TD 3: Convolutional Neural Network in PyTorch

1. Objective

In this TD, we will implement a Convolutional Neural Network (CNN) by using PyTorch. The steps in Review section will apply to classify the digits in MNIST database. In the Exercises section, you will create another CNN (more deeper) to classify the image in CIFAR10 dataset.

After this exercise, you will:

- Have a basic knowledge about CNN
- Know how to implement a CNN by using PyTorch library
- Have more understand about deep learning

2. Software and libraries:

Software and libraries require:

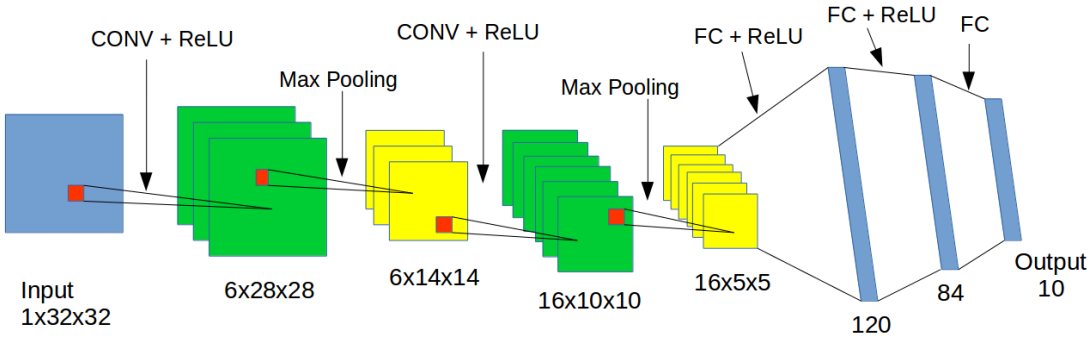
- Python
- Jupyter (iPython notebook)
- PyTorch
- Torchvision

3. Problem

In this exercise, we will implement CNN to solve classification problems by using PyTorch library.

4. Review:

The CNN model which we will implement is as following:



It includes:

- 2 convolutional layers (size of kernels are 5x5)

- 2 maximum pooling layers
- 3 fully connected layers
- All the hidden layer use ReLU activation functions, the last fully-connected layer using softmax activation function.

The process is as following:

- Load dataset
- Define the network architecture
- Define the loss function and update rule
- Train the model and test on several images

4.1. Load dataset

Implement the ***load_data()*** function to load MNIST dataset. Hints: You can see it in torchvision library.

```
import torchvision
import torchvision.transforms as transforms

# create a compose of transform operations
transform = transforms.Compose(
    [transforms.Resize(32),
     transforms.ToTensor(),
     transforms.Normalize((0.5,0.5,0.5), (0.5,0.5,0.5))])

#download train images from internet if it is not exists
trainset = torchvision.datasets.MNIST(root='./data', train=True,
                                      download=True, transform=transform)

# create the dataloader from downloaded images to use in PyTorch
trainloader = torch.utils.data.DataLoader(trainset, batch_size=4,
                                          shuffle=True, num_workers=2)
#download test images and create the loader to use in PyTorch
testset = torchvision.datasets.MNIST(root='./data', train=False,
                                     download=True, transform=transform)

testloader = torch.utils.data.DataLoader(testset, batch_size=4,
                                         shuffle=False, num_workers=2)

classes = ('zero', 'one', 'two', 'three', 'four',
           'five', 'six', 'seven', 'eight', 'nine')
```

