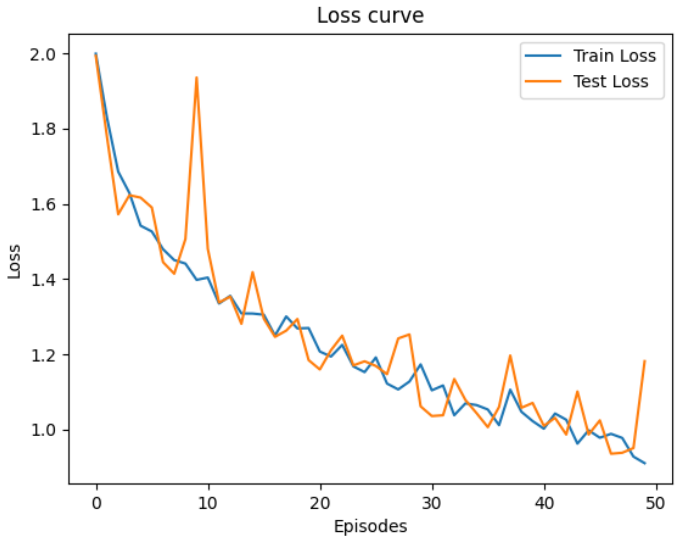
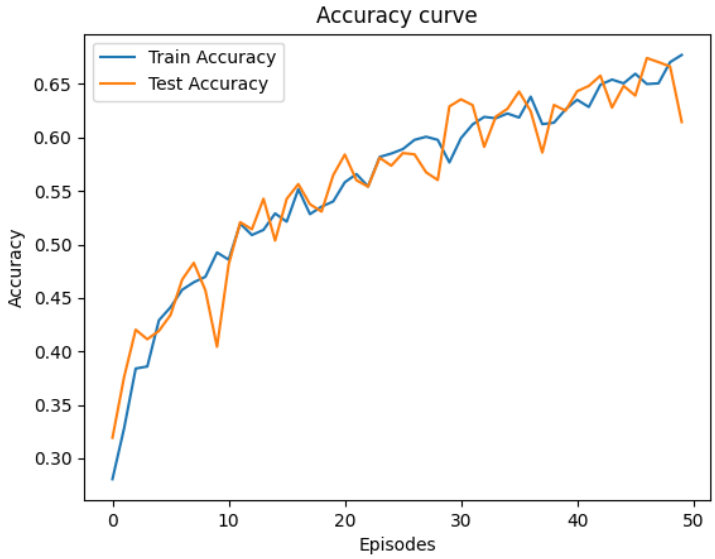
Discussion

(30%) The relationship between the accuracy, model size, and the training dataset size.
(Total 6 models. Small model trains on the sixteenth, half, and all data. Big model trains on the sixteenth, half, and all data. If the result is different from Fig.1, please explain the possible reasons.)

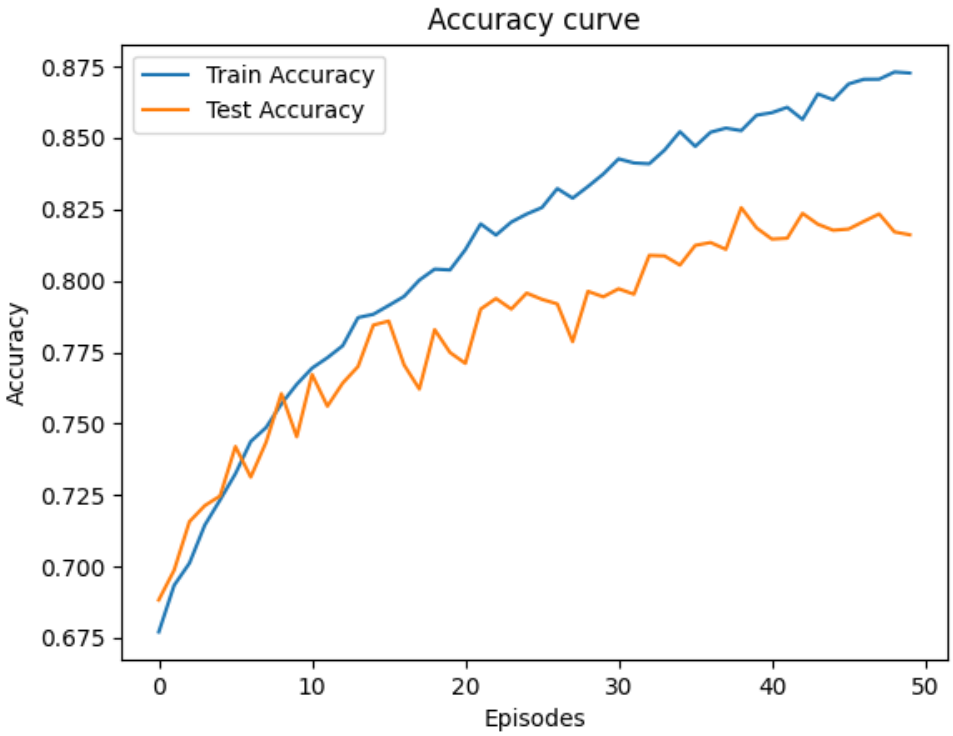
Ans:

