使用Cifar-10做影像分類

```
import torch
import torchvision
import torchvision.transforms as transforms
```

＋ 程式碼　＋ 文字

影像預處理

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

batch_size = 4    #1111

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

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

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

```
Downloading https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz to ./data/cifar-10-python.tar.gz
100%|██████████| 170498071/170498071 [00:02<00:00, 63599557.74it/s]
Extracting ./data/cifar-10-python.tar.gz to ./data
Files already downloaded and verified
```

Show Images

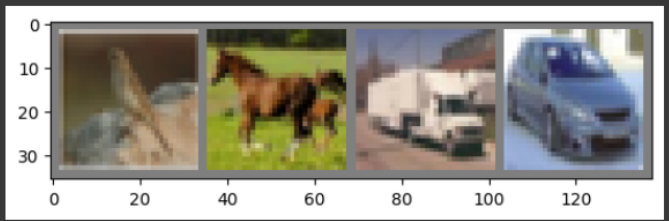
```
import matplotlib.pyplot as plt
import numpy as np

# functions to show an image
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(trainloader)
images, labels = next(dataiter)

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

# print labels
print(' '.join(f'{classes[labels[j]]:5s}' for j in range(batch_size)))
```

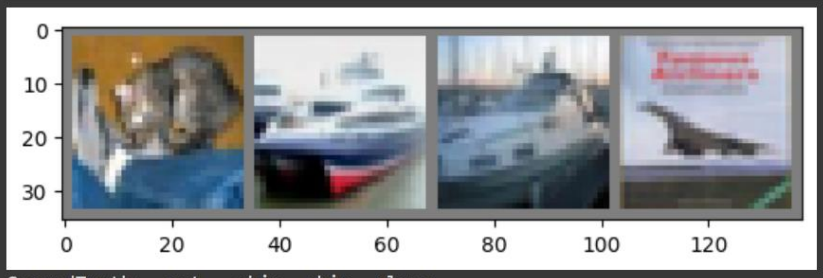



```
bird  horse truck car
```

進行測試

```
datatiter = iter(testloader)
images, labels = next(datatiter)

# print images
imshow(torchvision.utils.make_grid(images))
print('GroundTruth: ', ' '.join(f'{classes[labels[j]]:5s}' for j in range(4)))
```




```
GroundTruth:  cat   ship  ship  plane
```

```
[19] outputs = net(images)
```

```
[20] _, predicted = torch.max(outputs, 1)

print('Predicted: ', ' '.join(f'{classes[predicted[j]]:5s}'
                              for j in range(4)))
```

```
Predicted:  cat   ship  ship  plane
```

```
correct = 0
total = 0
# since we're not training, we don't need to calculate the gradients for our outputs
with torch.no_grad():
        for data in testloader:
                images, labels = data
                # calculate outputs by running images through the network
                outputs = net(images)
                # the class with the highest energy is what we choose as prediction
                _, predicted = torch.max(outputs.data, 1)
                total += labels.size(0)
                correct += (predicted == labels).sum().item()

print(f'Accuracy of the network on the 10000 test images: {100 * correct // total} %')
```

```
Accuracy of the network on the 10000 test images: 52 %
```

原本的 lr=0.001

```
[9] import torch.optim as optim

    criterion = nn.CrossEntropyLoss()
    optimizer = optim.SGD(net.parameters(), lr=0.001, momentum=0.9)      #2222
```

訓練卷積類神經網路

```
for epoch in range(2):   # loop over the dataset multiple times

        running_loss = 0.0
        for i, data in enumerate(trainloader, 0):
                # get the inputs; data is a list of [inputs, labels]
                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(f'[{epoch + 1}, {i + 1:5d}] loss: {running_loss / 2000:.3f}')
                        running_loss = 0.0

print('Finished Training')
```

```
[1,  2000] loss: 2.206
[1,  4000] loss: 1.882
[1,  6000] loss: 1.693
[1,  8000] loss: 1.585
[1, 10000] loss: 1.529
[1, 12000] loss: 1.474
[2,  2000] loss: 1.404
[2,  4000] loss: 1.369
[2,  6000] loss: 1.345
[2,  8000] loss: 1.341
[2, 10000] loss: 1.319
[2, 12000] loss: 1.308
Finished Training
```

lr 改成 0.02

```
[11] import torch.optim as optim

     criterion = nn.CrossEntropyLoss()
     optimizer = optim.SGD(net.parameters(), lr=0.02, momentum=0.9)      #2222  #lr改成0.02
```

訓練卷積類神經網路

```
for epoch in range(2):   # loop over the dataset multiple times

        running_loss = 0.0
        for i, data in enumerate(trainloader, 0):
                # get the inputs; data is a list of [inputs, labels]
                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(f'[{epoch + 1}, {i + 1:5d}] loss: {running_loss / 2000:.3f}')
                        running_loss = 0.0

print('Finished Training')
```

```
[1,  2000] loss: 2.190
[1,  4000] loss: 2.115
[1,  6000] loss: 2.135
[1,  8000] loss: 2.161
[1, 10000] loss: 2.144
[1, 12000] loss: 2.126
[2,  2000] loss: 2.116
[2,  4000] loss: 2.085
[2,  6000] loss: 2.121
[2,  8000] loss: 2.133
[2, 10000] loss: 2.197
[2, 12000] loss: 2.209
Finished Training
```

心得：

本次的實作演練主要是就 Image classification 的部分進行，並相較於課堂上看老師做給我們看，我自己操作起來更能加深學習的印象。後來，嘗試改了 learning rate 的部分，從 0.001 調整到 0.02，最大的差異在於 loss 在 0.001 的時候是逐漸向下遞減且遞減幅度漸減，而 loss 在 0.02 時沒有明顯遞減遞增，而是呈現跳動型態。