1.1036
Checking accuracy on validation set
Got 644 / 1000 correct (64.40)

Epoch 4, Iteration 0, loss = 0.5557
Checking accuracy on validation set
Got 640 / 1000 correct (64.00)

Epoch 4, Iteration 100, loss = 0.4355
Checking accuracy on validation set
Got 652 / 1000 correct (65.20)

Epoch 4, Iteration 200, loss = 0.6221
Checking accuracy on validation set
Got 653 / 1000 correct (65.30)

Epoch 4, Iteration 300, loss = 0.4446
Checking accuracy on validation set
Got 648 / 1000 correct (64.80)

Epoch 4, Iteration 400, loss = 0.5432
Checking accuracy on validation set
Got 651 / 1000 correct (65.10)

Epoch 4, Iteration 500, loss = 0.6013
Checking accuracy on validation set
Got 658 / 1000 correct (65.80)

Epoch 4, Iteration 600, loss = 0.6315
Checking accuracy on validation set
Got 650 / 1000 correct (65.00)

Epoch 4, Iteration 700, loss = 0.7326
Checking accuracy on validation set
Got 659 / 1000 correct (65.90)

Epoch 5, Iteration 0, loss = 0.3083
Checking accuracy on validation set
Got 656 / 1000 correct (65.60)

Epoch 5, Iteration 100, loss = 0.2382
Checking accuracy on validation set
Got 651 / 1000 correct (65.10)

Epoch 5, Iteration 200, loss = 0.2225
Checking accuracy on validation set
Got 652 / 1000 correct (65.20)
```

```
Epoch 5, Iteration 300, loss = 0.7069
Checking accuracy on validation set
Got 663 / 1000 correct (66.30)

Epoch 5, Iteration 400, loss = 0.2858
Checking accuracy on validation set
Got 642 / 1000 correct (64.20)

Epoch 5, Iteration 500, loss = 0.3043
Checking accuracy on validation set
Got 642 / 1000 correct (64.20)

Epoch 5, Iteration 600, loss = 0.5002
Checking accuracy on validation set
Got 649 / 1000 correct (64.90)

Epoch 5, Iteration 700, loss = 0.3562
Checking accuracy on validation set
Got 649 / 1000 correct (64.90)

Epoch 6, Iteration 0, loss = 0.2536
Checking accuracy on validation set
Got 651 / 1000 correct (65.10)

Epoch 6, Iteration 100, loss = 0.3126
Checking accuracy on validation set
Got 649 / 1000 correct (64.90)

Epoch 6, Iteration 200, loss = 0.3114
Checking accuracy on validation set
Got 644 / 1000 correct (64.40)

Epoch 6, Iteration 300, loss = 0.2444
Checking accuracy on validation set
Got 644 / 1000 correct (64.40)

Epoch 6, Iteration 400, loss = 0.3506
Checking accuracy on validation set
Got 645 / 1000 correct (64.50)

Epoch 6, Iteration 500, loss = 0.2397
Checking accuracy on validation set
Got 648 / 1000 correct (64.80)

Epoch 6, Iteration 600, loss = 0.3803
Checking accuracy on validation set
Got 642 / 1000 correct (64.20)
```

```
Epoch 6, Iteration 700, loss = 0.2979
Checking accuracy on validation set
Got 643 / 1000 correct (64.30)

Epoch 7, Iteration 0, loss = 0.1825
Checking accuracy on validation set
Got 638 / 1000 correct (63.80)

Epoch 7, Iteration 100, loss = 0.1549
Checking accuracy on validation set
Got 630 / 1000 correct (63.00)

Epoch 7, Iteration 200, loss = 0.1094
Checking accuracy on validation set
Got 640 / 1000 correct (64.00)

Epoch 7, Iteration 300, loss = 0.1826
Checking accuracy on validation set
Got 645 / 1000 correct (64.50)

Epoch 7, Iteration 400, loss = 0.2364
Checking accuracy on validation set
Got 629 / 1000 correct (62.90)

Epoch 7, Iteration 500, loss = 0.2258
Checking accuracy on validation set
Got 642 / 1000 correct (64.20)

Epoch 7, Iteration 600, loss = 0.2122
Checking accuracy on validation set
Got 626 / 1000 correct (62.60)

Epoch 7, Iteration 700, loss = 0.1620
Checking accuracy on validation set
Got 654 / 1000 correct (65.40)

Epoch 8, Iteration 0, loss = 0.1156
Checking accuracy on validation set
Got 634 / 1000 correct (63.40)

Epoch 8, Iteration 100, loss = 0.1672
Checking accuracy on validation set
Got 631 / 1000 correct (63.10)

Epoch 8, Iteration 200, loss = 0.1111
Checking accuracy on validation set
Got 642 / 1000 correct (64.20)
```

```
Epoch 8, Iteration 300, loss = 0.1551
Checking accuracy on validation set
Got 637 / 1000 correct (63.70)

Epoch 8, Iteration 400, loss = 0.1029
Checking accuracy on validation set
Got 626 / 1000 correct (62.60)

Epoch 8, Iteration 500, loss = 0.1642
Checking accuracy on validation set
Got 627 / 1000 correct (62.70)

Epoch 8, Iteration 600, loss = 0.1828
Checking accuracy on validation set
Got 626 / 1000 correct (62.60)

Epoch 8, Iteration 700, loss = 0.1041
Checking accuracy on validation set
Got 628 / 1000 correct (62.80)

Epoch 9, Iteration 0, loss = 0.0583
Checking accuracy on validation set
Got 636 / 1000 correct (63.60)

Epoch 9, Iteration 100, loss = 0.0605
Checking accuracy on validation set
Got 635 / 1000 correct (63.50)

Epoch 9, Iteration 200, loss = 0.0386
Checking accuracy on validation set
Got 632 / 1000 correct (63.20)

Epoch 9, Iteration 300, loss = 0.1753
Checking accuracy on validation set
Got 624 / 1000 correct (62.40)

Epoch 9, Iteration 400, loss = 0.0989
Checking accuracy on validation set
Got 625 / 1000 correct (62.50)

Epoch 9, Iteration 500, loss = 0.1496
Checking accuracy on validation set
Got 630 / 1000 correct (63.00)

Epoch 9, Iteration 600, loss = 0.2202
Checking accuracy on validation set
Got 632 / 1000 correct (63.20)
```

```
Epoch 9, Iteration 700, loss = 0.2224
Checking accuracy on validation set
Got 631 / 1000 correct (63.10)

Maximum accuracy attained:  67.5
```

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. *

[78]:
```
earlyAttention = model
check_accuracy(loader_test, earlyAttention)
```

