In [1]:
```
import torch
import torch.nn as nn
import torch.nn.functional as F
import torchvision
import torchvision.transforms as transforms
from torch.utils.data import SubsetRandomSampler
import torch.optim as optim

import numpy as np
import matplotlib.pyplot as plt
```

In [2]:
```
def get_train_valid_loader(batch_size,
                           random_seed,
                           valid_size=0.1,
                           shuffle=True,
                           show_sample=False,
                           num_workers=4,
                           pin_memory=False):

    error_msg = "[!] valid_size should be in the range [0, 1]."
    assert ((valid_size >= 0) and (valid_size <= 1)), error_msg


    train_transform = transforms.Compose(
        [transforms.ToTensor(),
         transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])
    valid_transform = transforms.Compose(
        [transforms.ToTensor(),
         transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])

    trainset = torchvision.datasets.CIFAR10(root='./cifardata', train=
True,
                                            download=True, transform=train
_transform)
    validset = torchvision.datasets.CIFAR10(root='./cifardata', train=
True,
                                            download=True, transform=valid
_transform)

    num_train = len(trainset)
    indices = list(range(num_train))
    split = int(np.floor(valid_size * num_train))

    if shuffle:
        np.random.seed(random_seed)
        np.random.shuffle(indices)
```

```
    train_idx, valid_idx = indices[split:], indices[:split]
    #print("from get loaders: ", len(train_idx))
    train_sampler = SubsetRandomSampler(train_idx)
    valid_sampler = SubsetRandomSampler(valid_idx)

    train_loader = torch.utils.data.DataLoader(
        trainset, batch_size=batch_size, sampler=train_sampler,
        num_workers=num_workers, pin_memory=pin_memory,
    )
    valid_loader = torch.utils.data.DataLoader(
        validset, batch_size=batch_size, sampler=valid_sampler,
        num_workers=num_workers, pin_memory=pin_memory,
    )

    return (train_loader, valid_loader)
```

In [3]:

```
def save_predictions(filename, y):
    """
    Dumps y into .npy file
    """
    np.save(filename, y)
```