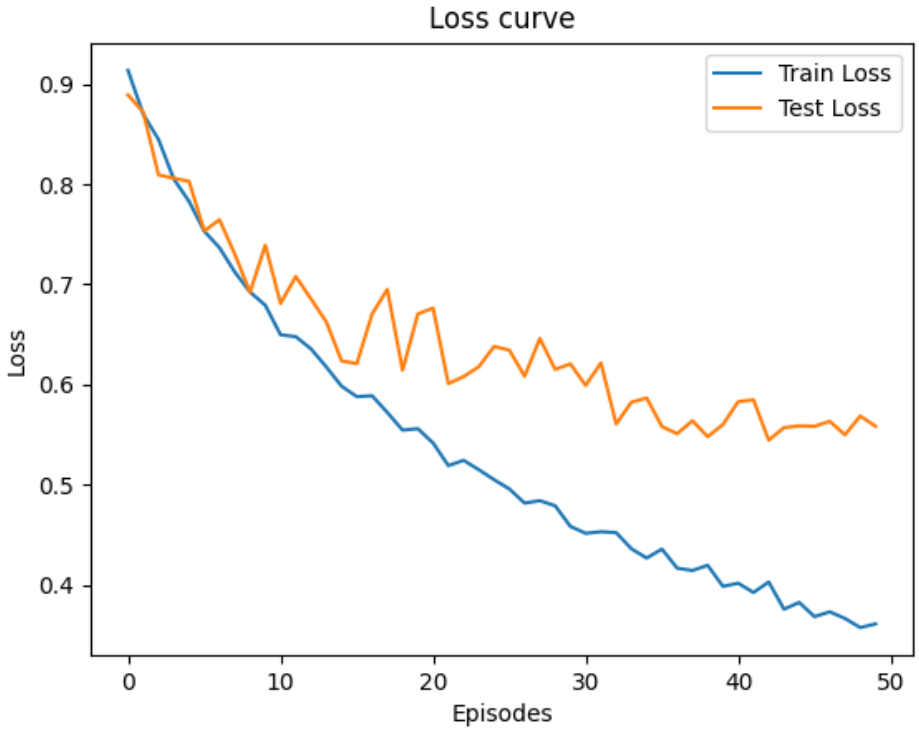
Small model trains on the sixteenth (None initialized weights):