```
Checking accuracy on test set
Got 6143 / 10000 correct (61.43)
```

[78]:
```
61.42999999999999
```

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

[79]:
```
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),
    nn.ReLU(),
    Attention(channel_2),
    nn.ReLU(),
    Flatten(),
    nn.Linear(channel_2*32*32, 10),
)

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

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

```
Epoch 0, Iteration 0, loss = 2.3077
Checking accuracy on validation set
```

```
Got 168 / 1000 correct (16.80)

Epoch 0, Iteration 100, loss = 1.4646
Checking accuracy on validation set
Got 419 / 1000 correct (41.90)

Epoch 0, Iteration 200, loss = 1.2998
Checking accuracy on validation set
Got 476 / 1000 correct (47.60)

Epoch 0, Iteration 300, loss = 1.2162
Checking accuracy on validation set
Got 530 / 1000 correct (53.00)

Epoch 0, Iteration 400, loss = 1.4571
Checking accuracy on validation set
Got 503 / 1000 correct (50.30)

Epoch 0, Iteration 500, loss = 1.1879
Checking accuracy on validation set
Got 559 / 1000 correct (55.90)

Epoch 0, Iteration 600, loss = 1.1361
Checking accuracy on validation set
Got 581 / 1000 correct (58.10)

Epoch 0, Iteration 700, loss = 1.4091
Checking accuracy on validation set
Got 565 / 1000 correct (56.50)

Epoch 1, Iteration 0, loss = 1.0744
Checking accuracy on validation set
Got 597 / 1000 correct (59.70)

Epoch 1, Iteration 100, loss = 1.1168
Checking accuracy on validation set
Got 590 / 1000 correct (59.00)

Epoch 1, Iteration 200, loss = 1.1873
Checking accuracy on validation set
Got 587 / 1000 correct (58.70)

Epoch 1, Iteration 300, loss = 0.9003
Checking accuracy on validation set
Got 609 / 1000 correct (60.90)

Epoch 1, Iteration 400, loss = 0.8909
Checking accuracy on validation set
```

```
Got 624 / 1000 correct (62.40)

Epoch 1, Iteration 500, loss = 1.0935
Checking accuracy on validation set
Got 612 / 1000 correct (61.20)

Epoch 1, Iteration 600, loss = 0.9409
Checking accuracy on validation set
Got 628 / 1000 correct (62.80)

Epoch 1, Iteration 700, loss = 1.2670
Checking accuracy on validation set
Got 625 / 1000 correct (62.50)

Epoch 2, Iteration 0, loss = 1.0128
Checking accuracy on validation set
Got 612 / 1000 correct (61.20)

Epoch 2, Iteration 100, loss = 0.7269
Checking accuracy on validation set
Got 625 / 1000 correct (62.50)

Epoch 2, Iteration 200, loss = 0.9698
Checking accuracy on validation set
Got 631 / 1000 correct (63.10)

Epoch 2, Iteration 300, loss = 1.0668
Checking accuracy on validation set
Got 640 / 1000 correct (64.00)

Epoch 2, Iteration 400, loss = 0.8458
Checking accuracy on validation set
Got 652 / 1000 correct (65.20)

Epoch 2, Iteration 500, loss = 0.8796
Checking accuracy on validation set
Got 651 / 1000 correct (65.10)

Epoch 2, Iteration 600, loss = 0.8085
Checking accuracy on validation set
Got 613 / 1000 correct (61.30)

Epoch 2, Iteration 700, loss = 0.9217
Checking accuracy on validation set
Got 637 / 1000 correct (63.70)

Epoch 3, Iteration 0, loss = 0.8025
Checking accuracy on validation set
```

```
Got 647 / 1000 correct (64.70)

Epoch 3, Iteration 100, loss = 0.8501
Checking accuracy on validation set
Got 640 / 1000 correct (64.00)

Epoch 3, Iteration 200, loss = 0.8331
Checking accuracy on validation set
Got 616 / 1000 correct (61.60)

Epoch 3, Iteration 300, loss = 1.1735
Checking accuracy on validation set
Got 638 / 1000 correct (63.80)

Epoch 3, Iteration 400, loss = 0.9421
Checking accuracy on validation set
Got 652 / 1000 correct (65.20)

Epoch 3, Iteration 500, loss = 0.5002
Checking accuracy on validation set
Got 645 / 1000 correct (64.50)

Epoch 3, Iteration 600, loss = 0.8466
Checking accuracy on validation set
Got 624 / 1000 correct (62.40)

Epoch 3, Iteration 700, loss = 0.8973
Checking accuracy on validation set
Got 641 / 1000 correct (64.10)

Epoch 4, Iteration 0, loss = 0.5287
Checking accuracy on validation set
Got 645 / 1000 correct (64.50)

Epoch 4, Iteration 100, loss = 0.5748
Checking accuracy on validation set
Got 650 / 1000 correct (65.00)

Epoch 4, Iteration 200, loss = 0.7261
Checking accuracy on validation set
Got 644 / 1000 correct (64.40)

Epoch 4, Iteration 300, loss = 0.5291
Checking accuracy on validation set
Got 643 / 1000 correct (64.30)

Epoch 4, Iteration 400, loss = 0.4969
Checking accuracy on validation set
```

```
Got 652 / 1000 correct (65.20)

Epoch 4, Iteration 500, loss = 0.5342
Checking accuracy on validation set
Got 638 / 1000 correct (63.80)

Epoch 4, Iteration 600, loss = 0.9069
Checking accuracy on validation set
Got 626 / 1000 correct (62.60)

Epoch 4, Iteration 700, loss = 0.5183
Checking accuracy on validation set
Got 628 / 1000 correct (62.80)

Epoch 5, Iteration 0, loss = 0.5493
Checking accuracy on validation set
Got 641 / 1000 correct (64.10)

Epoch 5, Iteration 100, loss = 0.3091
Checking accuracy on validation set
Got 640 / 1000 correct (64.00)

Epoch 5, Iteration 200, loss = 0.4340
Checking accuracy on validation set
Got 640 / 1000 correct (64.00)

Epoch 5, Iteration 300, loss = 0.5580
Checking accuracy on validation set
Got 633 / 1000 correct (63.30)

Epoch 5, Iteration 400, loss = 0.5448
Checking accuracy on validation set
Got 637 / 1000 correct (63.70)

Epoch 5, Iteration 500, loss = 0.5477
Checking accuracy on validation set
Got 635 / 1000 correct (63.50)

Epoch 5, Iteration 600, loss = 0.3799
Checking accuracy on validation set
Got 637 / 1000 correct (63.70)

Epoch 5, Iteration 700, loss = 0.6082
Checking accuracy on validation set
Got 639 / 1000 correct (63.90)

Epoch 6, Iteration 0, loss = 0.3817
Checking accuracy on validation set
```

```
Got 630 / 1000 correct (63.00)

Epoch 6, Iteration 100, loss = 0.2815
Checking accuracy on validation set
Got 645 / 1000 correct (64.50)

Epoch 6, Iteration 200, loss = 0.1886
Checking accuracy on validation set
Got 621 / 1000 correct (62.10)

Epoch 6, Iteration 300, loss = 0.4701
Checking accuracy on validation set
Got 635 / 1000 correct (63.50)

Epoch 6, Iteration 400, loss = 0.4539
Checking accuracy on validation set
Got 622 / 1000 correct (62.20)

Epoch 6, Iteration 500, loss = 0.2806
Checking accuracy on validation set
Got 645 / 1000 correct (64.50)

Epoch 6, Iteration 600, loss = 0.4829
Checking accuracy on validation set
Got 635 / 1000 correct (63.50)

Epoch 6, Iteration 700, loss = 0.6195
Checking accuracy on validation set
Got 623 / 1000 correct (62.30)

Epoch 7, Iteration 0, loss = 0.3313
Checking accuracy on validation set
Got 626 / 1000 correct (62.60)

Epoch 7, Iteration 100, loss = 0.3418
Checking accuracy on validation set
Got 620 / 1000 correct (62.00)

Epoch 7, Iteration 200, loss = 0.2370
Checking accuracy on validation set
Got 627 / 1000 correct (62.70)

Epoch 7, Iteration 300, loss = 0.3746
Checking accuracy on validation set
Got 617 / 1000 correct (61.70)

Epoch 7, Iteration 400, loss = 0.3976
Checking accuracy on validation set
```

```
Got 633 / 1000 correct (63.30)

Epoch 7, Iteration 500, loss = 0.6026
Checking accuracy on validation set
Got 612 / 1000 correct (61.20)

Epoch 7, Iteration 600, loss = 0.4728
Checking accuracy on validation set
Got 609 / 1000 correct (60.90)

Epoch 7, Iteration 700, loss = 0.3789
Checking accuracy on validation set
Got 624 / 1000 correct (62.40)

Epoch 8, Iteration 0, loss = 0.3086
Checking accuracy on validation set
Got 600 / 1000 correct (60.00)

Epoch 8, Iteration 100, loss = 0.2041
Checking accuracy on validation set
Got 614 / 1000 correct (61.40)

Epoch 8, Iteration 200, loss = 0.4008
Checking accuracy on validation set
Got 616 / 1000 correct (61.60)

Epoch 8, Iteration 300, loss = 0.1932
Checking accuracy on validation set
Got 602 / 1000 correct (60.20)

Epoch 8, Iteration 400, loss = 0.3230
Checking accuracy on validation set
Got 612 / 1000 correct (61.20)

Epoch 8, Iteration 500, loss = 0.2452
Checking accuracy on validation set
Got 612 / 1000 correct (61.20)

Epoch 8, Iteration 600, loss = 0.3668
Checking accuracy on validation set
Got 603 / 1000 correct (60.30)

Epoch 8, Iteration 700, loss = 0.4054
Checking accuracy on validation set
Got 614 / 1000 correct (61.40)

Epoch 9, Iteration 0, loss = 0.1729
Checking accuracy on validation set
```

```
Got 608 / 1000 correct (60.80)

Epoch 9, Iteration 100, loss = 0.1560
Checking accuracy on validation set
Got 617 / 1000 correct (61.70)

Epoch 9, Iteration 200, loss = 0.1799
Checking accuracy on validation set
Got 619 / 1000 correct (61.90)

Epoch 9, Iteration 300, loss = 0.1635
Checking accuracy on validation set
Got 610 / 1000 correct (61.00)

Epoch 9, Iteration 400, loss = 0.4225
Checking accuracy on validation set
Got 606 / 1000 correct (60.60)

Epoch 9, Iteration 500, loss = 0.3201
Checking accuracy on validation set
Got 604 / 1000 correct (60.40)

Epoch 9, Iteration 600, loss = 0.3285
Checking accuracy on validation set
Got 611 / 1000 correct (61.10)

Epoch 9, Iteration 700, loss = 0.1595
Checking accuracy on validation set
Got 612 / 1000 correct (61.20)

Maximum accuracy attained:  65.2
```