In [4]:

```
transform = transforms.Compose(
[transforms.ToTensor(),
     transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])

trainset = torchvision.datasets.CIFAR10(root='./data', train=True,
                                        download=True, transform=trans
form)
# trainloader = torch.utils.data.DataLoader(trainset, batch_size=50000
,
#                                            shuffle=True, num_workers=
2)

testset = torchvision.datasets.CIFAR10(root='./data', train=False,down
load=True, transform=transform)
testloader = torch.utils.data.DataLoader(testset, batch_size=4,
                                         shuffle=False, num_workers=2)

classes = ('plane', 'car', 'bird', 'cat','deer', 'dog', 'frog', 'horse
', 'ship', 'truck')


class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        self.conv1 = nn.Conv2d(3, 18, 3, padding=1)
        self.pool1 = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(18, 24, 3, padding=1)
```

```
        self.pool2 = nn.MaxPool2d(2, 2)
        self.conv3 = nn.Conv2d(24, 32, 3, padding=1)
        self.pool3 = nn.MaxPool2d(2, 2)



#         self.conv3 = nn.Conv2d(128, 256, 3, stride=1, padding=1)
#         self.pool3 = nn.MaxPool2d(2, 2)

#         self.conv4 = nn.Conv2d(256, 512, 3, stride=1, padding=1)
#         self.pool4 = nn.MaxPool2d(2, 2)

        self.fc1 = nn.Linear(32*4*4, 64)
        self.fc2 = nn.Linear(64, 10)
#         self.fc3 = nn.Linear(256, 512)
#         self.fc4 = nn.Linear(512, 1024)
#         self.fc5 = nn.Linear(1024, 10)

    def forward(self, x):
        x = self.pool1(F.relu(self.conv1(x)))
        #print("X size after conv1: ", x.size())
        x = self.pool2(F.relu(self.conv2(x)))
        x = self.pool3(F.relu(self.conv3(x)))
        #print("X size after conv2: ", x.size())
#         x = self.pool3(F.relu(self.conv3(x)))
#         x = self.pool4(F.relu(self.conv4(x)))
        x = x.view(-1, 32*4*4)
        #print("X size after flatten: ", x.size())
        x = F.relu(self.fc1(x))
        #print("X size after fc1: ", x.size())
#         x = F.relu(self.fc2(x))
#         x = F.relu(self.fc3(x))
#         x = F.relu(self.fc4(x))
        #print("X size after fc2: ", x.size())
        x = self.fc2(x)
        #print("X size after fc3: ", x.size())
        return x



net = CNN()


criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(net.parameters(), lr=0.001)

## Getting 2 different loaders for train set and validation sets
train_loader, valid_loader = get_train_valid_loader(4,1,0.1,False,Fals
e,4,False)
```

```
#trainloader2 = torch.utils.data.DataLoader(X_train, batch_size=4,shuf
fle=True, num_workers=2)

for epoch in range(10):  # loop over the dataset multiple times

    running_loss = 0.0
    # Train on batches, Test on train / validation sets

    for i, data in enumerate(train_loader, 0):
        # get the inputs
        inputs, labels = data

        # zero the parameter gradients
        optimizer.zero_grad()

        # forward + backward + optimize
        outputs = net(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        # print statistics
        running_loss += loss.item()
        if i % 2000 == 1999:    # print every 2000 mini-batches
            print('[%d, %5d] loss: %.3f' % (epoch + 1, i + 1, running_
loss / 2000))
            running_loss = 0.0

print('Finished Training')


#outputs = net(images_test)

#max_val, predicted = torch.max(outputs, 1)
##################### VALIDATION SET #################################

correct_valid = 0
total_valid = 0
with torch.no_grad():
    for data in valid_loader:
        images, labels = data
        outputs = net(images)
        max_val, predicted = torch.max(outputs.data, 1)
        total_valid += labels.size(0)
        correct_valid += (predicted == labels).sum().item()

print('Validation Accuracy of the network on the 5000 validation image
s: %d %%' % (
    100 * correct_valid / total_valid))
print("Total Validation Number: ", total_valid)
```

```
print("Correct Validation Number: ", correct_valid)

##################### TEST SET #################################

dataiter_testset = iter(testloader)
images_testset, labels_testset = dataiter_testset.next()

# outputs_testset = net(images_testset)
# max_val, predicted = torch.max(outputs_testset, 1)

correct_test = 0
total_test = 0
final_test_output_list = []
with torch.no_grad():
    for data in testloader:
        images, labels = data
        outputs = net(images)
        max_val, predicted = torch.max(outputs.data, 1)
        total_test += labels.size(0)
        correct_test += (predicted == labels).sum().item()
        for item in predicted:
            final_test_output_list.append(item.item())
    print('Validation Accuracy of the network on the 10000 Test images
: %d %%' % (100 * correct_test / total_test))
    print("Total Test Number: ", total_test)
    print("Correct Test Number: ", correct_test)


##################### SAVE PREDICTIONS ################################
##
# Save Predictions as numpy array
final_test_output_np = np.asarray(final_test_output_list)
filename = "ans2-v6-ung200-avs431"
save_predictions(filename, final_test_output_np)
```

```
Files already downloaded and verified
Files already downloaded and verified
Files already downloaded and verified
Files already downloaded and verified
[1,  2000] loss: 1.821
[1,  4000] loss: 1.495
[1,  6000] loss: 1.374
[1,  8000] loss: 1.289
[1, 10000] loss: 1.255
[2,  2000] loss: 1.109
[2,  4000] loss: 1.105
[2,  6000] loss: 1.049
[2,  8000] loss: 1.041
[2, 10000] loss: 1.039
[3,  2000] loss: 0.955
```

```
[3,  4000] loss: 0.959
[3,  6000] loss: 0.963
[3,  8000] loss: 0.943
[3, 10000] loss: 0.944
[4,  2000] loss: 0.881
[4,  4000] loss: 0.866
[4,  6000] loss: 0.868
[4,  8000] loss: 0.888
[4, 10000] loss: 0.897
[5,  2000] loss: 0.813
[5,  4000] loss: 0.823
[5,  6000] loss: 0.855
[5,  8000] loss: 0.863
[5, 10000] loss: 0.860
[6,  2000] loss: 0.799
[6,  4000] loss: 0.807
[6,  6000] loss: 0.791
[6,  8000] loss: 0.811
[6, 10000] loss: 0.822
[7,  2000] loss: 0.757
[7,  4000] loss: 0.775
[7,  6000] loss: 0.755
[7,  8000] loss: 0.777
[7, 10000] loss: 0.800
[8,  2000] loss: 0.711
[8,  4000] loss: 0.750
[8,  6000] loss: 0.744
[8,  8000] loss: 0.754
[8, 10000] loss: 0.777
[9,  2000] loss: 0.692
[9,  4000] loss: 0.710
[9,  6000] loss: 0.731
[9,  8000] loss: 0.756
[9, 10000] loss: 0.765
[10,  2000] loss: 0.673
[10,  4000] loss: 0.703
[10,  6000] loss: 0.709
[10,  8000] loss: 0.729
[10, 10000] loss: 0.730
Finished Training
Validation Accuracy of the network on the 5000 validation images: 69
%
Total Validation Number:  5000
Correct Validation Number:  3492
Validation Accuracy of the network on the 10000 Test images: 69 %
Total Test Number:  10000
Correct Test Number:  6924
```

Parameter Tuning:

1. We also tried SGD for the same setting which gave us an accuracy of 58%
2. We tried using RMS prop as our optimizer with Learning Rate 0.001 but for 10 epochs but it gave us an accuracy of 60%
3. In Adam's Optimizer, we first tried increasing the weight decay to 0.05 but our cost was constant and was not decreasing even after 5 epochs so we stopped it. We also tried increasing the learning rate to 0.05 but even that didn't help.
4. We increased the number of channels and kernel sizes but got an accuracy of 61%
5. We tried increasing the number of neurons after the convulation is being done, and reached 63%
6. After which we tried combining 4 and 5, so we increased the number of neurons, added a few convolutional layers and calculated and changed the kernel sizes and number of channels to get the accuracy of 69%

DATASET SIZES:

TRAIN: 45000

VALIDATION: 5000

TEST: 10000