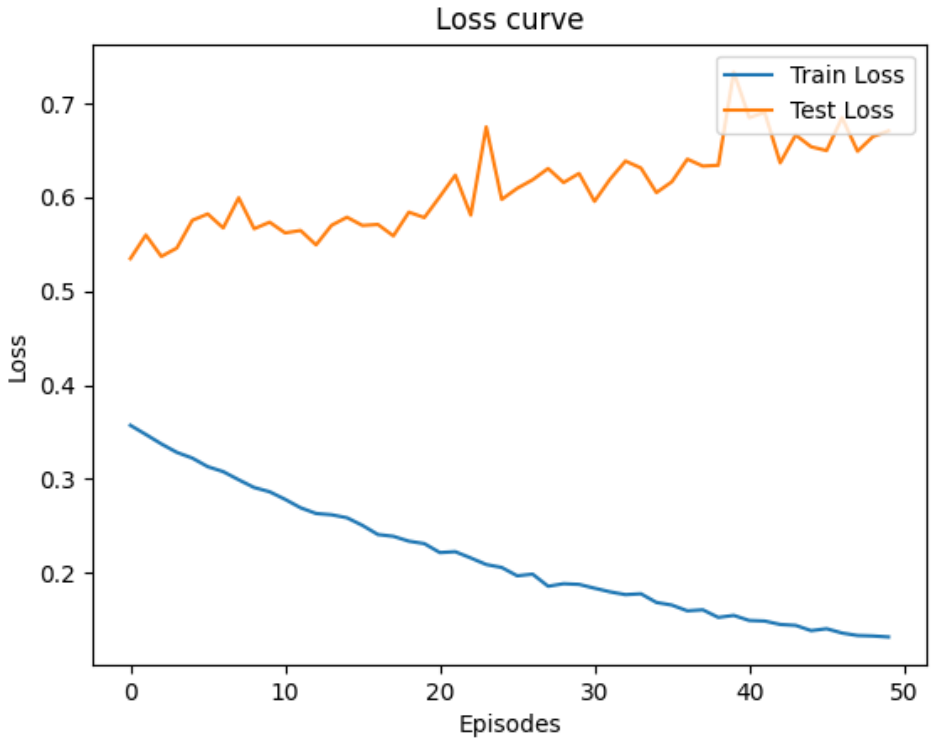


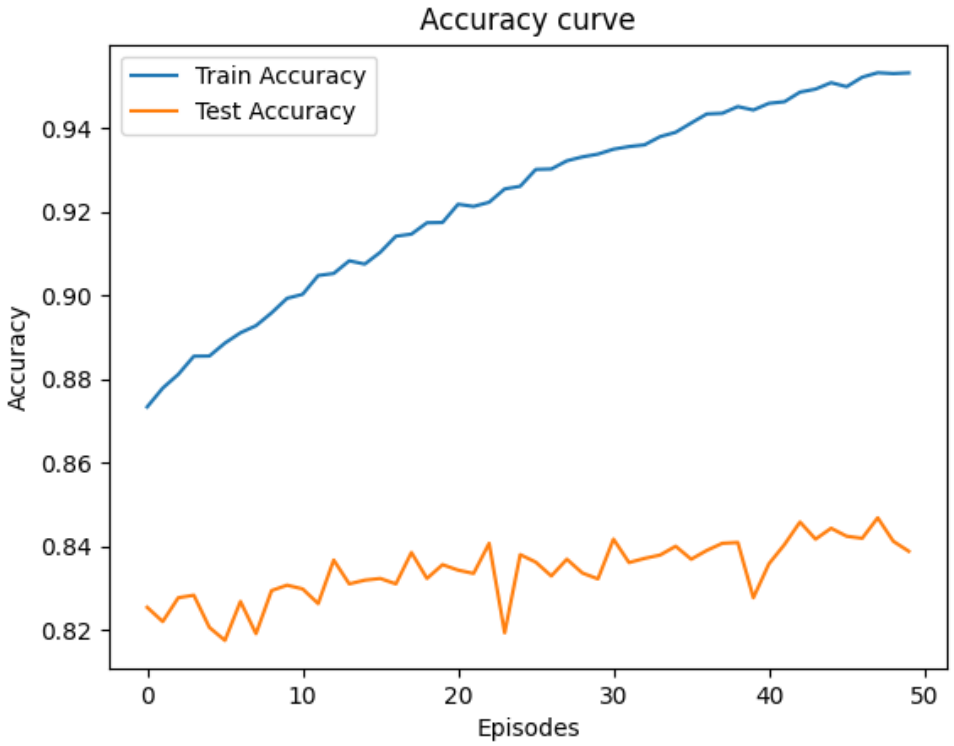
Small model trains on the half (None initialized weights):