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.

[80]:

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

```
Checking accuracy on test set
Got 6174 / 10000 correct (61.74)
```

[80]:

```
61.739999999999995
```

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: The main difference between the self-attention and attention mechanism is that self-attention can only learn attention from its own layer while the attention mechanism can learn it from other layers.

Attention Mechanism: Convolutional Block Attention Module

Self-attention Mechanism: Self-Attention Generative Adversarial Networks

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

[65]:

```
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),
    nn.ReLU(),
    Attention(channel_2),
    nn.ReLU(),
    Flatten(),
    nn.Linear(channel_2*32*32, 10),
)

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

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

```
Epoch 0, Iteration 0, loss = 2.3040
Checking accuracy on validation set
Got 119 / 1000 correct (11.90)

Epoch 0, Iteration 100, loss = 1.5456
Checking accuracy on validation set
Got 429 / 1000 correct (42.90)

Epoch 0, Iteration 200, loss = 1.4643
Checking accuracy on validation set
Got 484 / 1000 correct (48.40)
```

```
Epoch 0, Iteration 300, loss = 1.5998
Checking accuracy on validation set
Got 515 / 1000 correct (51.50)

Epoch 0, Iteration 400, loss = 1.5197
Checking accuracy on validation set
Got 545 / 1000 correct (54.50)

Epoch 0, Iteration 500, loss = 1.0481
Checking accuracy on validation set
Got 552 / 1000 correct (55.20)

Epoch 0, Iteration 600, loss = 1.1773
Checking accuracy on validation set
Got 553 / 1000 correct (55.30)

Epoch 0, Iteration 700, loss = 0.9678
Checking accuracy on validation set
Got 566 / 1000 correct (56.60)

Epoch 1, Iteration 0, loss = 1.0036
Checking accuracy on validation set
Got 575 / 1000 correct (57.50)

Epoch 1, Iteration 100, loss = 1.1466
Checking accuracy on validation set
Got 603 / 1000 correct (60.30)

Epoch 1, Iteration 200, loss = 0.9375
Checking accuracy on validation set
Got 588 / 1000 correct (58.80)

