

邊緣運算_HW2_B11002220

筆記區：

深度模型概念：

model optimizer scheduler 在深度學習扮演的角色

模型定義了從輸入到輸出的映射關係，優化器負責根據模型的表現更新其參數，而學習率調度器則在這一過程中適時調整優化器的學習率，幫助達到更好的訓練結果。

為什麼需要學習率一開始要高但要隨著訓練次數衰減？

1. 提高收斂速度

在訓練初期，模型距離最優解通常很遠，此時可以使用較大的學習率快速進行大幅度的參數更新。隨著模型接近最優解，使用較小的學習率可以進行更細微的調整，從而避免參數在最優解附近震盪，加快收斂速度。

2. 防止過度擬合

但隨著訓練的進行，模型可能會開始過度擬合訓練數據，即在訓練數據上表現很好，但在未見過的數據上表現不佳。逐步降低學習率可以幫助模型在學習的後期階段更加細緻地調整權重，從而提高其對未見數據的泛化能力。

訓練過程

1. `model.train()`

2. For batch in batchs

- a. 數據和標籤送入硬體
- b. 梯度歸零
- c. 向前傳播
- d. 計算損失
- e. 反向傳播
- f. 參數更新

ONNX是什麼?目的是?

ONNX (Open Neural Network Exchange) 是一種開放的文件格式，旨在讓不同的人工智能 (AI) 和機器學習框架能夠交換模型。它使得研究人員和開發人員能夠使用他們選擇的工具來訓練模型，然後輕鬆地將模型轉移到其他框架中，以便部署。

程式碼與解釋：

week3.py

我有試過多個內建的模型在內部跑的結果 我的CNN模型轉的結果最好是79%。而我還有試過其他幾款mobilenet在使用預訓練的情況下可以跑到83%。而Resnet18在使用預訓練的情況下可以跑到92%。可以看出來模型架構對結果影響是很深遠的。我們最初在跑CNN時我們嘗試加過更多的捲機層進去，剛加一兩層時，效果有顯著提升到79%。但是隨者我們持續加深，擬和的情況加深反而讓我們最後的結果掉下去76%。

```
1 # Modified from PyTorch examples:
2 # https://github.com/pytorch/examples/blob/master/cifar10/main.py
3 #
4 from __future__ import print_function
5 import argparse
6 import torch
7 import torch.nn as nn
8 import torch.nn.functional as F
9 import torch.optim as optim
10 from torchvision import datasets, transforms
11 from torch.optim.lr_scheduler import StepLR
12 import torch.nn as nn
13 import torchvision.models as models
14
```

```

15 # 定義網絡結構
16 class Net(nn.Module):
17     def __init__(self, num_classes=10):
18         super(Net, self).__init__()
19         self.resnet = models.resnet18(pretrained=True) # 使用預訓練的ResNet18
20         num_fters = self.resnet.fc.in_features # 獲取全連接層的輸入特徵數
21         self.resnet.fc = nn.Linear(num_fters, num_classes) # 由於原始的 ResNet18
是針對 ImageNet 數據集預訓練的，其全連接層的輸出維度對應於 ImageNet 的類別數（1000
類別）。對於 CIFAR-10 數據集（10類），需要將這個全連接層替換為一個新的線性層，其輸出維度為
CIFAR-10 的類別數。
22         # 修改第一層卷積層和移除最大池化層
23         self.resnet.conv1 = nn.Conv2d(3, 64, kernel_size=3, stride=1,
padding=1, bias=False) # 修改第一層捲機層以讓他適應新的輸入
24         self.resnet.maxpool = nn.Identity() # 移除最大池化層，保留原始細節。
25
26     def forward(self, x):
27         return self.resnet(x)
28
29 def train(args, model, device, train_loader, optimizer, epoch):
30     model.train() # 告訴硬體開始訓練 train表示可能會頻繁的更動參數
31     for batch_idx, (data, target) in enumerate(train_loader): # trainloader 會把
資料切成一筆一筆
32         data, target = data.to(device), target.to(device)
33         optimizer.zero_grad()
34         output = model(data)
35         loss = F.cross_entropy(output, target)
36         loss.backward()
37         optimizer.step()
38         if batch_idx % args.log_interval == 0:
39             print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'.format(
40                 epoch, batch_idx * len(data), len(train_loader.dataset),
41                 100. * batch_idx / len(train_loader), loss.item()))
42             if args.dry_run:
43                 break
44
45 def test(model, device, test_loader):
46     model.eval() # 將模型設置為評估模式。這會通知所有層在評估時應禁用特定行為，如
dropout 層的隨機失活。
47     test_loss = 0
48     correct = 0
49     with torch.no_grad(): #禁止某評估計算時儲存梯度，以節省記憶體與計算資源
50         for data, target in test_loader: #載入標籤與資料
51             data, target = data.to(device), target.to(device) #送入硬體做處理
52             output = model(data) #經過計算取得節骨
53             test_loss += F.cross_entropy(output, target,
reduction='sum').item() # 計算Loss 大小 loss 越小表示預測的與實際更近