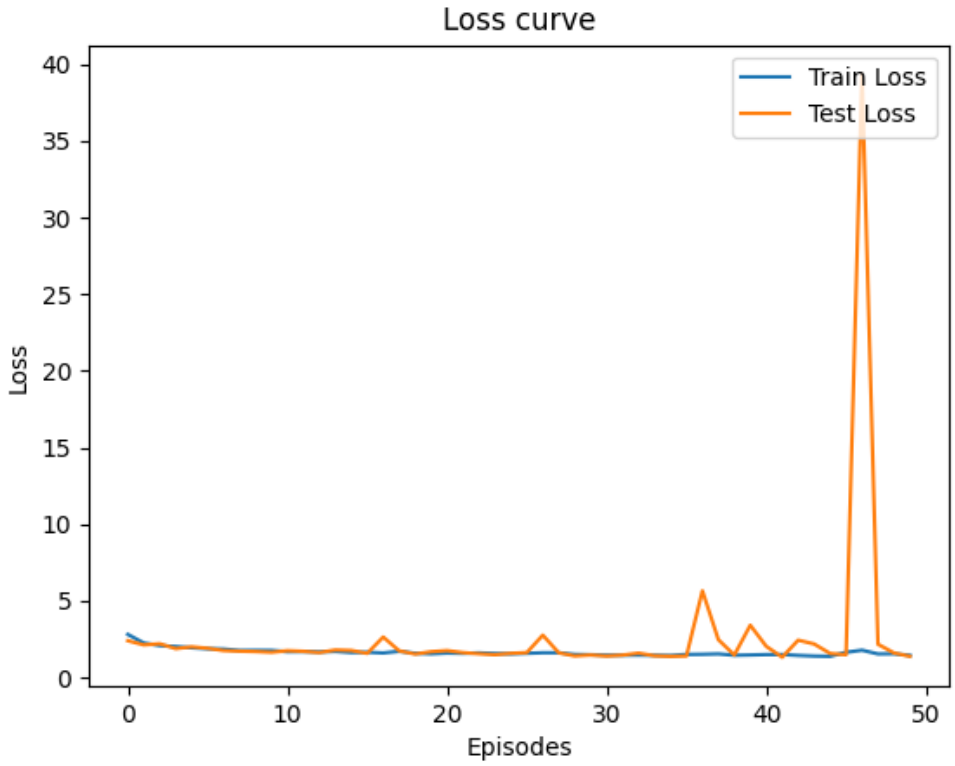


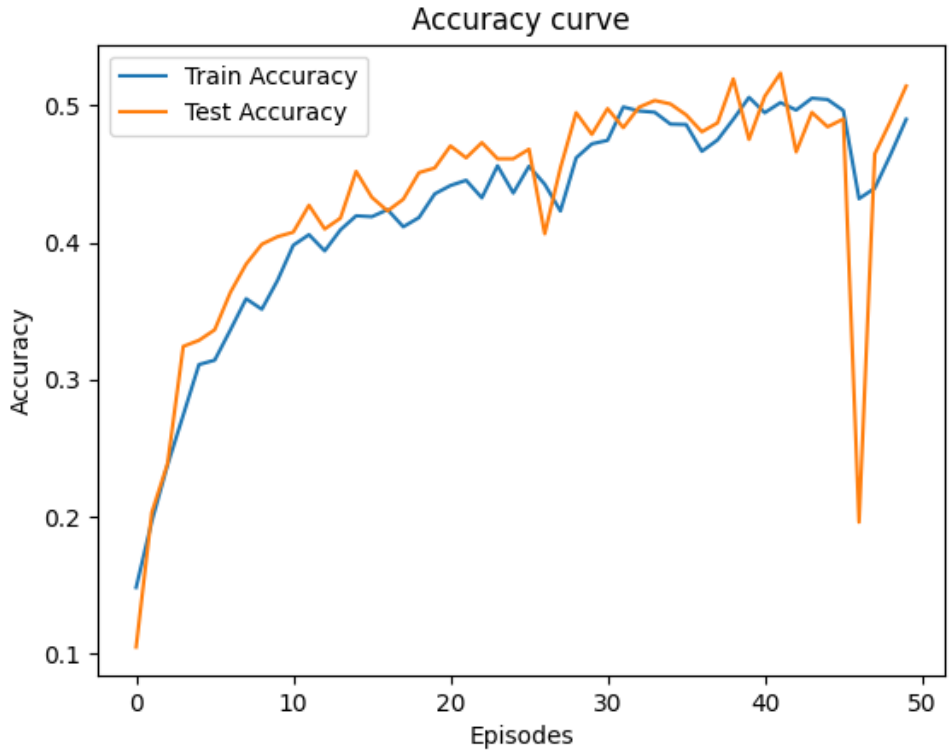
Small model trains on the all data (None initialized weights):



