110066540 陳哲瑋

National Tsing Hua University

Fall 2023 11210IPT 553000

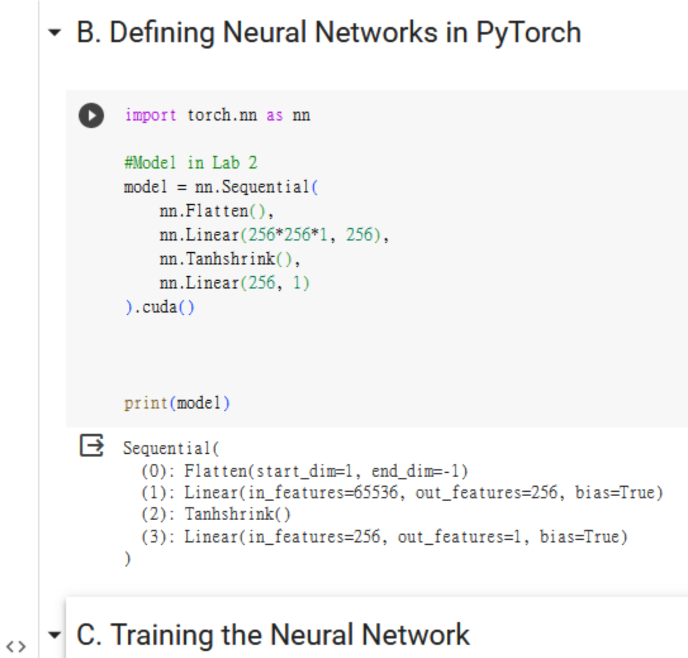
Deep Learning in Biomedical Optical Imaging

Homework 2

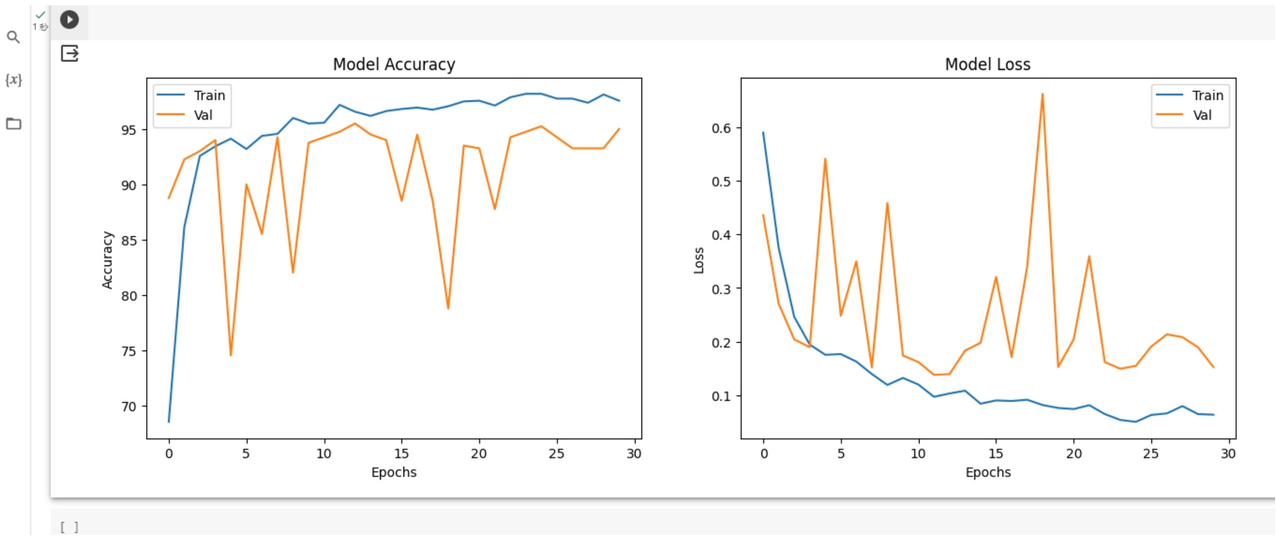
Coding:

