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.

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.

```
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
```

```
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=False,
                             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
```

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.

4
```
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
```

PyTorch Tensors: Flatten Function

A PyTorch Tensor is conceptionally similar to a numpy array: it is an n-dimensional grid of numbers, and like numpy PyTorch provides many functions to efficiently operate on Tensors. As a simple example, we provide a `flatten` function below which reshapes image data for use in a fully-connected neural network.

Recall that image data is typically stored in a Tensor of shape N x C x H x W, where:

- N is the number of datapoints
- C is the number of channels
- H is the height of the intermediate feature map in pixels
- W is the height of the intermediate feature map in pixels

This is the right way to represent the data when we are doing something like a 2D convolution, that needs spatial understanding of where the intermediate features are relative to each other. When we use fully connected affine layers to process the image, however, we want each datapoint to be represented by a single vector -- it's no longer useful to segregate the different channels, rows, and columns of the data. So, we use a "flatten" operation to collapse the `C x H x W` values per representation into a single long vector. The flatten function below first reads in the N, C, H, and W values from a given batch of data, and then returns a "view" of that data. "View" is analogous to numpy's "reshape" method: it reshapes x's dimensions to be N x ??, where ?? is allowed to be anything (in this case, it will be C x H x W, but we don't need to specify that explicitly).

```
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]])
```

Check Accuracy Function

When training the model we will use the following function to check the accuracy of our model on the training or validation sets.

When checking accuracy we don't need to compute any gradients; as a result we don't need PyTorch to build a computational graph for us when we compute scores. To prevent a graph from being built we scope our computation under a `torch.no_grad()` context manager.

```
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
```

Training Loop

We also use a slightly different training loop. Rather than updating the values of the weights ourselves, we use an Optimizer object from the `torch.optim` package, which abstract the notion

of an optimization algorithm and provides implementations of most of the algorithms commonly used to optimize neural networks.

```
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)
```

```
# 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)
```

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.

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

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, 10),
)