```

```

54         pred = output.argmax(dim=1, keepdim=True) # 從一維的標籤中尋找概率最高的
           作為標籤輸入
55         correct += pred.eq(target.view_as(pred)).sum().item() # 增加正確數量
56
57     test_loss /= len(test_loader.dataset)
58
59     print('\nTest set: Average loss: {:.4f}, Accuracy: {}/{}

```

```

94     device = torch.device("cpu")
95
96     train_kwargs = {'batch_size': args.batch_size}
97     test_kwargs = {'batch_size': args.test_batch_size} # 設定成測試集的大小
98     if use_cuda:
99         cuda_kwargs = {'num_workers': 1,
100                        'pin_memory': True,
101                        'shuffle': True}
102         train_kwargs.update(cuda_kwargs) #把新的設定更新到train_args 內
103         test_kwargs.update(cuda_kwargs)
104
105     transform = transforms.Compose([
106         transforms.ToTensor(),
107         transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
108     ])
109     # 載入訓練集及測試集
110     dataset1 = datasets.CIFAR10('./data', train=True, download=True,
111                                transform=transform)
112     dataset2 = datasets.CIFAR10('./data', train=False,
113                                transform=transform)
114     # DataLoader(dataset, batch_size=1, shuffle=False,
115     # sampler=None, batch_sampler=None, num_workers=0,
116     # collate_fn=None, pin_memory=False, drop_last=False,
117     # timeout=0, worker_init_fn=None, *, prefetch_factor=2, persistent_workers=False)
118     train_loader = torch.utils.data.DataLoader(dataset1, **train_kwargs) # 把對應
119     # 的參數對過關鍵字解包的方式載入
120     test_loader = torch.utils.data.DataLoader(dataset2, **test_kwargs)
121     # 載入模型並開始訓練
122     model = Net().to(device) #模型載入到cuda
123     optimizer = optim.Adam(model.parameters(), lr=args.lr) #優化器是根據模型在訓練
124     # 數據上的表現（通常是損失函數的值），來更新和調整模型的參數，以便減少損失函數的值，從而提高
125     # 模型的準確性。
126
127     scheduler = StepLR(optimizer, step_size=1, gamma=args.gamma) #Scheduler 通過
128     # 預定的策略在訓練過程中動態調整學習率，例如隨著訓練進展逐步降低學習率，以幫助模型更細緻地接
129     # 近全局最小損失。
130
131     for epoch in range(1, args.epochs + 1):
132         train(args, model, device, train_loader, optimizer, epoch) # 訓練一次模型
133         test(model, device, test_loader) # 測試模型輸出
134         scheduler.step() # 學習率調整
135     # 儲存模型
136     torch.save(model.state_dict(), "cifar10_model.pt")
137
138 if __name__ == '__main__':
139     main()
140

