Problem 5

In [ ]:
```
import torch.nn as nn
import torch
import torchvision
```

In [ ]:
```
class Net1(nn.Module):
    def __init__(self, num_classes=10):
        super(Net1, self).__init__()
        self.features = nn.Sequential(
            nn.Conv2d(3, 64, kernel_size=7, stride=1),
            nn.ReLU(),
            nn.Conv2d(64, 192, kernel_size=3, stride=1),
            nn.ReLU(),
            nn.Conv2d(192, 384, kernel_size=3, stride=1),
            nn.ReLU(),
            nn.Conv2d(384, 256, kernel_size=3, stride=1),
            nn.ReLU(),
            nn.Conv2d(256, 256, kernel_size=3, stride=1),
        )
        self.classifier = nn.Sequential(
            nn.Linear(256 * 18 * 18, 4096),
            nn.ReLU(),
            nn.Linear(4096, 4096),
            nn.ReLU(),
            nn.Linear(4096, num_classes)
        )

    def forward(self, x):
        x = self.features(x)
        x = torch.flatten(x, 1)
        x = self.classifier(x)
        return x
```

When we set kernel size of convolutional neural network equal to input image size,

- each out cell is of size 1×1, and
- each out cell is return of activation followed by linear combination of all input entries and bias.

Thus we set first Conv2d as (in_channel, out_channel, kernel_size) = (256,4096,18). Next, we use the property that 1×1 convolutional is equivalent to linear.

In [ ]:
```
class Net2(nn.Module):
    def __init__(self, num_classes=10):
        super(Net2, self).__init__()
        self.features = nn.Sequential(
            nn.Conv2d(3, 64, kernel_size=7, stride=1),
            nn.ReLU(),
            nn.Conv2d(64, 192, kernel_size=3, stride=1),
            nn.ReLU(),
            nn.Conv2d(192, 384, kernel_size=3, stride=1),
            nn.ReLU(),
            nn.Conv2d(384, 256, kernel_size=3, stride=1),
            nn.ReLU(),
            nn.Conv2d(256, 256, kernel_size=3, stride=1),
        )
```

```
    #####################################################
    ### TODO: Complete initialization of self.classifier    ###
    ###        by filling in the ...                        ###
    #####################################################
        self.classifier = nn.Sequential(
            nn.Conv2d(256, 4096, kernel_size=18),
            nn.ReLU(),
            nn.Conv2d(4096, 4096, kernel_size=1),
            nn.ReLU(),
            nn.Conv2d(4096, num_classes, kernel_size=1)
      )

    def copy_weights_from(self, net1 : Net1):
        with torch.no_grad():
            for i in range(0, len(self.features), 2):
                self.features[i].weight.copy_(net1.features[i].weight)
                self.features[i].bias.copy_(net1.features[i].bias)

            for i in range(len(self.classifier)):
                ################################################
                ### TO DO: Correctly transfer weight of Net1     ###
                ################################################
                if i == 0:
                    self.classifier[i].weight.data = torch.reshape(net1.classifier[
                    self.classifier[i].bias.data = net1.classifier[i].bias.data
                elif i == 2:
                    self.classifier[i].weight.data = torch.reshape(net1.classifier[
                    self.classifier[i].bias.data = net1.classifier[i].bias.data
                elif i == 4:
                    self.classifier[i].weight.data = torch.reshape(net1.classifier[
                    self.classifier[i].bias.data = net1.classifier[i].bias.data



    def forward(self, x):
        x = self.features(x)
        x = self.classifier(x)
        return x
```

In [ ]:

```
model1 = Net1() # model1 randomly initialized
model2 = Net2()
model2.copy_weights_from(model1)

test_dataset = torchvision.datasets.CIFAR10(
    root='./data',
    train=False,
    transform=torchvision.transforms.ToTensor(),
    download=True
)

test_loader = torch.utils.data.DataLoader(
    dataset=test_dataset,
    batch_size=10
)

imgs, _ = next(iter(test_loader))
diff = torch.mean((model1(imgs) - model2(imgs).squeeze()) ** 2)
print(f"Average Pixel Difference: {diff.item()}") # should be small


test_dataset = torchvision.datasets.CIFAR10(
    root='./cifar_10data',
```

```
    train=False,
    transform=torchvision.transforms.Compose([
        torchvision.transforms.Resize((36, 38)),
        torchvision.transforms.ToTensor()
        ]),
    download=True
)

test_loader = torch.utils.data.DataLoader(
    dataset=test_dataset,
    batch_size=10,
    shuffle=False
)
```

```
Downloading https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz to ./data₩cifar-10-python.tar.gz
  0%|          | 0/170498071 [00:00<?, ?it/s]
Extracting ./data₩cifar-10-python.tar.gz to ./data
Average Pixel Difference: 6.363434939665928e-17
Files already downloaded and verified
```

The difference is enoughly small, as we wished.

In [ ]:

```
images, _ = next(iter(test_loader))
b, w, h = images.shape[0], images.shape[-1], images.shape[-2]
out1 = torch.empty((b, 10, h - 31, w - 31))
for i in range(h - 31):
    for j in range(w - 31):
        ########################################################
        ### TO DO: fill in ... to make out1 and out2 equal   ###
        ########################################################

        out1[:, :, i, j] = model1(images[:,:,i:i+32,j:j+32])
out2 = model2(images)
diff = torch.mean((out1 - out2) ** 2)

print(f"Average Pixel Diff: {diff.item()}")
```

```
Average Pixel Diff: 6.630028633646868e-17
```

This difference is also small, and similar to previous one. This guarantees our expectation realizes well.