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


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

```
Epoch 0, Iteration 0, loss = 2.3219
Checking accuracy on validation set
Got 134 / 1000 correct (13.40)

Epoch 0, Iteration 100, loss = 1.7334
Checking accuracy on validation set
Got 435 / 1000 correct (43.50)

Epoch 0, Iteration 200, loss = 1.5357
Checking accuracy on validation set
Got 501 / 1000 correct (50.10)

Epoch 0, Iteration 300, loss = 1.2209
Checking accuracy on validation set
Got 539 / 1000 correct (53.90)

Epoch 0, Iteration 400, loss = 1.2170
Checking accuracy on validation set
Got 539 / 1000 correct (53.90)

Epoch 0, Iteration 500, loss = 1.2681
Checking accuracy on validation set
Got 560 / 1000 correct (56.00)

Epoch 0, Iteration 600, loss = 1.3910
Checking accuracy on validation set
Got 566 / 1000 correct (56.60)

Epoch 0, Iteration 700, loss = 1.3100
Checking accuracy on validation set
Got 579 / 1000 correct (57.90)

Epoch 1, Iteration 0, loss = 1.2064
Checking accuracy on validation set
Got 569 / 1000 correct (56.90)

Epoch 1, Iteration 100, loss = 1.0346
Checking accuracy on validation set
Got 555 / 1000 correct (55.50)

Epoch 1, Iteration 200, loss = 1.2041
Checking accuracy on validation set
Got 585 / 1000 correct (58.50)

Epoch 1, Iteration 300, loss = 1.1552
Checking accuracy on validation set
Got 596 / 1000 correct (59.60)
```

```
Epoch 1, Iteration 400, loss = 0.8565
Checking accuracy on validation set
Got 615 / 1000 correct (61.50)

Epoch 1, Iteration 500, loss = 0.7920
Checking accuracy on validation set
Got 612 / 1000 correct (61.20)

Epoch 1, Iteration 600, loss = 1.1695
Checking accuracy on validation set
Got 618 / 1000 correct (61.80)

Epoch 1, Iteration 700, loss = 0.9619
Checking accuracy on validation set
Got 634 / 1000 correct (63.40)

Epoch 2, Iteration 0, loss = 0.9319
Checking accuracy on validation set
Got 622 / 1000 correct (62.20)

Epoch 2, Iteration 100, loss = 0.8656
Checking accuracy on validation set
Got 630 / 1000 correct (63.00)

Epoch 2, Iteration 200, loss = 0.7423
Checking accuracy on validation set
Got 633 / 1000 correct (63.30)

Epoch 2, Iteration 300, loss = 0.6828
Checking accuracy on validation set
Got 615 / 1000 correct (61.50)

Epoch 2, Iteration 400, loss = 0.9887
Checking accuracy on validation set
Got 628 / 1000 correct (62.80)

Epoch 2, Iteration 500, loss = 0.8066
Checking accuracy on validation set
Got 614 / 1000 correct (61.40)

Epoch 2, Iteration 600, loss = 0.6559
Checking accuracy on validation set
Got 628 / 1000 correct (62.80)

Epoch 2, Iteration 700, loss = 0.9575
Checking accuracy on validation set
Got 643 / 1000 correct (64.30)

Epoch 3, Iteration 0, loss = 0.7157
Checking accuracy on validation set
Got 627 / 1000 correct (62.70)

Epoch 3, Iteration 100, loss = 0.7747
Checking accuracy on validation set
Got 628 / 1000 correct (62.80)

Epoch 3, Iteration 200, loss = 0.9543
Checking accuracy on validation set
Got 635 / 1000 correct (63.50)

Epoch 3, Iteration 300, loss = 0.6696
Checking accuracy on validation set
Got 635 / 1000 correct (63.50)
```

```
Epoch 3, Iteration 400, loss = 0.7516
Checking accuracy on validation set
Got 640 / 1000 correct (64.00)

Epoch 3, Iteration 500, loss = 0.6206
Checking accuracy on validation set
Got 636 / 1000 correct (63.60)

Epoch 3, Iteration 600, loss = 0.5516
Checking accuracy on validation set
Got 628 / 1000 correct (62.80)

Epoch 3, Iteration 700, loss = 0.7822
Checking accuracy on validation set
Got 613 / 1000 correct (61.30)

Epoch 4, Iteration 0, loss = 0.5412
Checking accuracy on validation set
Got 641 / 1000 correct (64.10)

Epoch 4, Iteration 100, loss = 0.7038
Checking accuracy on validation set
Got 651 / 1000 correct (65.10)

Epoch 4, Iteration 200, loss = 0.6516
Checking accuracy on validation set
Got 650 / 1000 correct (65.00)

Epoch 4, Iteration 300, loss = 0.6249
Checking accuracy on validation set
Got 601 / 1000 correct (60.10)

Epoch 4, Iteration 400, loss = 0.6779
Checking accuracy on validation set
Got 620 / 1000 correct (62.00)

Epoch 4, Iteration 500, loss = 0.7307
Checking accuracy on validation set
Got 629 / 1000 correct (62.90)

Epoch 4, Iteration 600, loss = 0.5662
Checking accuracy on validation set
Got 637 / 1000 correct (63.70)

Epoch 4, Iteration 700, loss = 0.8132
Checking accuracy on validation set
Got 633 / 1000 correct (63.30)

Epoch 5, Iteration 0, loss = 0.4583
Checking accuracy on validation set
Got 621 / 1000 correct (62.10)

Epoch 5, Iteration 100, loss = 0.4782
Checking accuracy on validation set
Got 626 / 1000 correct (62.60)

Epoch 5, Iteration 200, loss = 0.5840
Checking accuracy on validation set
Got 617 / 1000 correct (61.70)

Epoch 5, Iteration 300, loss = 0.7485
Checking accuracy on validation set
Got 613 / 1000 correct (61.30)

Epoch 5, Iteration 400, loss = 0.5841
```

```
Checking accuracy on validation set
Got 628 / 1000 correct (62.80)

Epoch 5, Iteration 500, loss = 0.5292
Checking accuracy on validation set
Got 601 / 1000 correct (60.10)

Epoch 5, Iteration 600, loss = 0.6805
Checking accuracy on validation set
Got 617 / 1000 correct (61.70)

Epoch 5, Iteration 700, loss = 0.4854
Checking accuracy on validation set
Got 616 / 1000 correct (61.60)

Epoch 6, Iteration 0, loss = 0.3152
Checking accuracy on validation set
Got 606 / 1000 correct (60.60)

Epoch 6, Iteration 100, loss = 0.2755
Checking accuracy on validation set
Got 616 / 1000 correct (61.60)

Epoch 6, Iteration 200, loss = 0.4328
Checking accuracy on validation set
Got 602 / 1000 correct (60.20)

Epoch 6, Iteration 300, loss = 0.4634
Checking accuracy on validation set
Got 615 / 1000 correct (61.50)

Epoch 6, Iteration 400, loss = 0.7025
Checking accuracy on validation set
Got 621 / 1000 correct (62.10)

Epoch 6, Iteration 500, loss = 0.6328
Checking accuracy on validation set
Got 617 / 1000 correct (61.70)

Epoch 6, Iteration 600, loss = 0.6804
Checking accuracy on validation set
Got 609 / 1000 correct (60.90)

Epoch 6, Iteration 700, loss = 0.3947
Checking accuracy on validation set
Got 614 / 1000 correct (61.40)

Epoch 7, Iteration 0, loss = 0.3245
Checking accuracy on validation set
Got 600 / 1000 correct (60.00)

Epoch 7, Iteration 100, loss = 0.2824
Checking accuracy on validation set
Got 625 / 1000 correct (62.50)

Epoch 7, Iteration 200, loss = 0.4617
Checking accuracy on validation set
Got 607 / 1000 correct (60.70)

Epoch 7, Iteration 300, loss = 0.3314
Checking accuracy on validation set
Got 599 / 1000 correct (59.90)

Epoch 7, Iteration 400, loss = 0.3954
Checking accuracy on validation set
```

```
Got 610 / 1000 correct (61.00)

Epoch 7, Iteration 500, loss = 0.5896
Checking accuracy on validation set
Got 600 / 1000 correct (60.00)

Epoch 7, Iteration 600, loss = 0.2911
Checking accuracy on validation set
Got 611 / 1000 correct (61.10)

Epoch 7, Iteration 700, loss = 0.6089
Checking accuracy on validation set
Got 597 / 1000 correct (59.70)

Epoch 8, Iteration 0, loss = 0.2767
Checking accuracy on validation set
Got 611 / 1000 correct (61.10)

Epoch 8, Iteration 100, loss = 0.3303
Checking accuracy on validation set
Got 600 / 1000 correct (60.00)

Epoch 8, Iteration 200, loss = 0.2689
Checking accuracy on validation set
Got 611 / 1000 correct (61.10)

Epoch 8, Iteration 300, loss = 0.4160
Checking accuracy on validation set
Got 601 / 1000 correct (60.10)

Epoch 8, Iteration 400, loss = 0.3391
Checking accuracy on validation set
Got 604 / 1000 correct (60.40)

Epoch 8, Iteration 500, loss = 0.3664
Checking accuracy on validation set
Got 595 / 1000 correct (59.50)

Epoch 8, Iteration 600, loss = 0.3796
Checking accuracy on validation set
Got 600 / 1000 correct (60.00)

Epoch 8, Iteration 700, loss = 0.3891
Checking accuracy on validation set
Got 591 / 1000 correct (59.10)

Epoch 9, Iteration 0, loss = 0.2017
Checking accuracy on validation set
Got 605 / 1000 correct (60.50)

Epoch 9, Iteration 100, loss = 0.2164
Checking accuracy on validation set
Got 605 / 1000 correct (60.50)

Epoch 9, Iteration 200, loss = 0.2009
Checking accuracy on validation set
Got 605 / 1000 correct (60.50)

Epoch 9, Iteration 300, loss = 0.3066
Checking accuracy on validation set
Got 596 / 1000 correct (59.60)

Epoch 9, Iteration 400, loss = 0.1967
Checking accuracy on validation set
Got 603 / 1000 correct (60.30)
```

```
Epoch 9, Iteration 500, loss = 0.2879
Checking accuracy on validation set
Got 616 / 1000 correct (61.60)

Epoch 9, Iteration 600, loss = 0.1661
Checking accuracy on validation set
Got 601 / 1000 correct (60.10)

Epoch 9, Iteration 700, loss = 0.3692
Checking accuracy on validation set
Got 613 / 1000 correct (61.30)

Maximum accuracy attained:  65.10000000000001
```

