CNN

January 10, 2022

[1]:
```
import torch
import torch.nn as nn
import torch.nn.functional as F
import torchvision
import torchvision.transforms as transforms
import matplotlib.pyplot as plt
import numpy as np
```

[2]:
```
# Device config
# device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
```

[3]:
```
# Hyper-parameters
num_epochs = 4
batch_size = 4
learning_rate = 0.001
```

[4]:
```
# Transform PILI images to Tensors of normalized range
transform = transforms.Compose([transforms.ToTensor(), transforms.Normalize((0.
 ↪5, 0.5, 0.5), (0.5, 0.5, 0.5))])
```

[5]:
```
# CIFAR 10 dataset
train_dataset = torchvision.datasets.CIFAR10(root='./CIFAR10', train=True,␣
 ↪transform=transform, download=True)
test_dataset = torchvision.datasets.CIFAR10(root='./CIFAR10', train=False,␣
 ↪transform=transform, download=True)
```

```
Files already downloaded and verified
Files already downloaded and verified
```

[6]:
```
# Data Loader
train_loader = torch.utils.data.DataLoader(dataset=train_dataset,␣
 ↪batch_size=batch_size, shuffle = True)
test_loader = torch.utils.data.DataLoader(dataset=test_dataset,␣
 ↪batch_size=batch_size, shuffle = False)
```

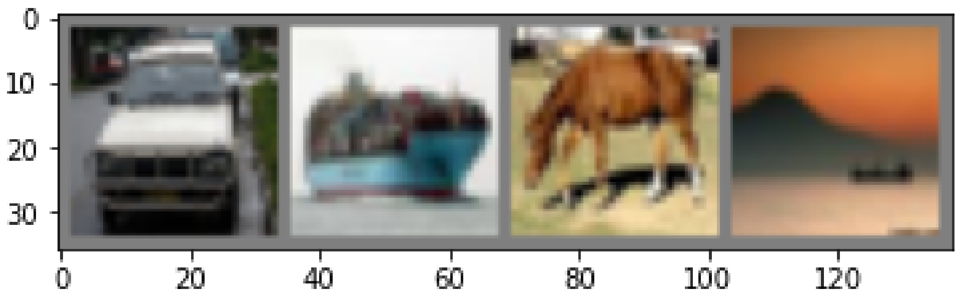
[7]:
```
classes = ('plane', 'car', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse',␣
 ↪'ship', 'truck')
```

```
[8]: #visualize data

def imshow(img):
    img = img / 2 + 0.5  # unnormalize
    npimg = img.numpy()
    plt.imshow(np.transpose(npimg, (1, 2, 0)))
    plt.show()


# get some random training images
dataiter = iter(train_loader)
images, labels = dataiter.next()

# show images
imshow(torchvision.utils.make_grid(images))
```



```
[9]: # implement CNN
class ConvNet(nn.Module):
    def __init__(self):
        super(ConvNet, self).__init__()
        self.conv1 = nn.Conv2d(3, 6, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(16*5*5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(-1, 16*5*5)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

```
model = ConvNet()#.to(device)
```

[10]:
```
# loss and optimizer
criterion = nn.CrossEntropyLoss() #softmax included
optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate)
```

[11]:
```
# Training loop
n_total_steps = len(train_loader)
for epoch in range(num_epochs):
    for i,(images, labels) in enumerate(train_loader):
        # origin shape: [4, 3, 32, 32] = 4, 3, 1024
        # input_layer: 3 input channels, 6 output channels, 5 kernel size
        images = images#.to(device)
        labels = labels#.to(device)

        # Forward pass
        outputs = model(images)
        loss = criterion(outputs, labels)

        # Backward and optimize
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        if (i+1) % 2000 == 0:
            print(f'Epoch [{epoch+1}/{num_epochs}], Step [{i+1}/
 ↪{n_total_steps}], Loss: {loss.item():.4f}')

print('Finished Training')
```

```
Epoch [1/4], Step [2000/12500], Loss: 2.2831
Epoch [1/4], Step [4000/12500], Loss: 2.2722
Epoch [1/4], Step [6000/12500], Loss: 2.2861
Epoch [1/4], Step [8000/12500], Loss: 2.1327
Epoch [1/4], Step [10000/12500], Loss: 2.1249
Epoch [1/4], Step [12000/12500], Loss: 1.8305
Epoch [2/4], Step [2000/12500], Loss: 1.5158
Epoch [2/4], Step [4000/12500], Loss: 1.6142
Epoch [2/4], Step [6000/12500], Loss: 1.7015
Epoch [2/4], Step [8000/12500], Loss: 0.9900
Epoch [2/4], Step [10000/12500], Loss: 2.1570
Epoch [2/4], Step [12000/12500], Loss: 1.9173
Epoch [3/4], Step [2000/12500], Loss: 1.3915
Epoch [3/4], Step [4000/12500], Loss: 1.1648
Epoch [3/4], Step [6000/12500], Loss: 0.9419
Epoch [3/4], Step [8000/12500], Loss: 0.9808
Epoch [3/4], Step [10000/12500], Loss: 2.2573
Epoch [3/4], Step [12000/12500], Loss: 1.4934
Epoch [4/4], Step [2000/12500], Loss: 1.9043
Epoch [4/4], Step [4000/12500], Loss: 1.1061
Epoch [4/4], Step [6000/12500], Loss: 1.2460
Epoch [4/4], Step [8000/12500], Loss: 1.7014
Epoch [4/4], Step [10000/12500], Loss: 2.2474
Epoch [4/4], Step [12000/12500], Loss: 1.7605
Finished Training
```

[13]:
```
# Testing loop for evaluation
with torch.no_grad():
    n_correct = 0
    n_samples = 0
    n_class_correct = [0 for i in range(10)]
    n_class_samples = [0 for i in range(10)]
    for images, labels in test_loader:
        images = images #.to(device)
        labels = labels #.to(device)
        outputs = model(images)

        # return value and index
        _, predicted = torch.max(outputs, 1)
        n_samples += labels.shape[0]
        n_correct += (predictions == labels).sum().item()

        for i in range(batch_size):
            label = labels[i]
            pred = predicted[i]
            if (label == pred):
                n_class_correct[label] += 1
            n_class_samples[label] += 1


    acc = 100.0 * n_correct / n_samples
    print(f'Accuracy of the CNN = {acc} %')

    for i in range (10):
        acc = 100.00 * n_class_correct[i] / n_class_samples[i]
        print(f'Accuracy of {classes[i]}: {acc} %')
```

```
Accuracy of the CNN = 10.16 %
Accuracy of plane: 42.9 %
Accuracy of car: 35.3 %
Accuracy of bird: 43.0 %
Accuracy of cat: 8.7 %
Accuracy of deer: 16.9 %
Accuracy of dog: 68.7 %
Accuracy of frog: 58.5 %
Accuracy of horse: 53.3 %
Accuracy of ship: 59.1 %
Accuracy of truck: 65.3 %
```