4.2. Define the network architecture

Question 1: Implement the network architecture (above image) by inherited nn.Module.

```
1. import torch
2. import torch.nn as nn
3. import torch.nn.functional as F
4.
5. class Net(nn.Module):
6.     '''''
7.         Define the layers in the network
8.     '''
9.     def __init__(self):
10.         super(Net, self).__init__()
11.         # 1 input image channel, 6 output channels, 5x5 square convolution
    kernel
12.         self.conv1 = nn.Conv2d(1, 6, 5)
13.         self.conv2 = nn.Conv2d(6, 16, 5)
14.         # an affine operation: y = Wx + b
15.         self.fc1 = nn.Linear(16 * 5 * 5, 120) # (size of input, size of out
    put)
16.         self.fc2 = nn.Linear(120, 84)
17.         self.fc3 = nn.Linear(84, 10)
18.
19.     '''''
20.         Implement the forward computation of the network
21.     '''
22.     def forward(self, x):
23.         # Max pooling over a (2, 2) window
24.         x = F.max_pool2d(F.relu(self.conv1(x)), (2, 2))
25.         # If the size is a square you can only specify a single number
26.         x = F.max_pool2d(F.relu(self.conv2(x)), 2)
27.         x = x.view(-1, self.num_flat_features(x))
28.         x = F.relu(self.fc1(x))
29.         x = F.relu(self.fc2(x))
30.         x = self.fc3(x)
31.         return x
32.
33.     def num_flat_features(self, x):
34.         size = x.size()[1:]  # all dimensions except the batch dimension
35.         num_features = 1
36.         for s in size:
37.             num_features *= s
38.         return num_features
39.
40. net = Net()
41. print(net)
```

Question 2: Define the loss function (**CrossEnTropyLoss**) and update rule (**SGD**).

```
import torch.optim as optim
# Define the loss function
criterion = nn.CrossEntropyLoss()
# Define the optimizer
optimizer = optim.SGD(net.parameters(), lr=0.001, momentum=0.9)
```

4.3. Train and predict

Question 3: Implement ***train()*** function to train the network.

```
def train(epochs = 200):
    for epoch in range(epochs):
        running_loss = 0.0
        for i, data in enumerate(trainloader,0):
            #get the input
            inputs, labels = data

            # clear the parameter gradients
            optimizer.zero_grad()

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

            running_loss += loss.item()
            if i % 200 == 1999:    # print every 2000 mini-batches
                print('[%d, %5d] loss: %.3f' %
                      (epoch + 1, i + 1, running_l0oss / 2000))
                running_loss = 0.0
    print('Finished Training')
```

Question 4: Try to predict some images in test dataset.

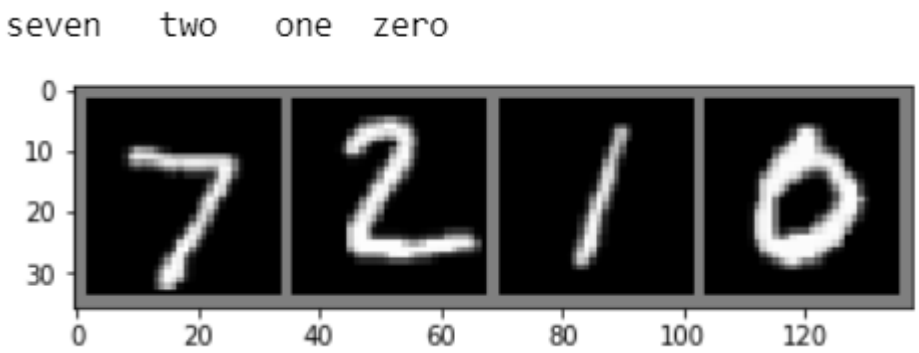
Load some images and display them.

```
dataiter = iter(testloader)
images,labels = dataiter.next()

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

#print labels
print(' '.join('%5s' % classes[labels[j]] for j in range(4)))
```

Results:



Predict the images by the network:

```
1. outputs = net(images)
2. _,predicted = torch.max(outputs,1)
3. print('Predicted: ', ' '.join('%5s' % classes[predicted[j]] for j in range(
   4)))
```

Results should be: `('Predicted: ', 'seven two one zero')`
Lets see how the network performs on the whole dataset:

```
1. correct = 0
2. total = 0
3. with torch.no_grad():
4.     for data in testloader:
5.         images, labels = data
6.         outputs = net(images)
7.         _, predicted = torch.max(outputs.data, 1)
8.         total += labels.size(0)
9.         correct += (predicted == labels).sum().item()
10.
11. print('Accuracy of the network on the 10000 test images: %d %%' % (
12.     100 * correct / total))
```

Result: `Accuracy of the network on the 10000 test images: 98 %`

5. Exercises

In this section, you will implement a CNN to solve another classification problem. Some attentions are:

- Your network will train and test on CIFAR10 (see in torchvision)
- You have to define the architecture of network by yourself.
- Try with different parameters