```

Export.py

這個檔案是把pytorch的權重跟模型轉換成onnx格式。方便後續透過onnx做推論。

```
1 import os
2 import argparse
3 import torch
4 import torch.nn as nn
5 import torch.nn.functional as F
6 from torchvision import datasets, transforms, models
7
8 import onnx
9 # 定義網絡結構
10 class Net(nn.Module):
11     def __init__(self, num_classes=10):
12         super(Net, self).__init__()
13         self.resnet = models.resnet18(pretrained=True) # 使用預訓練的ResNet18
14         num_fts = self.resnet.fc.in_features # 獲取全連接層的輸入特徵數
15         self.resnet.fc = nn.Linear(num_fts, num_classes) # 由於原始的 ResNet18
16         # 是針對 ImageNet 數據集預訓練的，其全連接層的輸出維度對應於 ImageNet 的類別數（1000
17         # 類）。對於 CIFAR-10 數據集（10類），需要將這個全連接層替換為一個新的線性層，其輸出維度為
18         # CIFAR-10 的類別數。
19         # 修改第一層卷積層和移除最大池化層
20         self.resnet.conv1 = nn.Conv2d(3, 64, kernel_size=3, stride=1,
21 padding=1, bias=False) # 修改第一層捲機層以讓他適應新的輸入
22         self.resnet.maxpool = nn.Identity() # 移除最大池化層，保留原始細節。
23
24     def forward(self, x):
25         return self.resnet(x)
26
27 #取得存這個檔案的資料夾名稱
28 package_dir = os.path.dirname(os.path.abspath(__file__))
29 default_model_path = os.path.join(package_dir, 'cifar10_model.pt') # 把路徑與模型
30 #組合成模型絕對路徑
31 parser = argparse.ArgumentParser(description='PyTorch cifar10 Predictor') # 參數
32 #設定
33 parser.add_argument('--model', type=str, default=default_model_path,
34 help='model for prediction (default:
35 {})'.format(default_model_path)) #參數設定type
36
37 args = parser.parse_args()
38 # 使用指定的模型檔案路徑
39 model = Net() #採用Net類別的模型
40 model_r = torch.load(default_model_path, map_location="cuda") #
41 model.load_state_dict(model_r) #保存
42 model.eval()
43
44
```

```
35 x = torch.randn(1, 3, 32, 32, requires_grad=True)
36 torch.onnx.export(model, x, 'cifar10.onnx', input_names=['input'],
    output_names=['output'], verbose=False)
37
```

onnxinfer.py

測試ONNX平台跑多筆資料的速度。

```
1 import onnxruntime
2 from torchvision import transforms, models
3 from PIL import Image
4 import numpy as np
5 import torch.nn.functional as F
6 import torch
7 import argparse
8 import time
9
10 def load_image(image_path):
11     image = Image.open(image_path).convert('RGB')
12     transform = transforms.Compose([#創建一個轉換序列
13         transforms.Resize((32, 32)), #將圖像大小調整為32x32像素。
14         transforms.ToTensor(), #將Pillow圖像或NumPy ndarray轉換為torch.Tensor。這也
        將圖像的像素值從0-255縮放到0-1之間。
15         transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)) #標準化圖像的每個通
        道。給定的均值（第一個三元組）和標準差（第二個三元組）用於將像素值進一步縮放。這裡，它將0-
        1範圍的像素值轉換為-1到1。
16     ])
17     image = transform(image).unsqueeze(0)
18     return image
19 def main():
20     parser = argparse.ArgumentParser(description='ONNX CIFAR-10 Inference')
21     parser.add_argument('--img', type=str, required=True, help='path to the
        input image')
22     args = parser.parse_args()
23
24     # 載入 ONNX 模型並指定 GPU 運行
25     session = onnxruntime.InferenceSession("./cifar10.onnx", providers=
        ['CUDAExecutionProvider'])
26
27     # 讀取並預處理圖片
28     image = load_image(args.img).to("cuda")
29     input_dict = {session.get_inputs()[0].name: image.cpu().numpy()}
30
31     # 進行推理
```



```

32     t0 = time.time()
33     for i in range(10000):
34         output = session.run(None, input_dict)
35     t1 = time.time()
36
37     # 將原始分數轉換為概率
38     probabilities = F.softmax(torch.tensor(output[0]), dim=1)
39
40     # 獲取最大概率對應的類別
41     predicted_class = torch.argmax(probabilities, dim=1).item()
42     labels = ["飛機", "汽車", "鳥", "貓", "鹿", "狗", "青蛙", "馬", "船", "卡車"]
43     print("概率分佈: ", probabilities.numpy())
44     print("預測類別: ", labels[predicted_class])
45     print('ONNX推論10000次消耗時間', int(t1-t0), 's')
46
47 if __name__ == '__main__':
48     main()
49