1.1 Task A: Transitioning to Cross-Entropy Loss:

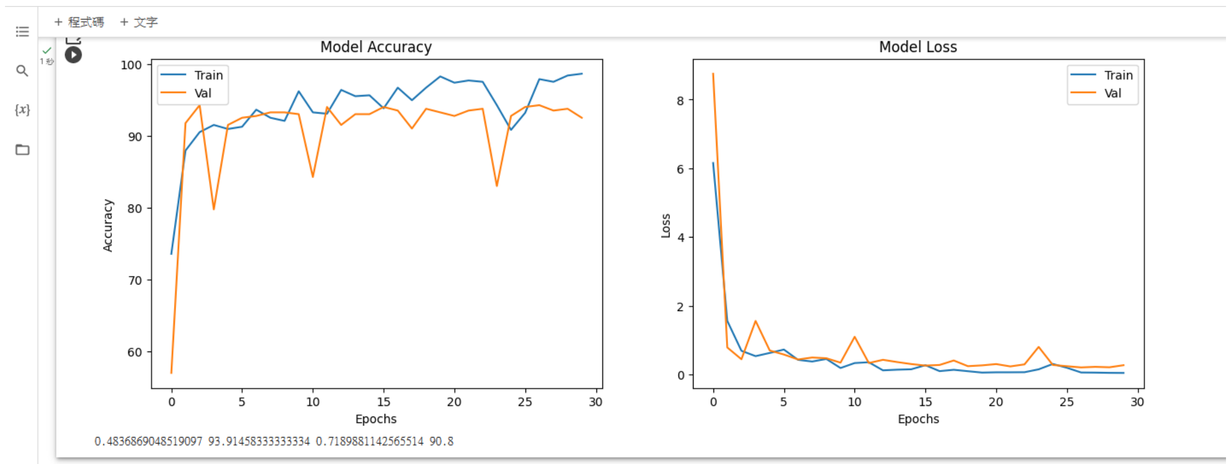
要將 BCE 改成 CE 首先需要使用 CrossEntropyLoss 替換掉 BCEWithLogiteLoss，其中 BCE 與 CE 有個最大的差距就是 BCE 是二元分類故要將最後一層 Node 從 1 改為 16。



1.2 Task B: Creating a Evaluation Code:



上圖為用上課 lab3 的程式跑出來的結果，Train Accuracy 以及 Val Accuracy 之間的差距很大，總合之前的判斷它應該是 over fitting 了，故就用之前的方法將將 neural network 的層數降為一層，下圖為重新訓練的結果，可以看到 Train Accuracy 與 Val Accurac 之間的差距變小，Val Loss 也有下降的跡象。



Report:

2.1 Task A: Performance between BCE loss and BC loss:

在此我們用四項參數的平均值來做分析

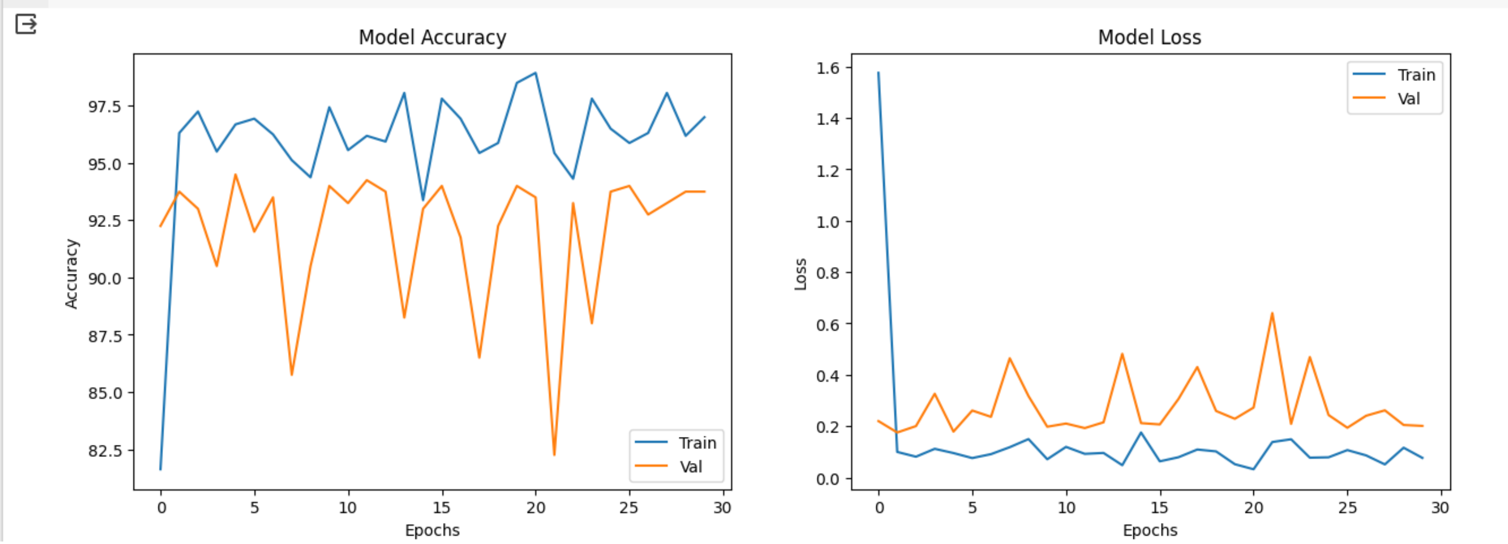
```
print(f'Epoch {epoch+1}/{epochs}, Train Loss: {avg_train_loss:.4f}, Tra
w+=avg_train_loss
x+=train_accuracy
y+=avg_val_loss
z+=val_accuracy
# Learning rate update
lr_scheduler.step()

# Checkpoint
if avg_val_loss < best_val_loss:
    best_val_loss = avg_val_loss
    torch.save(model.state_dict(), 'model_classification.pth')

# Store performance
train_losses.append(avg_train_loss)
train_accuracies.append(train_accuracy)
val_losses.append(avg_val_loss)
val_accuracies.append(val_accuracy)
```

```
Epoch 1/30, Train Loss: 1.5758, Train Accuracy: 81.62%, Val Loss: 0.2198, V
Epoch 2/30, Train Loss: 0.0997, Train Accuracy: 96.31%, Val Loss: 0.1754, V
Epoch 3/30, Train Loss: 0.0811, Train Accuracy: 97.25%, Val Loss: 0.2003, V
Epoch 4/30, Train Loss: 0.1118, Train Accuracy: 95.50%, Val Loss: 0.3263, V
Epoch 5/30, Train Loss: 0.0953, Train Accuracy: 96.69%, Val Loss: 0.1789, V
Epoch 6/30, Train Loss: 0.0759, Train Accuracy: 96.94%, Val Loss: 0.2610, V
Epoch 7/30, Train Loss: 0.0911, Train Accuracy: 96.25%, Val Loss: 0.2364, V
Epoch 8/30, Train Loss: 0.1182, Train Accuracy: 95.12%, Val Loss: 0.4644, V
```

Visualizing model performance

```
import matplotlib.pyplot as plt

fig, ax = plt.subplots(1, 2, figsize=(15, 5))

# Plotting training and validation accuracy
ax[0].plot(train_accuracies)
ax[0].plot(val_accuracies)
ax[0].set_title('Model Accuracy')
ax[0].set_xlabel('Epochs')
ax[0].set_ylabel('Accuracy')
ax[0].legend(['Train', 'Val'])

# Plotting training and validation loss
ax[1].plot(train_losses)
ax[1].plot(val_losses)
ax[1].set_title('Model Loss')
ax[1].set_xlabel('Epochs')
ax[1].set_ylabel('Loss')
ax[1].legend(['Train', 'Val'])

plt.show()
print(w/30,x/30,y/30,z/30)
```

下圖為第一次訓練單層 CE 所得到的圖:



	avg_train_loss	train_accuracy	avg_val_loss	val_accuracy
BCE	0.237	93.78%	0.259	92.55%
CE	0.1639	92.92%	0.1852	91.97%