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 at least 55% accuracy

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

```
Checking accuracy on test set
Got 5977 / 10000 correct (59.77)
134 59.77
```

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: $Attention(X) = X + Softmax(\frac{XW_Q(XW_K)^\top}{\sqrt{C}})XW_V$

Inline Question 1: Self-Attention is equivalent to which of the following: (10% of the grade)

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

Your Answer: 2. Non-local means k-means is a clustering algorithm. Residual add skip connection, but it does not provide attention using ScaledDotProductAttention. Gaussian blurring is like the reverse of attention, because it averages out the local part.

Non-local means is a filtering algorithm that computes a weighted mean of all pixels in an image. It allows distant pixels to contribute to the filtered response at a location based on patch appearance similarity. This is same as the self-attention mechanism which utilies the similarity of pixels at every location.

Here you implement the Attention module, and run it in the next section

```
135 # 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)
            self.conv_key = nn.Conv2d(in_channels,in_channels,kernel_size=1)
            self.conv_value = nn.Conv2d(in_channels,in_channels,kernel_size=1)

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

            # TODO: Pass the input through conv_query, reshape the output volume to (N, C, H*W)
            q = self.conv_query(x)
            q = q.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)
            k = k.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)
            v = v.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.sqrt(torch.Tensor([C])).to('cuda')

            attention = F.softmax(q @ torch.transpose(k,1,2) / temp, dim=-1)

            # print(attention.shape)
            attention = attention @ v

            # Reshape the output to (N, C, H, W) before adding to the input volume
            attention = torch.reshape(attention, (N, C, H, W))
```

```
        return x + attention
```