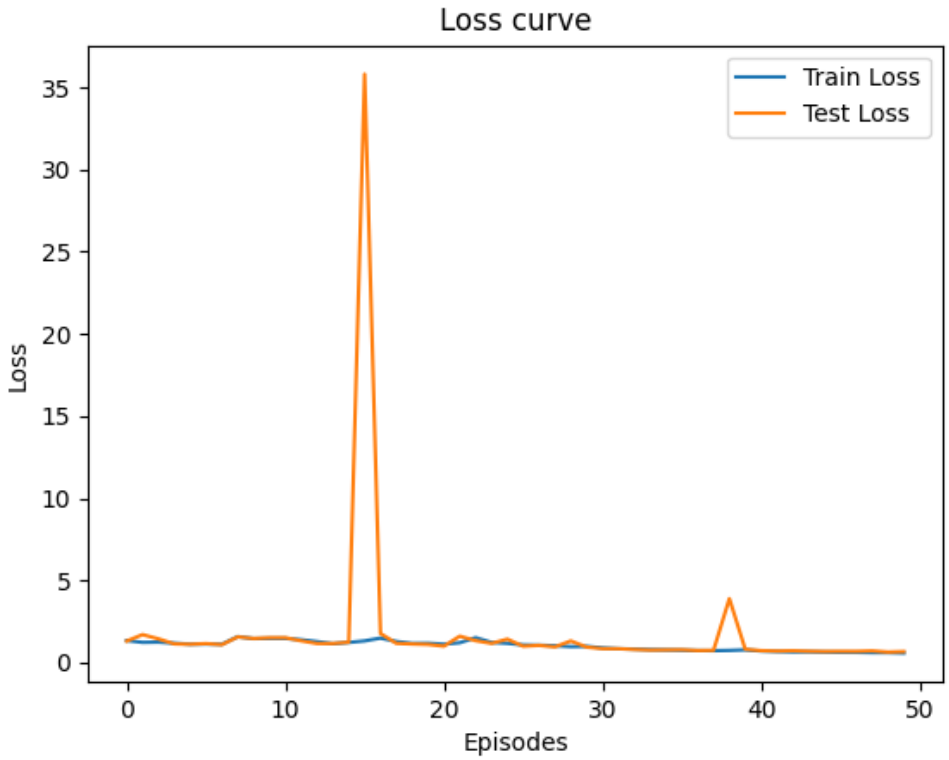


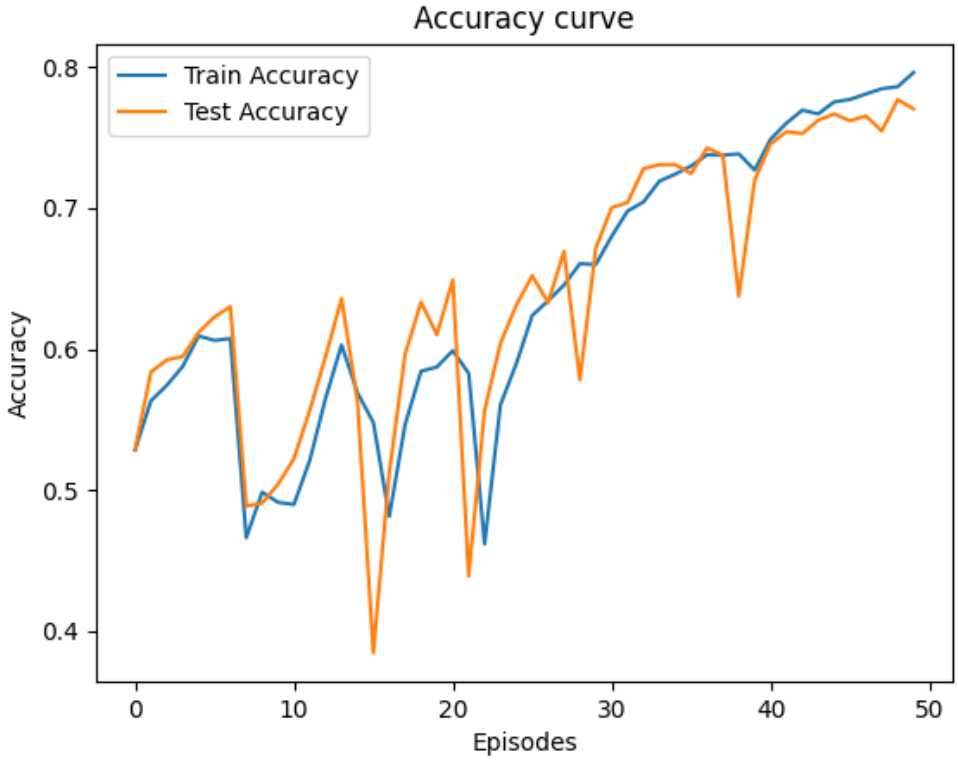
Big model trains on the sixteenth (None initialized weights):