```

pytorchinfer.py

測試Pytorch平台在推論多筆資料的速度。

```

1  import torch
2  import torch.nn as nn
3  import torch.nn.functional as F
4  from torchvision import transforms, models
5  from PIL import Image
6  import argparse
7  import time
8
9  # 定義網絡結構
10 class Net(nn.Module):
11     def __init__(self, num_classes=10):
12         super(Net, self).__init__()
13         self.resnet = models.resnet18(pretrained=True) # 使用預訓練的ResNet18
14         num_ftrs = self.resnet.fc.in_features # 獲取全連接層的輸入特徵數
15         self.resnet.fc = nn.Linear(num_ftrs, num_classes) # 替換全連接層
16         # 修改第一層卷積層和移除最大池化層
17         self.resnet.conv1 = nn.Conv2d(3, 64, kernel_size=3, stride=1,
padding=1, bias=False)
18         self.resnet.maxpool = nn.Identity() # 移除最大池化層
19
20     def forward(self, x):
21         return self.resnet(x)

```



```

22
23 def load_image(image_path):
24     image = Image.open(image_path).convert('RGB')
25     transform = transforms.Compose([
26         transforms.Resize((32, 32)), #將圖像大小調整為32x32像素。
27         transforms.ToTensor(), #將Pillow圖像或NumPy ndarray轉換為torch.Tensor。這也
    將圖像的像素值從0-255縮放到0-1之間。
28         transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
29     ])
30     image = transform(image).unsqueeze(0)
31     return image
32
33 def main():
34     parser = argparse.ArgumentParser(description='PyTorch CIFAR-10 Inference')
35     parser.add_argument('--img', type=str, required=True, help='path to the
    input image')
36     args = parser.parse_args()
37
38     device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
39
40     model = Net().to(device)
41     model.load_state_dict(torch.load("cifar10_model.pt", map_location=device))
42     model.eval()
43
44     with torch.no_grad():
45         image = load_image(args.img).to(device)
46         # 進行推理
47         t0 = time.time()
48         for i in range(10000):
49             output = model(image)
50             t1 = time.time()
51             probabilities = F.softmax(output, dim=1)
52             predicted_class = torch.argmax(probabilities, dim=1).item()
53
54     labels = ["飛機", "汽車", "鳥", "貓", "鹿", "狗", "青蛙", "馬", "船", "卡車"]
55     print("概率分佈: ", probabilities)
56     print("預測類別: ", labels[predicted_class])
57     print('PyTorch推論10000次消耗時間', int(t1-t0), 's')
58
59 if __name__ == '__main__':
60     main()
61

```

結果圖：

模型訓練效果

CNN 嘗試很多解法，不過我們最高也只把 CNN 的模型調到79%與 Resnet 模型 91% 還是有很大的差距。

```
Train Epoch: 4 [0/50000 (0%)] Loss: 0.047017
Train Epoch: 4 [2560/50000 (5%)] Loss: 0.070151
Train Epoch: 4 [5120/50000 (10%)] Loss: 0.045667
Train Epoch: 4 [7680/50000 (15%)] Loss: 0.025545
Train Epoch: 4 [10240/50000 (20%)] Loss: 0.029990
Train Epoch: 4 [12800/50000 (26%)] Loss: 0.037951
Train Epoch: 4 [15360/50000 (31%)] Loss: 0.022666
Train Epoch: 4 [17920/50000 (36%)] Loss: 0.032338
Train Epoch: 4 [20480/50000 (41%)] Loss: 0.028089
Train Epoch: 4 [23040/50000 (46%)] Loss: 0.043432
Train Epoch: 4 [25600/50000 (51%)] Loss: 0.005883
Train Epoch: 4 [28160/50000 (56%)] Loss: 0.055886
Train Epoch: 4 [30720/50000 (61%)] Loss: 0.030181
Train Epoch: 4 [33280/50000 (66%)] Loss: 0.036334
Train Epoch: 4 [35840/50000 (71%)] Loss: 0.026204
Train Epoch: 4 [38400/50000 (77%)] Loss: 0.029126
Train Epoch: 4 [40960/50000 (82%)] Loss: 0.017099
Train Epoch: 4 [43520/50000 (87%)] Loss: 0.030636
Train Epoch: 4 [46080/50000 (92%)] Loss: 0.023616
Train Epoch: 4 [48640/50000 (97%)] Loss: 0.042987

Test set: Average loss: 0.3377, Accuracy: 9077/10000 (91%)
```