Single Attention Block: Early attention; After the first conv layer. (50% of the grade)

```
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 = 2.3331
Checking accuracy on validation set
Got 105 / 1000 correct (10.50)

Epoch 0, Iteration 100, loss = 1.6872
Checking accuracy on validation set
Got 433 / 1000 correct (43.30)

Epoch 0, Iteration 200, loss = 1.3894
Checking accuracy on validation set
Got 476 / 1000 correct (47.60)

Epoch 0, Iteration 300, loss = 1.4099
Checking accuracy on validation set
Got 545 / 1000 correct (54.50)

Epoch 0, Iteration 400, loss = 1.3013
Checking accuracy on validation set
Got 536 / 1000 correct (53.60)

Epoch 0, Iteration 500, loss = 1.4612
Checking accuracy on validation set
Got 554 / 1000 correct (55.40)

Epoch 0, Iteration 600, loss = 1.0938
Checking accuracy on validation set
Got 566 / 1000 correct (56.60)

Epoch 0, Iteration 700, loss = 1.0170
Checking accuracy on validation set
Got 573 / 1000 correct (57.30)

Epoch 1, Iteration 0, loss = 1.0923
```

```
Checking accuracy on validation set
Got 556 / 1000 correct (55.60)

Epoch 1, Iteration 100, loss = 1.1667
Checking accuracy on validation set
Got 594 / 1000 correct (59.40)

Epoch 1, Iteration 200, loss = 1.0249
Checking accuracy on validation set
Got 581 / 1000 correct (58.10)

Epoch 1, Iteration 300, loss = 1.0044
Checking accuracy on validation set
Got 604 / 1000 correct (60.40)

Epoch 1, Iteration 400, loss = 1.0689
Checking accuracy on validation set
Got 599 / 1000 correct (59.90)

Epoch 1, Iteration 500, loss = 1.0359
Checking accuracy on validation set
Got 609 / 1000 correct (60.90)

Epoch 1, Iteration 600, loss = 1.0947
Checking accuracy on validation set
Got 607 / 1000 correct (60.70)

Epoch 1, Iteration 700, loss = 1.0123
Checking accuracy on validation set
Got 625 / 1000 correct (62.50)

Epoch 2, Iteration 0, loss = 1.0454
Checking accuracy on validation set
Got 621 / 1000 correct (62.10)

Epoch 2, Iteration 100, loss = 0.8884
Checking accuracy on validation set
Got 606 / 1000 correct (60.60)

Epoch 2, Iteration 200, loss = 1.0927
Checking accuracy on validation set
Got 638 / 1000 correct (63.80)

Epoch 2, Iteration 300, loss = 0.8120
Checking accuracy on validation set
Got 632 / 1000 correct (63.20)

Epoch 2, Iteration 400, loss = 0.9423
Checking accuracy on validation set
Got 626 / 1000 correct (62.60)

Epoch 2, Iteration 500, loss = 0.8851
Checking accuracy on validation set
Got 638 / 1000 correct (63.80)

Epoch 2, Iteration 600, loss = 0.7478
Checking accuracy on validation set
Got 629 / 1000 correct (62.90)

Epoch 2, Iteration 700, loss = 0.9841
Checking accuracy on validation set
Got 628 / 1000 correct (62.80)

Epoch 3, Iteration 0, loss = 0.6506
Checking accuracy on validation set
```

```
Got 636 / 1000 correct (63.60)

Epoch 3, Iteration 100, loss = 0.5787
Checking accuracy on validation set
Got 643 / 1000 correct (64.30)

Epoch 3, Iteration 200, loss = 0.5650
Checking accuracy on validation set
Got 644 / 1000 correct (64.40)

Epoch 3, Iteration 300, loss = 0.8680
Checking accuracy on validation set
Got 640 / 1000 correct (64.00)

Epoch 3, Iteration 400, loss = 0.7739
Checking accuracy on validation set
Got 633 / 1000 correct (63.30)

Epoch 3, Iteration 500, loss = 0.7801
Checking accuracy on validation set
Got 643 / 1000 correct (64.30)

Epoch 3, Iteration 600, loss = 0.6830
Checking accuracy on validation set
Got 634 / 1000 correct (63.40)

Epoch 3, Iteration 700, loss = 0.8111
Checking accuracy on validation set
Got 629 / 1000 correct (62.90)

Epoch 4, Iteration 0, loss = 0.4444
Checking accuracy on validation set
Got 630 / 1000 correct (63.00)

Epoch 4, Iteration 100, loss = 0.6933
Checking accuracy on validation set
Got 635 / 1000 correct (63.50)

Epoch 4, Iteration 200, loss = 0.3616
Checking accuracy on validation set
Got 631 / 1000 correct (63.10)

Epoch 4, Iteration 300, loss = 0.6321
Checking accuracy on validation set
Got 647 / 1000 correct (64.70)

Epoch 4, Iteration 400, loss = 0.3967
Checking accuracy on validation set
Got 638 / 1000 correct (63.80)

Epoch 4, Iteration 500, loss = 0.6226
Checking accuracy on validation set
Got 649 / 1000 correct (64.90)

Epoch 4, Iteration 600, loss = 0.6335
Checking accuracy on validation set
Got 637 / 1000 correct (63.70)

Epoch 4, Iteration 700, loss = 0.5533
Checking accuracy on validation set
Got 617 / 1000 correct (61.70)

Epoch 5, Iteration 0, loss = 0.3613
Checking accuracy on validation set
Got 630 / 1000 correct (63.00)
```

```
Epoch 5, Iteration 100, loss = 0.3916
Checking accuracy on validation set
Got 649 / 1000 correct (64.90)

Epoch 5, Iteration 200, loss = 0.3433
Checking accuracy on validation set
Got 622 / 1000 correct (62.20)

Epoch 5, Iteration 300, loss = 0.3913
Checking accuracy on validation set
Got 616 / 1000 correct (61.60)

Epoch 5, Iteration 400, loss = 0.5281
Checking accuracy on validation set
Got 629 / 1000 correct (62.90)

Epoch 5, Iteration 500, loss = 0.4905
Checking accuracy on validation set
Got 627 / 1000 correct (62.70)

Epoch 5, Iteration 600, loss = 0.5372
Checking accuracy on validation set
Got 617 / 1000 correct (61.70)

Epoch 5, Iteration 700, loss = 0.3988
Checking accuracy on validation set
Got 631 / 1000 correct (63.10)

Epoch 6, Iteration 0, loss = 0.2265
Checking accuracy on validation set
Got 629 / 1000 correct (62.90)

Epoch 6, Iteration 100, loss = 0.2397
Checking accuracy on validation set
Got 634 / 1000 correct (63.40)

Epoch 6, Iteration 200, loss = 0.1149
Checking accuracy on validation set
Got 623 / 1000 correct (62.30)

Epoch 6, Iteration 300, loss = 0.2076
Checking accuracy on validation set
Got 612 / 1000 correct (61.20)

Epoch 6, Iteration 400, loss = 0.1378
Checking accuracy on validation set
Got 622 / 1000 correct (62.20)

Epoch 6, Iteration 500, loss = 0.2051
Checking accuracy on validation set
Got 620 / 1000 correct (62.00)

Epoch 6, Iteration 600, loss = 0.2435
Checking accuracy on validation set
Got 625 / 1000 correct (62.50)

Epoch 6, Iteration 700, loss = 0.2940
Checking accuracy on validation set
Got 618 / 1000 correct (61.80)

Epoch 7, Iteration 0, loss = 0.1138
Checking accuracy on validation set
Got 613 / 1000 correct (61.30)
```

```
Epoch 7, Iteration 100, loss = 0.2130
Checking accuracy on validation set
Got 629 / 1000 correct (62.90)

Epoch 7, Iteration 200, loss = 0.0965
Checking accuracy on validation set
Got 625 / 1000 correct (62.50)

Epoch 7, Iteration 300, loss = 0.1632
Checking accuracy on validation set
Got 627 / 1000 correct (62.70)

Epoch 7, Iteration 400, loss = 0.0929
Checking accuracy on validation set
Got 622 / 1000 correct (62.20)

Epoch 7, Iteration 500, loss = 0.1742
Checking accuracy on validation set
Got 626 / 1000 correct (62.60)

Epoch 7, Iteration 600, loss = 0.1590
Checking accuracy on validation set
Got 618 / 1000 correct (61.80)

Epoch 7, Iteration 700, loss = 0.1879
Checking accuracy on validation set
Got 624 / 1000 correct (62.40)

Epoch 8, Iteration 0, loss = 0.1327
Checking accuracy on validation set
Got 623 / 1000 correct (62.30)

Epoch 8, Iteration 100, loss = 0.0703
Checking accuracy on validation set
Got 623 / 1000 correct (62.30)

Epoch 8, Iteration 200, loss = 0.0711
Checking accuracy on validation set
Got 618 / 1000 correct (61.80)

Epoch 8, Iteration 300, loss = 0.0486
Checking accuracy on validation set
Got 615 / 1000 correct (61.50)

Epoch 8, Iteration 400, loss = 0.1292
Checking accuracy on validation set
Got 620 / 1000 correct (62.00)

Epoch 8, Iteration 500, loss = 0.0943
Checking accuracy on validation set
Got 633 / 1000 correct (63.30)

Epoch 8, Iteration 600, loss = 0.1549
Checking accuracy on validation set
Got 613 / 1000 correct (61.30)

Epoch 8, Iteration 700, loss = 0.1220
Checking accuracy on validation set
Got 619 / 1000 correct (61.90)

Epoch 9, Iteration 0, loss = 0.1088
Checking accuracy on validation set
Got 618 / 1000 correct (61.80)

Epoch 9, Iteration 100, loss = 0.0309
```

```
Checking accuracy on validation set
Got 613 / 1000 correct (61.30)

Epoch 9, Iteration 200, loss = 0.0707
Checking accuracy on validation set
Got 625 / 1000 correct (62.50)

Epoch 9, Iteration 300, loss = 0.1041
Checking accuracy on validation set
Got 626 / 1000 correct (62.60)

Epoch 9, Iteration 400, loss = 0.0920
Checking accuracy on validation set
Got 614 / 1000 correct (61.40)

Epoch 9, Iteration 500, loss = 0.1300
Checking accuracy on validation set
Got 625 / 1000 correct (62.50)

Epoch 9, Iteration 600, loss = 0.0790
Checking accuracy on validation set
Got 623 / 1000 correct (62.30)

Epoch 9, Iteration 700, loss = 0.0333
Checking accuracy on validation set
Got 619 / 1000 correct (61.90)

Maximum accuracy attained:  64.9
```