Epoch 1, Iteration 300, loss = 1.1397
Checking accuracy on validation set
Got 622 / 1000 correct (62.20)

Epoch 1, Iteration 400, loss = 1.1576
Checking accuracy on validation set
Got 611 / 1000 correct (61.10)

Epoch 1, Iteration 500, loss = 1.0000
Checking accuracy on validation set
Got 608 / 1000 correct (60.80)

Epoch 1, Iteration 600, loss = 1.0864
Checking accuracy on validation set
Got 590 / 1000 correct (59.00)
```

```
Epoch 1, Iteration 700, loss = 1.1386
Checking accuracy on validation set
Got 609 / 1000 correct (60.90)

Epoch 2, Iteration 0, loss = 0.7981
Checking accuracy on validation set
Got 620 / 1000 correct (62.00)

Epoch 2, Iteration 100, loss = 0.7128
Checking accuracy on validation set
Got 640 / 1000 correct (64.00)

Epoch 2, Iteration 200, loss = 0.8733
Checking accuracy on validation set
Got 643 / 1000 correct (64.30)

Epoch 2, Iteration 300, loss = 0.9795
Checking accuracy on validation set
Got 639 / 1000 correct (63.90)

Epoch 2, Iteration 400, loss = 0.5842
Checking accuracy on validation set
Got 636 / 1000 correct (63.60)

Epoch 2, Iteration 500, loss = 0.9182
Checking accuracy on validation set
Got 659 / 1000 correct (65.90)

Epoch 2, Iteration 600, loss = 0.8507
Checking accuracy on validation set
Got 640 / 1000 correct (64.00)

Epoch 2, Iteration 700, loss = 0.8502
Checking accuracy on validation set
Got 642 / 1000 correct (64.20)

Epoch 3, Iteration 0, loss = 0.6253
Checking accuracy on validation set
Got 642 / 1000 correct (64.20)

Epoch 3, Iteration 100, loss = 0.8884
Checking accuracy on validation set
Got 647 / 1000 correct (64.70)

Epoch 3, Iteration 200, loss = 1.1805
Checking accuracy on validation set
Got 648 / 1000 correct (64.80)
```

```
Epoch 3, Iteration 300, loss = 0.7876
Checking accuracy on validation set
Got 651 / 1000 correct (65.10)

Epoch 3, Iteration 400, loss = 0.8909
Checking accuracy on validation set
Got 652 / 1000 correct (65.20)

Epoch 3, Iteration 500, loss = 0.8778
Checking accuracy on validation set
Got 660 / 1000 correct (66.00)

Epoch 3, Iteration 600, loss = 0.6423
Checking accuracy on validation set
Got 662 / 1000 correct (66.20)

Epoch 3, Iteration 700, loss = 0.7938
Checking accuracy on validation set
Got 678 / 1000 correct (67.80)

Epoch 4, Iteration 0, loss = 0.7666
Checking accuracy on validation set
Got 643 / 1000 correct (64.30)

Epoch 4, Iteration 100, loss = 0.7139
Checking accuracy on validation set
Got 673 / 1000 correct (67.30)

Epoch 4, Iteration 200, loss = 0.6053
Checking accuracy on validation set
Got 666 / 1000 correct (66.60)

Epoch 4, Iteration 300, loss = 0.5525
Checking accuracy on validation set
Got 668 / 1000 correct (66.80)

Epoch 4, Iteration 400, loss = 0.5519
Checking accuracy on validation set
Got 662 / 1000 correct (66.20)

Epoch 4, Iteration 500, loss = 0.6539
Checking accuracy on validation set
Got 672 / 1000 correct (67.20)

Epoch 4, Iteration 600, loss = 0.9470
Checking accuracy on validation set
Got 654 / 1000 correct (65.40)
```

```
Epoch 4, Iteration 700, loss = 0.7922
Checking accuracy on validation set
Got 649 / 1000 correct (64.90)

Epoch 5, Iteration 0, loss = 0.4748
Checking accuracy on validation set
Got 649 / 1000 correct (64.90)

Epoch 5, Iteration 100, loss = 0.5339
Checking accuracy on validation set
Got 662 / 1000 correct (66.20)

Epoch 5, Iteration 200, loss = 0.6886
Checking accuracy on validation set
Got 654 / 1000 correct (65.40)

Epoch 5, Iteration 300, loss = 0.5690
Checking accuracy on validation set
Got 658 / 1000 correct (65.80)

Epoch 5, Iteration 400, loss = 0.6491
Checking accuracy on validation set
Got 657 / 1000 correct (65.70)

Epoch 5, Iteration 500, loss = 0.6101
Checking accuracy on validation set
Got 661 / 1000 correct (66.10)

Epoch 5, Iteration 600, loss = 0.5917
Checking accuracy on validation set
Got 649 / 1000 correct (64.90)

Epoch 5, Iteration 700, loss = 0.7913
Checking accuracy on validation set
Got 666 / 1000 correct (66.60)

Epoch 6, Iteration 0, loss = 0.4137
Checking accuracy on validation set
Got 650 / 1000 correct (65.00)

Epoch 6, Iteration 100, loss = 0.4396
Checking accuracy on validation set
Got 657 / 1000 correct (65.70)

Epoch 6, Iteration 200, loss = 0.3564
Checking accuracy on validation set
Got 682 / 1000 correct (68.20)
```

```
Epoch 6, Iteration 300, loss = 0.4653
Checking accuracy on validation set
Got 646 / 1000 correct (64.60)

Epoch 6, Iteration 400, loss = 0.4535
Checking accuracy on validation set
Got 656 / 1000 correct (65.60)

Epoch 6, Iteration 500, loss = 0.3107
Checking accuracy on validation set
Got 659 / 1000 correct (65.90)

Epoch 6, Iteration 600, loss = 0.3786
Checking accuracy on validation set
Got 646 / 1000 correct (64.60)

Epoch 6, Iteration 700, loss = 0.4257
Checking accuracy on validation set
Got 665 / 1000 correct (66.50)

Epoch 7, Iteration 0, loss = 0.2928
Checking accuracy on validation set
Got 641 / 1000 correct (64.10)

Epoch 7, Iteration 100, loss = 0.2369
Checking accuracy on validation set
Got 649 / 1000 correct (64.90)

Epoch 7, Iteration 200, loss = 0.4301
Checking accuracy on validation set
Got 658 / 1000 correct (65.80)

Epoch 7, Iteration 300, loss = 0.2768
Checking accuracy on validation set
Got 647 / 1000 correct (64.70)