再經過 3 次訓練後

	avg_train_loss	train_accuracy	avg_val_loss	val_accuracy
BCE	0.012	99.72%	0.317	93.66%
CE	0.0912	98.22%	0.276	93.23%

BCE 的 avg_train_loss 經過訓練後有所下降，avg_val_loss 以及 Val Accuracy 變化不大，反而 train_accuracy 有大幅度的上升，這代表 BCE 只需要可能 1~2 次訓練就有非常好的效果，就結果來說 CE 的 train_accuracy 及 val_accuracy 都有所上升，證明這個訓練架構是很有效的。

2.2 Task B: Performance between Different Hyperparameters:

首先先進行在相同單層 BCE 的環境下更改 ReLU、Leaky ReLU、Tanhchrink 同時一樣取四項參數的平均來分析

	avg_train_loss	train_accuracy	avg_val_loss	val_accuracy
ReLU	0.237	93.78%	0.259	92.55%
Leaky ReLU	0.268	93.36%	0.22	92.18%
Tanhchrink	0.619	93.97%	0.3659	92.18%

再經過 3 次訓練後

	avg_train_loss	train_accuracy	avg_val_loss	val_accuracy
ReLU	0.0069	99.83%	0.2786	94.4%
Leaky ReLU	0.015	99.50%	0.375	93.48%
Tanhchrink	0.054	98.61%	0.406	93.275%

從結果可以發現 ReLU 擁有最小的 avg_train_loss 及最高的正確率，說明用 ReLU 是對於訓練的效果是最好的。

B. Defining Neural Networks in PyTorch

```
import torch.nn as nn

#Model in Lab 2
model = nn.Sequential(
    nn.Flatten(),
    nn.Linear(256*256*1, 256),
    nn.Tanhshrink(),
    nn.Linear(256, 1)
).cuda()



print(model)
```

接著我要改的是 batch size 原本的大小為 32，分別使用 64 及 16 來測試，並與上面一樣在相同單層 BCE 的環境下同時一樣取四項參數的平均來分析：

batch size	avg_train_loss	train_accuracy	avg_val_loss	val_accuracy
64	0.325	94.5%	0.238	91.44%
32	0.237	94.1%	0.259	92.55%
16	0.20	93.8%	0.2259	92.35%

再經過 3 次訓練後

batch size	avg_train_loss	train_accuracy	avg_val_loss	val_accuracy
64	0.0379	98.95%	0.2155	93.91%
32	0.012	99.72%	0.317	93.66%
16	0.051	99.24%	0.2950	93.46%

可以看出經過 3 次訓練後較小的 batch size 對於 train_accuracy 的提升相當的大，val_accuracy 也有小幅度的提升，說明使用較小的 batch size 是相當不錯的。

```
y_train = np.concatenate((abnormal_labels[:split_point], normal_labels[:split_point]), axis=0)
x_val = np.concatenate((abnormal_scans[split_point:], normal_scans[split_point:]), axis=0)
y_val = np.concatenate((abnormal_labels[split_point:], normal_labels[split_point:]), axis=0)

# Convert to PyTorch tensors
x_train = torch.from_numpy(x_train).float()
y_train = torch.from_numpy(y_train).long()
x_val = torch.from_numpy(x_val).float()
y_val = torch.from_numpy(y_val).long()

# Create datasets
train_dataset = TensorDataset(x_train, y_train)
val_dataset = TensorDataset(x_val, y_val)

# Create dataloaders
train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=32, shuffle=False)

print(f'Number of samples in train and validation are {len(train_loader.dataset)} and {len(val_loader.dataset)}.')
print(f'X_train: max value is {x_train.max().item()}, min value is {x_train.min().item()}, data type is {x_train.dtype}.')
```

```
Shape of abnormal_scans: (1000, 256, 256)
Shape of normal_scans: (1000, 256, 256)
Number of samples in train and validation are 1600 and 400.
X_train: max value is 255.0, min value is 0.0, data type is torch.float32.
```