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. *

137
```
earlyAttention = model
check_accuracy(loader_test, earlyAttention)
```

```
Checking accuracy on test set
Got 6079 / 10000 correct (60.79)
```

137
```
60.79
```

Single Attention Block: Late attention; After the second conv layer. (10% of the grade)

138
```
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 = 2.2973
Checking accuracy on validation set
Got 130 / 1000 correct (13.00)

Epoch 0, Iteration 100, loss = 1.3413
Checking accuracy on validation set
Got 454 / 1000 correct (45.40)

Epoch 0, Iteration 200, loss = 1.3692
Checking accuracy on validation set
Got 491 / 1000 correct (49.10)

Epoch 0, Iteration 300, loss = 1.4121
Checking accuracy on validation set
Got 525 / 1000 correct (52.50)

Epoch 0, Iteration 400, loss = 1.0802
Checking accuracy on validation set
Got 565 / 1000 correct (56.50)

Epoch 0, Iteration 500, loss = 1.2060
Checking accuracy on validation set
Got 566 / 1000 correct (56.60)

Epoch 0, Iteration 600, loss = 1.1976
Checking accuracy on validation set
Got 580 / 1000 correct (58.00)

Epoch 0, Iteration 700, loss = 1.2609
Checking accuracy on validation set
Got 609 / 1000 correct (60.90)

Epoch 1, Iteration 0, loss = 1.0984
Checking accuracy on validation set
Got 611 / 1000 correct (61.10)

Epoch 1, Iteration 100, loss = 1.0881
Checking accuracy on validation set
Got 617 / 1000 correct (61.70)

Epoch 1, Iteration 200, loss = 0.7825
Checking accuracy on validation set
Got 598 / 1000 correct (59.80)

Epoch 1, Iteration 300, loss = 0.8874
Checking accuracy on validation set
Got 601 / 1000 correct (60.10)

Epoch 1, Iteration 400, loss = 0.9824
Checking accuracy on validation set
Got 629 / 1000 correct (62.90)

Epoch 1, Iteration 500, loss = 1.0145
Checking accuracy on validation set
```

```
Got 621 / 1000 correct (62.10)

Epoch 1, Iteration 600, loss = 1.3908
Checking accuracy on validation set
Got 638 / 1000 correct (63.80)

Epoch 1, Iteration 700, loss = 0.8114
Checking accuracy on validation set
Got 651 / 1000 correct (65.10)

Epoch 2, Iteration 0, loss = 0.6418
Checking accuracy on validation set
Got 645 / 1000 correct (64.50)

Epoch 2, Iteration 100, loss = 0.8578
Checking accuracy on validation set
Got 628 / 1000 correct (62.80)

Epoch 2, Iteration 200, loss = 0.6747
Checking accuracy on validation set
Got 652 / 1000 correct (65.20)

Epoch 2, Iteration 300, loss = 0.8210
Checking accuracy on validation set
Got 657 / 1000 correct (65.70)

Epoch 2, Iteration 400, loss = 0.6470
Checking accuracy on validation set
Got 660 / 1000 correct (66.00)

Epoch 2, Iteration 500, loss = 0.7945
Checking accuracy on validation set
Got 655 / 1000 correct (65.50)

Epoch 2, Iteration 600, loss = 0.8036
Checking accuracy on validation set
Got 670 / 1000 correct (67.00)

Epoch 2, Iteration 700, loss = 0.9375
Checking accuracy on validation set
Got 656 / 1000 correct (65.60)

Epoch 3, Iteration 0, loss = 0.7084
Checking accuracy on validation set
Got 667 / 1000 correct (66.70)

Epoch 3, Iteration 100, loss = 0.5526
Checking accuracy on validation set
Got 653 / 1000 correct (65.30)

Epoch 3, Iteration 200, loss = 0.6307
Checking accuracy on validation set
Got 647 / 1000 correct (64.70)

Epoch 3, Iteration 300, loss = 0.7853
Checking accuracy on validation set
Got 652 / 1000 correct (65.20)

Epoch 3, Iteration 400, loss = 0.6132
Checking accuracy on validation set
Got 657 / 1000 correct (65.70)

Epoch 3, Iteration 500, loss = 0.6862
Checking accuracy on validation set
Got 638 / 1000 correct (63.80)
```

```
Epoch 3, Iteration 600, loss = 0.6121
Checking accuracy on validation set
Got 622 / 1000 correct (62.20)

Epoch 3, Iteration 700, loss = 0.6392
Checking accuracy on validation set
Got 674 / 1000 correct (67.40)

Epoch 4, Iteration 0, loss = 0.4847
Checking accuracy on validation set
Got 668 / 1000 correct (66.80)

Epoch 4, Iteration 100, loss = 0.3688
Checking accuracy on validation set
Got 661 / 1000 correct (66.10)

Epoch 4, Iteration 200, loss = 0.4061
Checking accuracy on validation set
Got 649 / 1000 correct (64.90)

Epoch 4, Iteration 300, loss = 0.5048
Checking accuracy on validation set
Got 649 / 1000 correct (64.90)

Epoch 4, Iteration 400, loss = 0.5549
Checking accuracy on validation set
Got 626 / 1000 correct (62.60)

Epoch 4, Iteration 500, loss = 0.5396
Checking accuracy on validation set
Got 657 / 1000 correct (65.70)

Epoch 4, Iteration 600, loss = 0.6802
Checking accuracy on validation set
Got 659 / 1000 correct (65.90)

Epoch 4, Iteration 700, loss = 0.7105
Checking accuracy on validation set
Got 649 / 1000 correct (64.90)

Epoch 5, Iteration 0, loss = 0.3327
Checking accuracy on validation set
Got 641 / 1000 correct (64.10)

Epoch 5, Iteration 100, loss = 0.2930
Checking accuracy on validation set
Got 654 / 1000 correct (65.40)

Epoch 5, Iteration 200, loss = 0.3265