ResNet18

```
Train Epoch: 16 [0/50000 (0%)] Loss: 0.648172
Train Epoch: 16 [2560/50000 (5%)] Loss: 0.559666
Train Epoch: 16 [5120/50000 (10%)] Loss: 0.513309
Train Epoch: 16 [7680/50000 (15%)] Loss: 0.614287
Train Epoch: 16 [10240/50000 (20%)] Loss: 0.587342
Train Epoch: 16 [12800/50000 (26%)] Loss: 0.593466
Train Epoch: 16 [15360/50000 (31%)] Loss: 0.566662
Train Epoch: 16 [17920/50000 (36%)] Loss: 0.535327
Train Epoch: 16 [20480/50000 (41%)] Loss: 0.603542
Train Epoch: 16 [23040/50000 (46%)] Loss: 0.587627
Train Epoch: 16 [25600/50000 (51%)] Loss: 0.618910
Train Epoch: 16 [28160/50000 (56%)] Loss: 0.571292
Train Epoch: 16 [30720/50000 (61%)] Loss: 0.501184
Train Epoch: 16 [33280/50000 (66%)] Loss: 0.522760
Train Epoch: 16 [35840/50000 (71%)] Loss: 0.421309
Train Epoch: 16 [38400/50000 (77%)] Loss: 0.497884
Train Epoch: 16 [40960/50000 (82%)] Loss: 0.495090
Train Epoch: 16 [43520/50000 (87%)] Loss: 0.462805
Train Epoch: 16 [46080/50000 (92%)] Loss: 0.538615
Train Epoch: 16 [48640/50000 (97%)] Loss: 0.633777

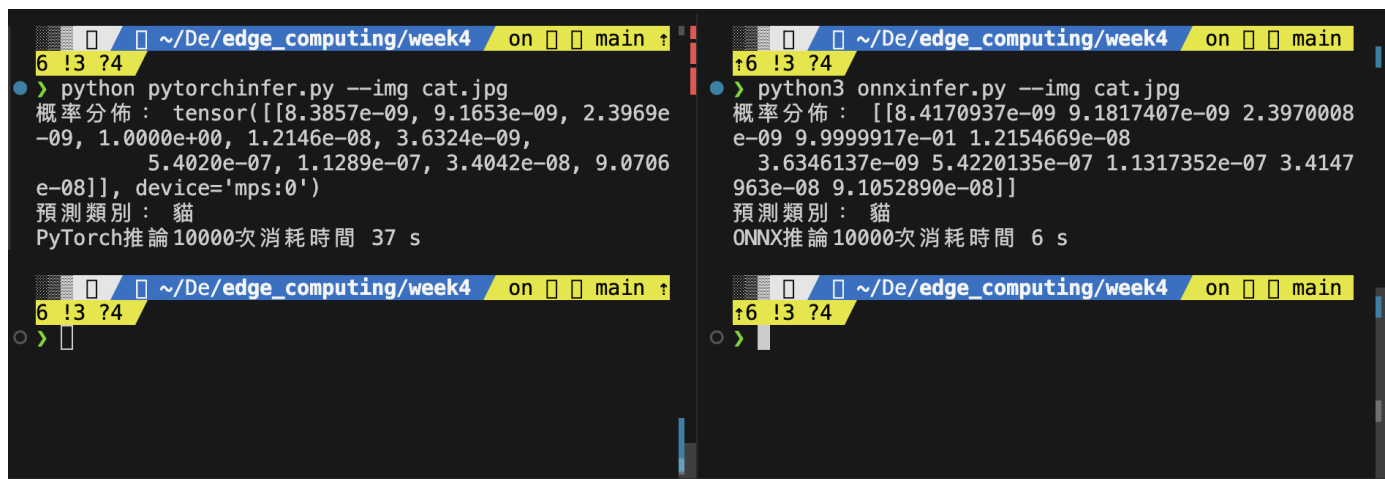
Test set: Average loss: 0.6032, Accuracy: 7893/10000 (79%)
```

CNN

測試 pytorch 平台與 onnx 平台的速度差異（左pytorch , 右onnx）

Mac M2:

我嘗試在我自己的電腦上跑看看這個模型，可以看到結果非常明顯。pytorch對於mac m2加速的速度完全低於ONXX的效果。



```
~/De/edge_computing/week4 on main
6 !3 ?4
> python pytorchinfer.py --img cat.jpg
概率分佈: tensor([[8.3857e-09, 9.1653e-09, 2.3969e-09, 1.0000e+00, 1.2146e-08, 3.6324e-09, 5.4020e-07, 1.1289e-07, 3.4042e-08, 9.0706e-08]], device='mps:0')
預測類別: 貓
PyTorch推論 10000次消耗時間 37 s

~/De/edge_computing/week4 on main
6 !3 ?4
> python3 onnxinfer.py --img cat.jpg
概率分佈: [[8.4170937e-09 9.1817407e-09 2.3970008e-09 9.9999917e-01 1.2154669e-08 3.6346137e-09 5.4220135e-07 1.1317352e-07 3.4147963e-08 9.1052890e-08]]
預測類別: 貓
ONNX推論 10000次消耗時間 6 s
```

ORIN NANO 1000筆資料:

下圖是輸入船的圖片，我們從結果可以發現Pytorch跑的時間比Onnx慢了1秒。

```
jetson@jetson-orin-nano:~/Downloads/week4_2hr$ python3 pytorchinfer.py --img ship.png
概率分佈: tensor([[4.7530e-04, 3.0954e-05, 7.5339e-07, 7.4022e-05, 1.4335e-06, 2.0431e-06,
1.6288e-06, 4.0533e-07, 9.9940e-01, 1.0893e-05]], device='cuda:0')
預測類別: 船
PyTorch推論1000次消耗時間 11 s
jetson@jetson-orin-nano:~/Downloads/week4_2hr$
```

```
jetson@jetson-orin-nano:~/Downloads/week4_2hr$ python3 onnxinfer.py --img ship.png
RuntimeError: module compiled against API version 0x10 but this version of numpy is 0x11
ImportError: numpy.core.multiarray failed to import

The above exception was the direct cause of the following exception:

SystemError: <built-in function __import__> returned a result with an error set
概率分佈: [[4.74505912e-04 3.09100105e-05 7.51575783e-07 7.38648305e-05
1.43061709e-06 2.03824220e-06 1.62554772e-06 4.04459939e-07
9.99403590e-01 1.08791355e-05]]
預測類別: 船
ONNX推論1000次消耗時間 10 s
jetson@jetson-orin-nano:~/Downloads/week4_2hr$
```

ORIN NANO 10000筆資料:

下圖是輸入船的圖片，數量是10000筆，我們從結果可以發現Pytorch需要85秒，而Onnx需要73秒。足足比Onnx慢了12秒才完成任務。

```
jetson@jetson-orin-nano:~/Downloads/week4_2hr$ python3 pytorchinfer.py --img ship.png
概率分佈: tensor([[4.7530e-04, 3.0954e-05, 7.5339e-07, 7.4022e-05, 1.4335e-06, 2.0431e-06,
1.6288e-06, 4.0533e-07, 9.9940e-01, 1.0893e-05]], device='cuda:0')
預測類別: 船
PyTorch推論10000次消耗時間 85 s
jetson@jetson-orin-nano:~/Downloads/week4_2hr$
```

```
jetson@jetson-orin-nano:~/Downloads/week4_2hr$ python3 onnxinfer.py --img ship.png
RuntimeError: module compiled against API version 0x10 but this version of numpy is 0x11 . Check the
y.org/devdocs/user/troubleshooting-importerror.html#api-incompatibility for indications on how to
ImportError: numpy.core.multiarray failed to import

The above exception was the direct cause of the following exception:

SystemError: <built-in function __import__> returned a result with an error set
概率分佈: [[4.7501546e-04 3.0945004e-05 7.5248687e-07 7.3956769e-05 1.4326450e-06
2.0413167e-06 1.6276782e-06 4.0499233e-07 9.9940300e-01 1.0891189e-05]]
預測類別: 船
ONNX推論10000次消耗時間 73 s
jetson@jetson-orin-nano:~/Downloads/week4_2hr$
```

結論:

這次Lab我學到如何訓練模型的概念，儘管對於實際的運作原理不是很熟悉，但是我查了很多資補了很多知識，也讓我對於訓練模型產生很大的興趣

就上述的模型訓練效果對比圖，我有想過一個新的辦法就是把CNN MaxPool 移除因為 Cifra-10 的數據集都是屬於小型圖片(32*32)。然後如果圖片小就盡量保持它的特徵避免在運算的過程中消失。再來從測試不同平台推論的速度不難發現，ONNX在推論中不論是在 Mac M2 或是在 ORIN NANO 都有不錯加速的效果。