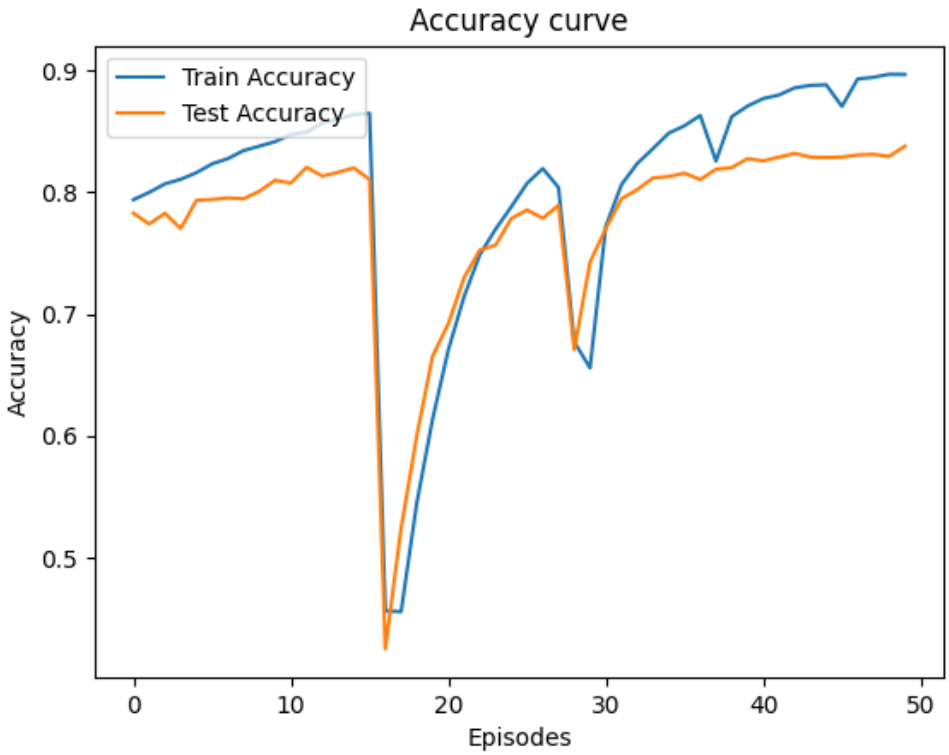


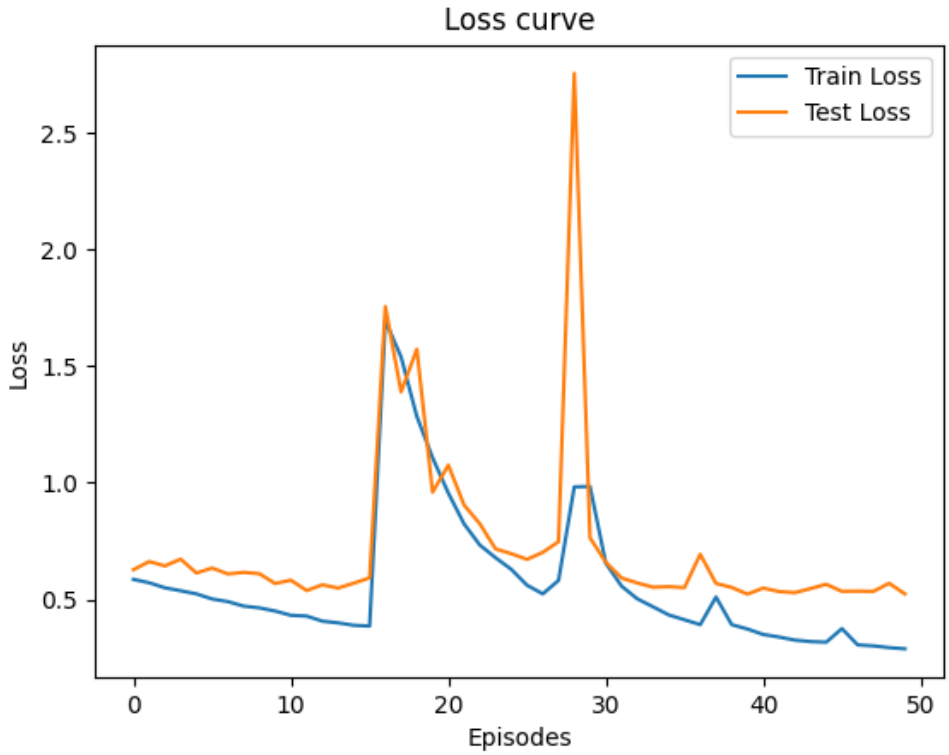
Big model trains on the half (None initialized weights):



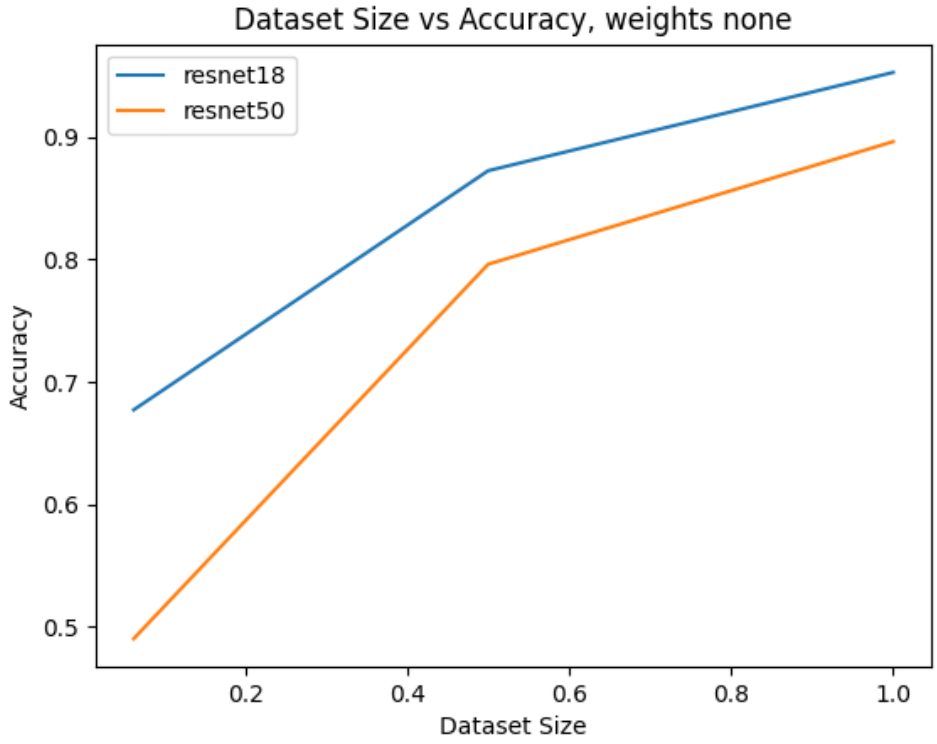


Big model trains on the all data (None initialized weights):





Dataset Size vs Accuracy:



我們可以從 Dataset Size vs Accuracy 圖中觀察到，小模型比大模型正確率還高，與 Fig1 圖結果不一樣，我分析原因可能是，大模型具有更多的參數和複雜性，容易在數據量較少時出現過擬合，因此在訓練數據上表現得非常好，但在未見過的數據上表現不佳。而小模型由於複雜度較低，更不容易過度擬合。

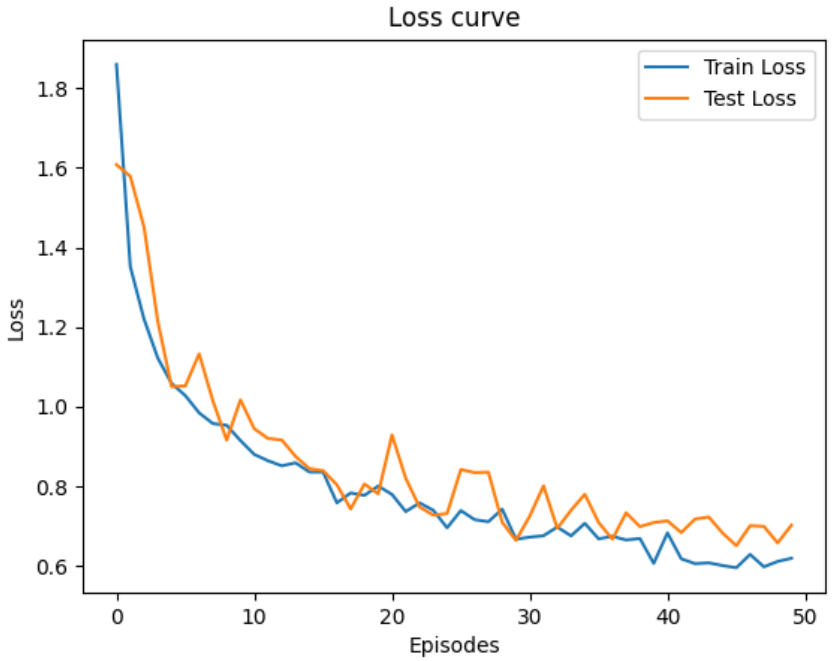
另外可能也有初始化差異，因為訓練是隨機初始化的權重通常會為模型提供不同的起始點。對於大型模型，具有更多參數和複雜性，這種不同的起始點可能導致較長的訓練時間和更容易陷入局部極小值。相反，小型模型通常具有較少的參數，因此可能較容易找到較好的權重初始化。

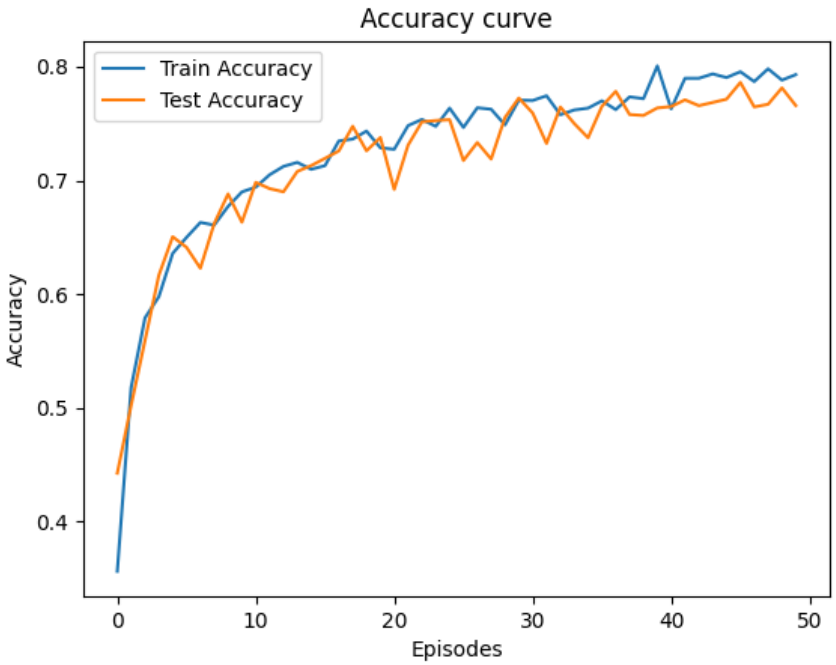
另外從觀察 Big model 和 Small model 在不同 size data 訓練，可以看到在小 size data 訓練不容易出現 overfitting 的情況，而在越多 data 的訓練中越容易出現 overfitting。

(10%) What if we train the ResNet with ImageNet initialized weights (weights="IMAGENET1K_V1"). Please explain why the relationship changed this way?