Checking accuracy on validation set
Got 640 / 1000 correct (64.00)

Epoch 5, Iteration 300, loss = 0.4680
Checking accuracy on validation set
Got 646 / 1000 correct (64.60)

Epoch 5, Iteration 400, loss = 0.4523
Checking accuracy on validation set
Got 630 / 1000 correct (63.00)

Epoch 5, Iteration 500, loss = 0.3504
Checking accuracy on validation set
Got 636 / 1000 correct (63.60)
```

```
Epoch 5, Iteration 600, loss = 0.4826
Checking accuracy on validation set
Got 624 / 1000 correct (62.40)

Epoch 5, Iteration 700, loss = 0.3877
Checking accuracy on validation set
Got 633 / 1000 correct (63.30)

Epoch 6, Iteration 0, loss = 0.3340
Checking accuracy on validation set
Got 640 / 1000 correct (64.00)

Epoch 6, Iteration 100, loss = 0.3008
Checking accuracy on validation set
Got 622 / 1000 correct (62.20)

Epoch 6, Iteration 200, loss = 0.2267
Checking accuracy on validation set
Got 639 / 1000 correct (63.90)

Epoch 6, Iteration 300, loss = 0.2209
Checking accuracy on validation set
Got 637 / 1000 correct (63.70)

Epoch 6, Iteration 400, loss = 0.2637
Checking accuracy on validation set
Got 608 / 1000 correct (60.80)

Epoch 6, Iteration 500, loss = 0.4079
Checking accuracy on validation set
Got 634 / 1000 correct (63.40)

Epoch 6, Iteration 600, loss = 0.2750
Checking accuracy on validation set
Got 637 / 1000 correct (63.70)

Epoch 6, Iteration 700, loss = 0.3442
Checking accuracy on validation set
Got 620 / 1000 correct (62.00)

Epoch 7, Iteration 0, loss = 0.2962
Checking accuracy on validation set
Got 607 / 1000 correct (60.70)

Epoch 7, Iteration 100, loss = 0.2368
Checking accuracy on validation set
Got 629 / 1000 correct (62.90)

Epoch 7, Iteration 200, loss = 0.1804
Checking accuracy on validation set
Got 624 / 1000 correct (62.40)

Epoch 7, Iteration 300, loss = 0.2619
Checking accuracy on validation set
Got 637 / 1000 correct (63.70)

Epoch 7, Iteration 400, loss = 0.3467
Checking accuracy on validation set
Got 634 / 1000 correct (63.40)

Epoch 7, Iteration 500, loss = 0.2569
Checking accuracy on validation set
Got 643 / 1000 correct (64.30)

Epoch 7, Iteration 600, loss = 0.4019
```

```
Checking accuracy on validation set
Got 636 / 1000 correct (63.60)

Epoch 7, Iteration 700, loss = 0.2949
Checking accuracy on validation set
Got 626 / 1000 correct (62.60)

Epoch 8, Iteration 0, loss = 0.2418
Checking accuracy on validation set
Got 631 / 1000 correct (63.10)

Epoch 8, Iteration 100, loss = 0.1215
Checking accuracy on validation set
Got 618 / 1000 correct (61.80)

Epoch 8, Iteration 200, loss = 0.3627
Checking accuracy on validation set
Got 632 / 1000 correct (63.20)

Epoch 8, Iteration 300, loss = 0.1761
Checking accuracy on validation set
Got 622 / 1000 correct (62.20)

Epoch 8, Iteration 400, loss = 0.0852
Checking accuracy on validation set
Got 628 / 1000 correct (62.80)

Epoch 8, Iteration 500, loss = 0.0666
Checking accuracy on validation set
Got 613 / 1000 correct (61.30)

Epoch 8, Iteration 600, loss = 0.1444
Checking accuracy on validation set
Got 617 / 1000 correct (61.70)

Epoch 8, Iteration 700, loss = 0.2341
Checking accuracy on validation set
Got 598 / 1000 correct (59.80)

Epoch 9, Iteration 0, loss = 0.1425
Checking accuracy on validation set
Got 607 / 1000 correct (60.70)

Epoch 9, Iteration 100, loss = 0.0908
Checking accuracy on validation set
Got 607 / 1000 correct (60.70)

Epoch 9, Iteration 200, loss = 0.1056
Checking accuracy on validation set
Got 633 / 1000 correct (63.30)

Epoch 9, Iteration 300, loss = 0.1142
Checking accuracy on validation set
Got 617 / 1000 correct (61.70)

Epoch 9, Iteration 400, loss = 0.2569
Checking accuracy on validation set
Got 625 / 1000 correct (62.50)

Epoch 9, Iteration 500, loss = 0.2346
Checking accuracy on validation set
Got 613 / 1000 correct (61.30)

Epoch 9, Iteration 600, loss = 0.2183
Checking accuracy on validation set
```

```
Got 620 / 1000 correct (62.00)

Epoch 9, Iteration 700, loss = 0.1708
Checking accuracy on validation set
Got 625 / 1000 correct (62.50)

Maximum accuracy attained:  67.4
```

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.

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

```
Checking accuracy on test set
Got 6146 / 10000 correct (61.46)
```

```
61.46
```

Inline Question 2: Provide one example each of usage of self-attention and attention in computer vision. Explain the difference between the two. (10% of the grade)

Your Answer:

The difference is that, attention can look at other layers, while self attention looks at the same layer.

Self attention: e.g., Non-local neural network. The self attention module is inserted in the plain network, by taking in the previous feature map. It computes the response at a position in a sequence (e.g., a sentence) by attending to all positions and taking their weighted average in an embedding space.