Epoch 7, Iteration 400, loss = 0.2547
Checking accuracy on validation set
Got 665 / 1000 correct (66.50)

Epoch 7, Iteration 500, loss = 0.1906
Checking accuracy on validation set
Got 646 / 1000 correct (64.60)

Epoch 7, Iteration 600, loss = 0.5434
Checking accuracy on validation set
Got 644 / 1000 correct (64.40)
```

```
Epoch 7, Iteration 700, loss = 0.1987
Checking accuracy on validation set
Got 654 / 1000 correct (65.40)

Epoch 8, Iteration 0, loss = 0.1638
Checking accuracy on validation set
Got 628 / 1000 correct (62.80)

Epoch 8, Iteration 100, loss = 0.0924
Checking accuracy on validation set
Got 656 / 1000 correct (65.60)

Epoch 8, Iteration 200, loss = 0.4765
Checking accuracy on validation set
Got 644 / 1000 correct (64.40)

Epoch 8, Iteration 300, loss = 0.1267
Checking accuracy on validation set
Got 648 / 1000 correct (64.80)

Epoch 8, Iteration 400, loss = 0.3426
Checking accuracy on validation set
Got 654 / 1000 correct (65.40)

Epoch 8, Iteration 500, loss = 0.2426
Checking accuracy on validation set
Got 654 / 1000 correct (65.40)

Epoch 8, Iteration 600, loss = 0.1685
Checking accuracy on validation set
Got 661 / 1000 correct (66.10)

Epoch 8, Iteration 700, loss = 0.2025
Checking accuracy on validation set
Got 644 / 1000 correct (64.40)

Epoch 9, Iteration 0, loss = 0.1441
Checking accuracy on validation set
Got 631 / 1000 correct (63.10)

Epoch 9, Iteration 100, loss = 0.2722
Checking accuracy on validation set
Got 660 / 1000 correct (66.00)

Epoch 9, Iteration 200, loss = 0.0577
Checking accuracy on validation set
Got 656 / 1000 correct (65.60)
```

```
Epoch 9, Iteration 300, loss = 0.0841
Checking accuracy on validation set
Got 645 / 1000 correct (64.50)

Epoch 9, Iteration 400, loss = 0.1870
Checking accuracy on validation set
Got 638 / 1000 correct (63.80)

Epoch 9, Iteration 500, loss = 0.3446
Checking accuracy on validation set
Got 646 / 1000 correct (64.60)

Epoch 9, Iteration 600, loss = 0.2645
Checking accuracy on validation set
Got 643 / 1000 correct (64.30)

Epoch 9, Iteration 700, loss = 0.1594
Checking accuracy on validation set
Got 639 / 1000 correct (63.90)

Maximum accuracy attained:  68.2
```

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.

[66]:

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

```
Checking accuracy on test set
Got 6367 / 10000 correct (63.67)
```

[66]: 63.67

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.

[67]:

```
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)
```

[68]:

```
learning_rate = 1e-3

model = ResNet10()

