一、結果

(一) 固定 Batch_size = 4，改變 Learning rate

1

使用Cifar-10做影像分類

```
[61] import torch
     import torchvision
     import torchvision.transforms as transforms
```

影像預處理

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

     batch_size = 4    #1
     #batch_size = 8    #2
     #batch_size = 2    #3

     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')

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

2

Show Images

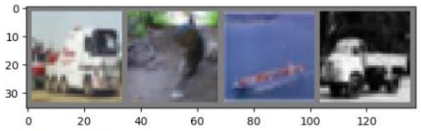
```
[14] import matplotlib.pyplot as plt
     import numpy as np

     # 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)))
```



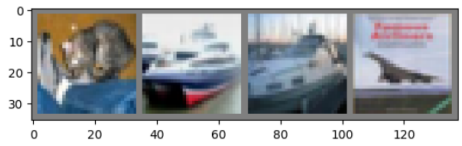
truck cat ship truck

3

進行測試

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

# 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

4.

Lr=0.1

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

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

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

Predicted: car car car car

```
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: 10 %

Lr=0.01

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

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

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

Predicted: ship plane ship ship

```
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: 25 %

Lr=0.001

Lr=0.0001

```
outputs = net(images)
```

```
_, 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: 55 %
```

```
outputs = net(images)
```

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

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

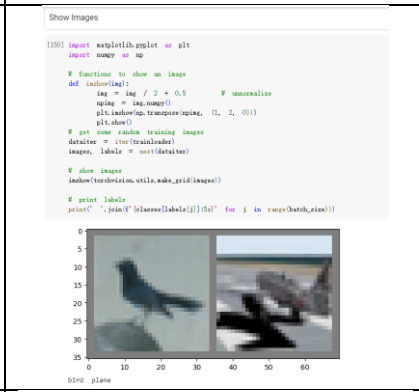
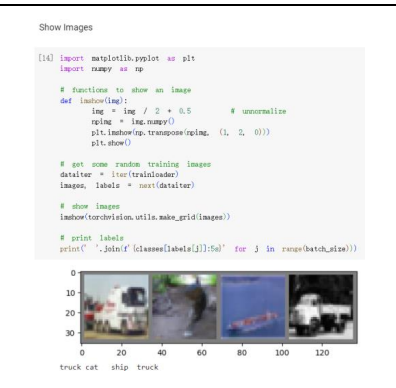
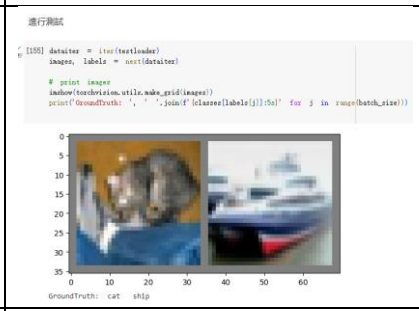
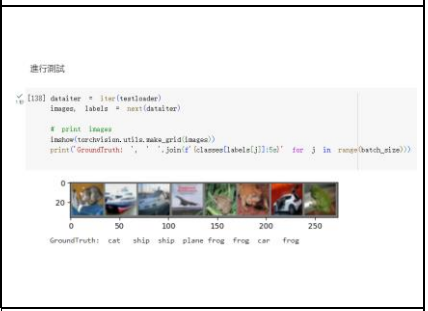
```
Predicted:  cat   ship  ship  ship
```

```
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: 36 %
```

(二) 固定 Learning rate = 0.001，改變 Batch_size

Batch size	2	4	8
1			
2			
3			
4			

心得:

在本實驗中，我們調整 Learning rate 與 Batch size 兩個參數，並觀察其對準確率的影響。我們以 10 的倍率(0.1、0.01、0.001、0.0001)調整 Learning rate，並且發現當 Learning rate 是 0.001 時，準

確度最高(55%)，而 Learning rate 降低會使準確率增加。我們皆以 2 的整數次方倍改變 Batch size，實驗結果發現 Batch size 對準確率僅有些微差異，Batch size=8 時的準確率最高，為 55%，當 Batch size 為 2 和 8 時，準確度皆為 52%。藉由調整參數優化模型，讓我們對於模型的訓練都更能掌握得宜。