Attention: Learn to pay attention.  Learn to pay attention network For attention layer 1, 2 and 3, each take two input, which are different layers. So, it can facilitate the learning of diverse and complementary attention-weighted features.

Double Attention Blocks: After conv layers 1 and 2 (10% of the grade)

```
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->Linea
```

```
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 = 2.3058
Checking accuracy on validation set
Got 108 / 1000 correct (10.80)

Epoch 0, Iteration 100, loss = 1.7420
Checking accuracy on validation set
Got 423 / 1000 correct (42.30)

Epoch 0, Iteration 200, loss = 1.6482
Checking accuracy on validation set
Got 490 / 1000 correct (49.00)

Epoch 0, Iteration 300, loss = 1.3416
Checking accuracy on validation set
Got 507 / 1000 correct (50.70)

Epoch 0, Iteration 400, loss = 1.5649
Checking accuracy on validation set
Got 524 / 1000 correct (52.40)

Epoch 0, Iteration 500, loss = 1.1091
Checking accuracy on validation set
Got 568 / 1000 correct (56.80)

Epoch 0, Iteration 600, loss = 1.2562
Checking accuracy on validation set
Got 568 / 1000 correct (56.80)

Epoch 0, Iteration 700, loss = 1.2298
Checking accuracy on validation set
Got 583 / 1000 correct (58.30)

Epoch 1, Iteration 0, loss = 1.3650
Checking accuracy on validation set
Got 594 / 1000 correct (59.40)

Epoch 1, Iteration 100, loss = 1.0319
Checking accuracy on validation set
Got 582 / 1000 correct (58.20)

Epoch 1, Iteration 200, loss = 1.0088
Checking accuracy on validation set
Got 604 / 1000 correct (60.40)

Epoch 1, Iteration 300, loss = 0.8968
Checking accuracy on validation set
Got 603 / 1000 correct (60.30)
```

```
Epoch 1, Iteration 400, loss = 0.9863
Checking accuracy on validation set
Got 638 / 1000 correct (63.80)

Epoch 1, Iteration 500, loss = 0.8812
Checking accuracy on validation set
Got 639 / 1000 correct (63.90)

Epoch 1, Iteration 600, loss = 0.7624
Checking accuracy on validation set
Got 622 / 1000 correct (62.20)

Epoch 1, Iteration 700, loss = 0.9288
Checking accuracy on validation set
Got 646 / 1000 correct (64.60)

Epoch 2, Iteration 0, loss = 0.9072
Checking accuracy on validation set
Got 642 / 1000 correct (64.20)

Epoch 2, Iteration 100, loss = 1.1104
Checking accuracy on validation set
Got 628 / 1000 correct (62.80)

Epoch 2, Iteration 200, loss = 0.8449
Checking accuracy on validation set
Got 634 / 1000 correct (63.40)

Epoch 2, Iteration 300, loss = 0.9172
Checking accuracy on validation set
Got 651 / 1000 correct (65.10)

Epoch 2, Iteration 400, loss = 0.7421
Checking accuracy on validation set
Got 628 / 1000 correct (62.80)

Epoch 2, Iteration 500, loss = 0.8396
Checking accuracy on validation set
Got 657 / 1000 correct (65.70)

Epoch 2, Iteration 600, loss = 0.8567
Checking accuracy on validation set
Got 638 / 1000 correct (63.80)

Epoch 2, Iteration 700, loss = 0.6974
Checking accuracy on validation set
Got 638 / 1000 correct (63.80)

Epoch 3, Iteration 0, loss = 0.6501
Checking accuracy on validation set
Got 655 / 1000 correct (65.50)

Epoch 3, Iteration 100, loss = 0.6193
Checking accuracy on validation set
Got 650 / 1000 correct (65.00)

Epoch 3, Iteration 200, loss = 0.5977
Checking accuracy on validation set
Got 618 / 1000 correct (61.80)

Epoch 3, Iteration 300, loss = 0.7247
Checking accuracy on validation set
Got 652 / 1000 correct (65.20)
```

```
Epoch 3, Iteration 400, loss = 0.8300
Checking accuracy on validation set
Got 649 / 1000 correct (64.90)

Epoch 3, Iteration 500, loss = 0.5797
Checking accuracy on validation set
Got 661 / 1000 correct (66.10)

Epoch 3, Iteration 600, loss = 0.7892
Checking accuracy on validation set
Got 652 / 1000 correct (65.20)

Epoch 3, Iteration 700, loss = 0.6587
Checking accuracy on validation set
Got 676 / 1000 correct (67.60)

Epoch 4, Iteration 0, loss = 0.5098
Checking accuracy on validation set
Got 661 / 1000 correct (66.10)

Epoch 4, Iteration 100, loss = 0.5056
Checking accuracy on validation set
Got 646 / 1000 correct (64.60)

Epoch 4, Iteration 200, loss = 0.5622
Checking accuracy on validation set
Got 643 / 1000 correct (64.30)

Epoch 4, Iteration 300, loss = 0.6241
Checking accuracy on validation set
Got 638 / 1000 correct (63.80)

Epoch 4, Iteration 400, loss = 0.6607
Checking accuracy on validation set
Got 652 / 1000 correct (65.20)

Epoch 4, Iteration 500, loss = 0.7982
Checking accuracy on validation set
Got 653 / 1000 correct (65.30)

Epoch 4, Iteration 600, loss = 0.6257
Checking accuracy on validation set
Got 653 / 1000 correct (65.30)

Epoch 4, Iteration 700, loss = 0.5088
Checking accuracy on validation set
Got 657 / 1000 correct (65.70)

Epoch 5, Iteration 0, loss = 0.5851
Checking accuracy on validation set
Got 651 / 1000 correct (65.10)

Epoch 5, Iteration 100, loss = 0.4000
Checking accuracy on validation set
Got 665 / 1000 correct (66.50)

Epoch 5, Iteration 200, loss = 0.3223
Checking accuracy on validation set
Got 648 / 1000 correct (64.80)

Epoch 5, Iteration 300, loss = 0.3842
Checking accuracy on validation set
Got 658 / 1000 correct (65.80)

Epoch 5, Iteration 400, loss = 0.5419
```

```
Checking accuracy on validation set
Got 662 / 1000 correct (66.20)