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

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

```
Epoch 0, Iteration 0, loss = 4.5434
Checking accuracy on validation set
Got 151 / 1000 correct (15.10)

Epoch 0, Iteration 100, loss = 1.4887
Checking accuracy on validation set
Got 388 / 1000 correct (38.80)

Epoch 0, Iteration 200, loss = 1.5201
Checking accuracy on validation set
Got 479 / 1000 correct (47.90)

Epoch 0, Iteration 300, loss = 1.2628
Checking accuracy on validation set
Got 486 / 1000 correct (48.60)

Epoch 0, Iteration 400, loss = 1.2302
Checking accuracy on validation set
Got 477 / 1000 correct (47.70)

Epoch 0, Iteration 500, loss = 0.9602
Checking accuracy on validation set
Got 530 / 1000 correct (53.00)
```

```
Epoch 0, Iteration 600, loss = 1.3069
Checking accuracy on validation set
Got 584 / 1000 correct (58.40)

Epoch 0, Iteration 700, loss = 1.2056
Checking accuracy on validation set
Got 582 / 1000 correct (58.20)

Epoch 1, Iteration 0, loss = 1.0182
Checking accuracy on validation set
Got 580 / 1000 correct (58.00)

Epoch 1, Iteration 100, loss = 1.0285
Checking accuracy on validation set
Got 581 / 1000 correct (58.10)

Epoch 1, Iteration 200, loss = 0.8674
Checking accuracy on validation set
Got 600 / 1000 correct (60.00)

Epoch 1, Iteration 300, loss = 1.0057
Checking accuracy on validation set
Got 568 / 1000 correct (56.80)

Epoch 1, Iteration 400, loss = 0.8202
Checking accuracy on validation set
Got 604 / 1000 correct (60.40)

Epoch 1, Iteration 500, loss = 1.1076
Checking accuracy on validation set
Got 655 / 1000 correct (65.50)

Epoch 1, Iteration 600, loss = 1.2029
Checking accuracy on validation set
Got 656 / 1000 correct (65.60)

Epoch 1, Iteration 700, loss = 1.0370
Checking accuracy on validation set
Got 680 / 1000 correct (68.00)

Epoch 2, Iteration 0, loss = 0.8561
Checking accuracy on validation set
Got 625 / 1000 correct (62.50)

Epoch 2, Iteration 100, loss = 0.8698
Checking accuracy on validation set
Got 683 / 1000 correct (68.30)
```

```
Epoch 2, Iteration 200, loss = 0.6480
Checking accuracy on validation set
Got 664 / 1000 correct (66.40)

Epoch 2, Iteration 300, loss = 0.9393
Checking accuracy on validation set
Got 676 / 1000 correct (67.60)

Epoch 2, Iteration 400, loss = 0.8605
Checking accuracy on validation set
Got 690 / 1000 correct (69.00)

Epoch 2, Iteration 500, loss = 0.8336
Checking accuracy on validation set
Got 673 / 1000 correct (67.30)

Epoch 2, Iteration 600, loss = 0.9443
Checking accuracy on validation set
Got 692 / 1000 correct (69.20)

Epoch 2, Iteration 700, loss = 0.8440
Checking accuracy on validation set
Got 704 / 1000 correct (70.40)

Epoch 3, Iteration 0, loss = 0.7921
Checking accuracy on validation set
Got 681 / 1000 correct (68.10)

Epoch 3, Iteration 100, loss = 0.6200
Checking accuracy on validation set
Got 699 / 1000 correct (69.90)

Epoch 3, Iteration 200, loss = 0.6603
Checking accuracy on validation set
Got 717 / 1000 correct (71.70)

Epoch 3, Iteration 300, loss = 0.5882
Checking accuracy on validation set
Got 721 / 1000 correct (72.10)

Epoch 3, Iteration 400, loss = 0.8736
Checking accuracy on validation set
Got 737 / 1000 correct (73.70)

Epoch 3, Iteration 500, loss = 0.7149
Checking accuracy on validation set
Got 730 / 1000 correct (73.00)
```

```
Epoch 3, Iteration 600, loss = 0.7857
Checking accuracy on validation set
Got 742 / 1000 correct (74.20)

Epoch 3, Iteration 700, loss = 0.7240
Checking accuracy on validation set
Got 739 / 1000 correct (73.90)

Epoch 4, Iteration 0, loss = 0.6703
Checking accuracy on validation set
Got 735 / 1000 correct (73.50)

Epoch 4, Iteration 100, loss = 0.5237
Checking accuracy on validation set
Got 731 / 1000 correct (73.10)

Epoch 4, Iteration 200, loss = 0.7656
Checking accuracy on validation set
Got 732 / 1000 correct (73.20)

Epoch 4, Iteration 300, loss = 0.8240
Checking accuracy on validation set
Got 724 / 1000 correct (72.40)

Epoch 4, Iteration 400, loss = 0.6238
Checking accuracy on validation set
Got 751 / 1000 correct (75.10)

Epoch 4, Iteration 500, loss = 0.8366
Checking accuracy on validation set
Got 738 / 1000 correct (73.80)

Epoch 4, Iteration 600, loss = 0.6932
Checking accuracy on validation set
Got 737 / 1000 correct (73.70)

Epoch 4, Iteration 700, loss = 0.7625
Checking accuracy on validation set
Got 741 / 1000 correct (74.10)

Epoch 5, Iteration 0, loss = 0.5276
Checking accuracy on validation set
Got 747 / 1000 correct (74.70)

Epoch 5, Iteration 100, loss = 0.5472
Checking accuracy on validation set
Got 752 / 1000 correct (75.20)
```

```
Epoch 5, Iteration 200, loss = 0.7943
Checking accuracy on validation set
Got 755 / 1000 correct (75.50)

Epoch 5, Iteration 300, loss = 0.6090
Checking accuracy on validation set
Got 753 / 1000 correct (75.30)

Epoch 5, Iteration 400, loss = 0.6784
Checking accuracy on validation set
Got 737 / 1000 correct (73.70)

Epoch 5, Iteration 500, loss = 0.6227
Checking accuracy on validation set
Got 750 / 1000 correct (75.00)

Epoch 5, Iteration 600, loss = 0.5892
Checking accuracy on validation set
Got 752 / 1000 correct (75.20)

Epoch 5, Iteration 700, loss = 0.4446
Checking accuracy on validation set
Got 731 / 1000 correct (73.10)

Epoch 6, Iteration 0, loss = 0.3672
Checking accuracy on validation set
Got 761 / 1000 correct (76.10)

Epoch 6, Iteration 100, loss = 0.3348
Checking accuracy on validation set
Got 758 / 1000 correct (75.80)

Epoch 6, Iteration 200, loss = 0.2626
Checking accuracy on validation set
Got 759 / 1000 correct (75.90)

Epoch 6, Iteration 300, loss = 0.6663
Checking accuracy on validation set
Got 751 / 1000 correct (75.10)

Epoch 6, Iteration 400, loss = 0.5963
Checking accuracy on validation set
Got 760 / 1000 correct (76.00)

Epoch 6, Iteration 500, loss = 0.6616
Checking accuracy on validation set
Got 768 / 1000 correct (76.80)
```

```
Epoch 6, Iteration 600, loss = 0.5670
Checking accuracy on validation set
Got 761 / 1000 correct (76.10)

Epoch 6, Iteration 700, loss = 0.5541
Checking accuracy on validation set
Got 782 / 1000 correct (78.20)

Epoch 7, Iteration 0, loss = 0.3806
Checking accuracy on validation set
Got 765 / 1000 correct (76.50)

Epoch 7, Iteration 100, loss = 0.3037
Checking accuracy on validation set
Got 768 / 1000 correct (76.80)

Epoch 7, Iteration 200, loss = 0.2902
Checking accuracy on validation set
Got 778 / 1000 correct (77.80)

Epoch 7, Iteration 300, loss = 0.7427
Checking accuracy on validation set
Got 762 / 1000 correct (76.20)

Epoch 7, Iteration 400, loss = 0.5025
Checking accuracy on validation set
Got 759 / 1000 correct (75.90)

Epoch 7, Iteration 500, loss = 0.5131
Checking accuracy on validation set
Got 758 / 1000 correct (75.80)

Epoch 7, Iteration 600, loss = 0.5793
Checking accuracy on validation set
Got 765 / 1000 correct (76.50)

Epoch 7, Iteration 700, loss = 0.4585
Checking accuracy on validation set
Got 773 / 1000 correct (77.30)

Epoch 8, Iteration 0, loss = 0.3855
Checking accuracy on validation set
Got 778 / 1000 correct (77.80)

Epoch 8, Iteration 100, loss = 0.1932
Checking accuracy on validation set
Got 777 / 1000 correct (77.70)
```

```
Epoch 8, Iteration 200, loss = 0.4004
Checking accuracy on validation set
Got 774 / 1000 correct (77.40)

Epoch 8, Iteration 300, loss = 0.5578
Checking accuracy on validation set
Got 779 / 1000 correct (77.90)

Epoch 8, Iteration 400, loss = 0.3639
Checking accuracy on validation set
Got 767 / 1000 correct (76.70)

Epoch 8, Iteration 500, loss = 0.5250
Checking accuracy on validation set
Got 748 / 1000 correct (74.80)

Epoch 8, Iteration 600, loss = 0.6792
Checking accuracy on validation set
Got 766 / 1000 correct (76.60)

Epoch 8, Iteration 700, loss = 0.3286
Checking accuracy on validation set
Got 757 / 1000 correct (75.70)

Epoch 9, Iteration 0, loss = 0.2836
Checking accuracy on validation set
Got 769 / 1000 correct (76.90)

Epoch 9, Iteration 100, loss = 0.3131
Checking accuracy on validation set
Got 766 / 1000 correct (76.60)

Epoch 9, Iteration 200, loss = 0.2703
Checking accuracy on validation set
Got 788 / 1000 correct (78.80)

Epoch 9, Iteration 300, loss = 0.2838
Checking accuracy on validation set
Got 776 / 1000 correct (77.60)

Epoch 9, Iteration 400, loss = 0.2914
Checking accuracy on validation set
Got 776 / 1000 correct (77.60)

Epoch 9, Iteration 500, loss = 0.3777
Checking accuracy on validation set
Got 781 / 1000 correct (78.10)
```

```
Epoch 9, Iteration 600, loss = 0.4068
Checking accuracy on validation set
Got 770 / 1000 correct (77.00)

Epoch 9, Iteration 700, loss = 0.2769
Checking accuracy on validation set
Got 773 / 1000 correct (77.30)

Maximum accuracy attained:  78.8
```

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.

[69]:

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

```
Checking accuracy on test set
Got 7430 / 10000 correct (74.30)
```

[69]:

```
74.3
```

2.14 Resnet with Attention (5 points)

[70]:

```
import torch
import torch.nn as nn

class ResNet_Attention(nn.Module):

    def __init__(self, block, layers, img_channels=3, num_classes=100,␣
 ↪batchnorm=False):
        super(ResNet_Attention, 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 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_Attention(num_classes = 100, batchnorm= False):

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

[71]:

```
## 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_Attention()
```

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

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

```
Epoch 0, Iteration 0, loss = 4.6222
Checking accuracy on validation set
Got 117 / 1000 correct (11.70)

Epoch 0, Iteration 100, loss = 1.4455
Checking accuracy on validation set
Got 399 / 1000 correct (39.90)

Epoch 0, Iteration 200, loss = 1.4561
Checking accuracy on validation set
Got 452 / 1000 correct (45.20)

Epoch 0, Iteration 300, loss = 1.4625
Checking accuracy on validation set
Got 496 / 1000 correct (49.60)

Epoch 0, Iteration 400, loss = 1.4097
Checking accuracy on validation set
Got 544 / 1000 correct (54.40)

Epoch 0, Iteration 500, loss = 1.5534
Checking accuracy on validation set
Got 546 / 1000 correct (54.60)

Epoch 0, Iteration 600, loss = 1.4604
Checking accuracy on validation set
Got 507 / 1000 correct (50.70)

Epoch 0, Iteration 700, loss = 1.2995
Checking accuracy on validation set
Got 560 / 1000 correct (56.00)

Epoch 1, Iteration 0, loss = 1.1074
Checking accuracy on validation set
Got 532 / 1000 correct (53.20)

Epoch 1, Iteration 100, loss = 1.1402
Checking accuracy on validation set
Got 621 / 1000 correct (62.10)

Epoch 1, Iteration 200, loss = 0.9059
Checking accuracy on validation set
```

```
Got 643 / 1000 correct (64.30)

Epoch 1, Iteration 300, loss = 0.9125
Checking accuracy on validation set
Got 607 / 1000 correct (60.70)

Epoch 1, Iteration 400, loss = 1.0332
Checking accuracy on validation set
Got 618 / 1000 correct (61.80)

Epoch 1, Iteration 500, loss = 1.0257
Checking accuracy on validation set
Got 635 / 1000 correct (63.50)

Epoch 1, Iteration 600, loss = 1.1687
Checking accuracy on validation set
Got 660 / 1000 correct (66.00)

Epoch 1, Iteration 700, loss = 0.8253
Checking accuracy on validation set
Got 656 / 1000 correct (65.60)

Epoch 2, Iteration 0, loss = 0.8601
Checking accuracy on validation set
Got 673 / 1000 correct (67.30)

Epoch 2, Iteration 100, loss = 0.7447
Checking accuracy on validation set
Got 694 / 1000 correct (69.40)

Epoch 2, Iteration 200, loss = 0.8928
Checking accuracy on validation set
Got 670 / 1000 correct (67.00)

Epoch 2, Iteration 300, loss = 1.0075
Checking accuracy on validation set
Got 675 / 1000 correct (67.50)

Epoch 2, Iteration 400, loss = 0.9007
Checking accuracy on validation set
Got 678 / 1000 correct (67.80)

Epoch 2, Iteration 500, loss = 0.8675
Checking accuracy on validation set
Got 667 / 1000 correct (66.70)

Epoch 2, Iteration 600, loss = 0.6820
Checking accuracy on validation set
```

```
Got 709 / 1000 correct (70.90)

Epoch 2, Iteration 700, loss = 0.7014
Checking accuracy on validation set
Got 690 / 1000 correct (69.00)

Epoch 3, Iteration 0, loss = 0.4932
Checking accuracy on validation set
Got 705 / 1000 correct (70.50)

Epoch 3, Iteration 100, loss = 0.4152
Checking accuracy on validation set
Got 691 / 1000 correct (69.10)

Epoch 3, Iteration 200, loss = 0.7111
Checking accuracy on validation set
Got 711 / 1000 correct (71.10)

Epoch 3, Iteration 300, loss = 0.7422
Checking accuracy on validation set
Got 686 / 1000 correct (68.60)

Epoch 3, Iteration 400, loss = 0.8809
Checking accuracy on validation set
Got 710 / 1000 correct (71.00)

Epoch 3, Iteration 500, loss = 0.6823
Checking accuracy on validation set
Got 710 / 1000 correct (71.00)

Epoch 3, Iteration 600, loss = 0.7089
Checking accuracy on validation set
Got 719 / 1000 correct (71.90)

Epoch 3, Iteration 700, loss = 0.5916
Checking accuracy on validation set
Got 741 / 1000 correct (74.10)

Epoch 4, Iteration 0, loss = 0.4461
Checking accuracy on validation set
Got 750 / 1000 correct (75.00)

Epoch 4, Iteration 100, loss = 0.5904
Checking accuracy on validation set
Got 722 / 1000 correct (72.20)

Epoch 4, Iteration 200, loss = 0.5736
Checking accuracy on validation set
```

```
Got 727 / 1000 correct (72.70)

Epoch 4, Iteration 300, loss = 0.6834
Checking accuracy on validation set
Got 747 / 1000 correct (74.70)

Epoch 4, Iteration 400, loss = 0.6924
Checking accuracy on validation set
Got 737 / 1000 correct (73.70)

Epoch 4, Iteration 500, loss = 0.6679
Checking accuracy on validation set
Got 756 / 1000 correct (75.60)

Epoch 4, Iteration 600, loss = 0.5853
Checking accuracy on validation set
Got 749 / 1000 correct (74.90)

Epoch 4, Iteration 700, loss = 0.3729
Checking accuracy on validation set
Got 765 / 1000 correct (76.50)

Epoch 5, Iteration 0, loss = 0.5974
Checking accuracy on validation set
Got 739 / 1000 correct (73.90)