Epoch 5, Iteration 500, loss = 0.4190
Checking accuracy on validation set
Got 671 / 1000 correct (67.10)

Epoch 5, Iteration 600, loss = 0.4086
Checking accuracy on validation set
Got 654 / 1000 correct (65.40)

Epoch 5, Iteration 700, loss = 0.7103
Checking accuracy on validation set
Got 659 / 1000 correct (65.90)

Epoch 6, Iteration 0, loss = 0.3056
Checking accuracy on validation set
Got 653 / 1000 correct (65.30)

Epoch 6, Iteration 100, loss = 0.3308
Checking accuracy on validation set
Got 650 / 1000 correct (65.00)

Epoch 6, Iteration 200, loss = 0.4443
Checking accuracy on validation set
Got 650 / 1000 correct (65.00)

Epoch 6, Iteration 300, loss = 0.4316
Checking accuracy on validation set
Got 637 / 1000 correct (63.70)

Epoch 6, Iteration 400, loss = 0.3421
Checking accuracy on validation set
Got 635 / 1000 correct (63.50)

Epoch 6, Iteration 500, loss = 0.3182
Checking accuracy on validation set
Got 641 / 1000 correct (64.10)

Epoch 6, Iteration 600, loss = 0.4028
Checking accuracy on validation set
Got 655 / 1000 correct (65.50)

Epoch 6, Iteration 700, loss = 0.3951
Checking accuracy on validation set
Got 640 / 1000 correct (64.00)

Epoch 7, Iteration 0, loss = 0.2627
Checking accuracy on validation set
Got 647 / 1000 correct (64.70)

Epoch 7, Iteration 100, loss = 0.1297
Checking accuracy on validation set
Got 639 / 1000 correct (63.90)

Epoch 7, Iteration 200, loss = 0.2590
Checking accuracy on validation set
Got 641 / 1000 correct (64.10)

Epoch 7, Iteration 300, loss = 0.2817
Checking accuracy on validation set
Got 647 / 1000 correct (64.70)

Epoch 7, Iteration 400, loss = 0.2291
Checking accuracy on validation set
```

```
Got 654 / 1000 correct (65.40)

Epoch 7, Iteration 500, loss = 0.2627
Checking accuracy on validation set
Got 628 / 1000 correct (62.80)

Epoch 7, Iteration 600, loss = 0.2377
Checking accuracy on validation set
Got 643 / 1000 correct (64.30)

Epoch 7, Iteration 700, loss = 0.2287
Checking accuracy on validation set
Got 643 / 1000 correct (64.30)

Epoch 8, Iteration 0, loss = 0.1444
Checking accuracy on validation set
Got 644 / 1000 correct (64.40)

Epoch 8, Iteration 100, loss = 0.2840
Checking accuracy on validation set
Got 647 / 1000 correct (64.70)

Epoch 8, Iteration 200, loss = 0.1046
Checking accuracy on validation set
Got 643 / 1000 correct (64.30)

Epoch 8, Iteration 300, loss = 0.1472
Checking accuracy on validation set
Got 636 / 1000 correct (63.60)

Epoch 8, Iteration 400, loss = 0.2890
Checking accuracy on validation set
Got 653 / 1000 correct (65.30)

Epoch 8, Iteration 500, loss = 0.2172
Checking accuracy on validation set
Got 618 / 1000 correct (61.80)

Epoch 8, Iteration 600, loss = 0.2859
Checking accuracy on validation set
Got 632 / 1000 correct (63.20)

Epoch 8, Iteration 700, loss = 0.3395
Checking accuracy on validation set
Got 648 / 1000 correct (64.80)

Epoch 9, Iteration 0, loss = 0.1015
Checking accuracy on validation set
Got 653 / 1000 correct (65.30)

Epoch 9, Iteration 100, loss = 0.1523
Checking accuracy on validation set
Got 641 / 1000 correct (64.10)

Epoch 9, Iteration 200, loss = 0.1137
Checking accuracy on validation set
Got 660 / 1000 correct (66.00)

Epoch 9, Iteration 300, loss = 0.0811
Checking accuracy on validation set
Got 660 / 1000 correct (66.00)

Epoch 9, Iteration 400, loss = 0.1525
Checking accuracy on validation set
Got 652 / 1000 correct (65.20)
```

```
Epoch 9, Iteration 500, loss = 0.1637
Checking accuracy on validation set
Got 647 / 1000 correct (64.70)

Epoch 9, Iteration 600, loss = 0.1361
Checking accuracy on validation set
Got 641 / 1000 correct (64.10)

Epoch 9, Iteration 700, loss = 0.1870
Checking accuracy on validation set
Got 646 / 1000 correct (64.60)

Maximum accuracy attained:  67.60000000000001
```

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.

```
127 doubleModel = model
    check_accuracy(loader_test, doubleModel)
```

```
Checking accuracy on test set
Got 6242 / 10000 correct (62.42)
```

```
127 62.419999999999995
```

Inline Question 3: Rank the above models based on their performance on test dataset (20% of the grade)

(You are encouraged to run each of the experiments (training) atleast 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. Vanilla CNN - 59.77, 60.81, 57.70, 59.77 Average: 59.51
2. early attention - 59.39, 61.11, 60.81, 60.79 Average: 60.525
3. late attention - 58.11, 61.62, 60.21, 61.46 Average: 60.24
4. double attention - 67.7, 62.88, 61.76, 62.41 Average: 64.68

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

Your Answer: Double attention > early attention > late attention > vanilla

Generally speaking, more attention leads to better result. The paper explains that non-local blocks(attention blocks) can perform long-range multi-hop communication. Messages can be delivered back and forth between distant positions in spacetime, which is hard to do via local models.

Another explanation is that attention block make the network deeper. Usually deeper network and network with more parameters have better results, because they are more powerful.

The early attention is generally better than the late attention, mainly because the resolution for the feature map at early attention is higher.