Epoch 5, Iteration 100, loss = 0.6254
Checking accuracy on validation set
Got 761 / 1000 correct (76.10)

Epoch 5, Iteration 200, loss = 0.6146
Checking accuracy on validation set
Got 756 / 1000 correct (75.60)

Epoch 5, Iteration 300, loss = 0.3981
Checking accuracy on validation set
Got 748 / 1000 correct (74.80)

Epoch 5, Iteration 400, loss = 0.6244
Checking accuracy on validation set
Got 728 / 1000 correct (72.80)

Epoch 5, Iteration 500, loss = 0.5604
Checking accuracy on validation set
Got 769 / 1000 correct (76.90)

Epoch 5, Iteration 600, loss = 0.5282
Checking accuracy on validation set
```

```
Got 733 / 1000 correct (73.30)

Epoch 5, Iteration 700, loss = 0.3662
Checking accuracy on validation set
Got 774 / 1000 correct (77.40)

Epoch 6, Iteration 0, loss = 0.4832
Checking accuracy on validation set
Got 772 / 1000 correct (77.20)

Epoch 6, Iteration 100, loss = 0.5003
Checking accuracy on validation set
Got 765 / 1000 correct (76.50)

Epoch 6, Iteration 200, loss = 0.3064
Checking accuracy on validation set
Got 742 / 1000 correct (74.20)

Epoch 6, Iteration 300, loss = 0.4395
Checking accuracy on validation set
Got 774 / 1000 correct (77.40)

Epoch 6, Iteration 400, loss = 0.4166
Checking accuracy on validation set
Got 769 / 1000 correct (76.90)

Epoch 6, Iteration 500, loss = 0.5721
Checking accuracy on validation set
Got 758 / 1000 correct (75.80)

Epoch 6, Iteration 600, loss = 0.4389
Checking accuracy on validation set
Got 763 / 1000 correct (76.30)

Epoch 6, Iteration 700, loss = 0.3792
Checking accuracy on validation set
Got 763 / 1000 correct (76.30)

Epoch 7, Iteration 0, loss = 0.2246
Checking accuracy on validation set
Got 773 / 1000 correct (77.30)

Epoch 7, Iteration 100, loss = 0.3967
Checking accuracy on validation set
Got 758 / 1000 correct (75.80)

Epoch 7, Iteration 200, loss = 0.2013
Checking accuracy on validation set
```

```
Got 777 / 1000 correct (77.70)

Epoch 7, Iteration 300, loss = 0.5103
Checking accuracy on validation set
Got 790 / 1000 correct (79.00)

Epoch 7, Iteration 400, loss = 0.3722
Checking accuracy on validation set
Got 764 / 1000 correct (76.40)

Epoch 7, Iteration 500, loss = 0.4503
Checking accuracy on validation set
Got 770 / 1000 correct (77.00)

Epoch 7, Iteration 600, loss = 0.3566
Checking accuracy on validation set
Got 766 / 1000 correct (76.60)

Epoch 7, Iteration 700, loss = 0.2483
Checking accuracy on validation set
Got 766 / 1000 correct (76.60)

Epoch 8, Iteration 0, loss = 0.3512
Checking accuracy on validation set
Got 779 / 1000 correct (77.90)

Epoch 8, Iteration 100, loss = 0.3850
Checking accuracy on validation set
Got 797 / 1000 correct (79.70)

Epoch 8, Iteration 200, loss = 0.1385
Checking accuracy on validation set
Got 789 / 1000 correct (78.90)

Epoch 8, Iteration 300, loss = 0.3697
Checking accuracy on validation set
Got 782 / 1000 correct (78.20)

Epoch 8, Iteration 400, loss = 0.2100
Checking accuracy on validation set
Got 758 / 1000 correct (75.80)

Epoch 8, Iteration 500, loss = 0.3651
Checking accuracy on validation set
Got 775 / 1000 correct (77.50)

Epoch 8, Iteration 600, loss = 0.5510
Checking accuracy on validation set
```

```
Got 781 / 1000 correct (78.10)

Epoch 8, Iteration 700, loss = 0.4757
Checking accuracy on validation set
Got 772 / 1000 correct (77.20)

Epoch 9, Iteration 0, loss = 0.1594
Checking accuracy on validation set
Got 789 / 1000 correct (78.90)

Epoch 9, Iteration 100, loss = 0.1351
Checking accuracy on validation set
Got 783 / 1000 correct (78.30)

Epoch 9, Iteration 200, loss = 0.2040
Checking accuracy on validation set
Got 777 / 1000 correct (77.70)

Epoch 9, Iteration 300, loss = 0.2327
Checking accuracy on validation set
Got 773 / 1000 correct (77.30)

Epoch 9, Iteration 400, loss = 0.2803
Checking accuracy on validation set
Got 786 / 1000 correct (78.60)

Epoch 9, Iteration 500, loss = 0.3249
Checking accuracy on validation set
Got 768 / 1000 correct (76.80)

Epoch 9, Iteration 600, loss = 0.2418
Checking accuracy on validation set
Got 778 / 1000 correct (77.80)

Epoch 9, Iteration 700, loss = 0.3013
Checking accuracy on validation set
Got 780 / 1000 correct (78.00)

Maximum accuracy attained:  79.7
```

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.

[72]:
```
AttentionResnet = model
check_accuracy(loader_test, AttentionResnet)
```

```
Checking accuracy on test set
Got 7656 / 10000 correct (76.56)
```

```
[72]: 76.55999999999999
```

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. Attention ResNet10 (75.8%, 76.98%, 76.56%)
2. Vanilla ResNet10 (74.59%, 74.94%, 74.3%)
3. Double Attention Blocks CNN (61.91%, 62.42%, 63.67%)
4. Single Early Attention CNN (60.5%, 60.91%, 60.08%, 61.26%, 61.43%) - avg: 60.84%
5. Single Late Attention CNN (60.18%, 61.29%, 60.04%, 59.05%, 61.74%) - avg: 60.46%
6. Vanilla CNN (57.25%, 59.4%, 57.98%)

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

Your Answer: Residual Network provids a skip connection between every two layers while all layers are connected directly. This architecture of residual block allows the network to learn parameters from more deeper layers than classical CNN which may face issues of gradient vanishing when learning from deep layers. Based on that, ResNet10 has overall better performance than the Vanilla CNN has. Attention mechanism can lead model to pay "attention" to specific features which increases the accuracy as well. The ranks of single early attention and single late attention are hard to determined since the difference of accuracy between two stategies is relatively low. On the other hand, the double attention blocks indeed have better improvement on accuracy than the single attention block has.

```
[ ]:
```