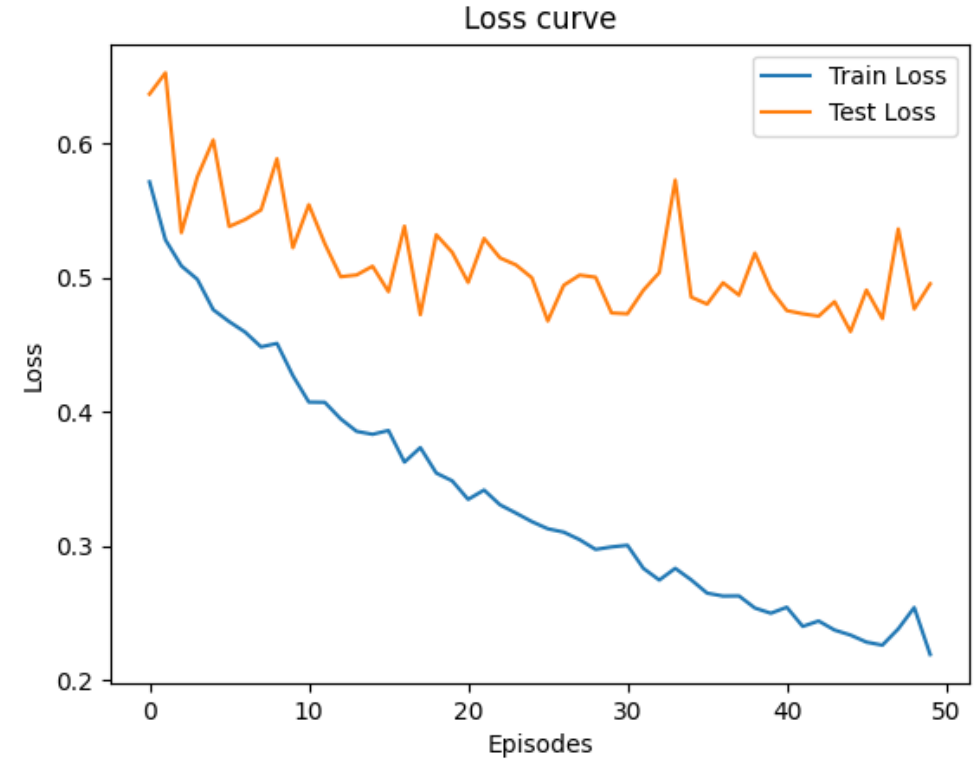
Ans:

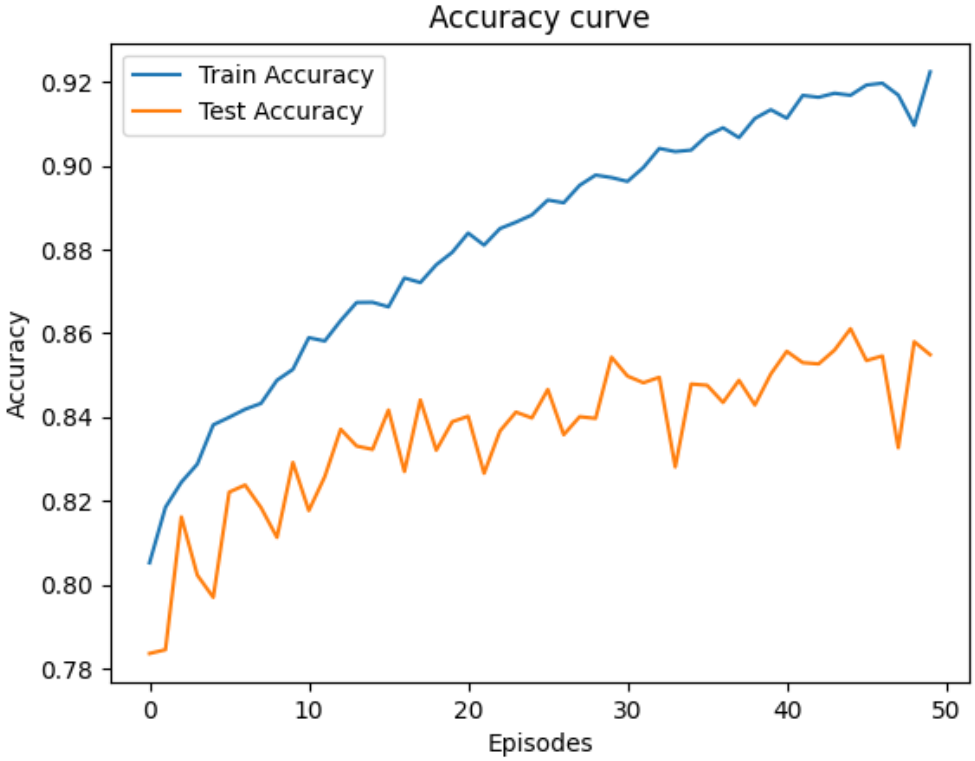
Small model trains on the sixteenth (initialized weights):



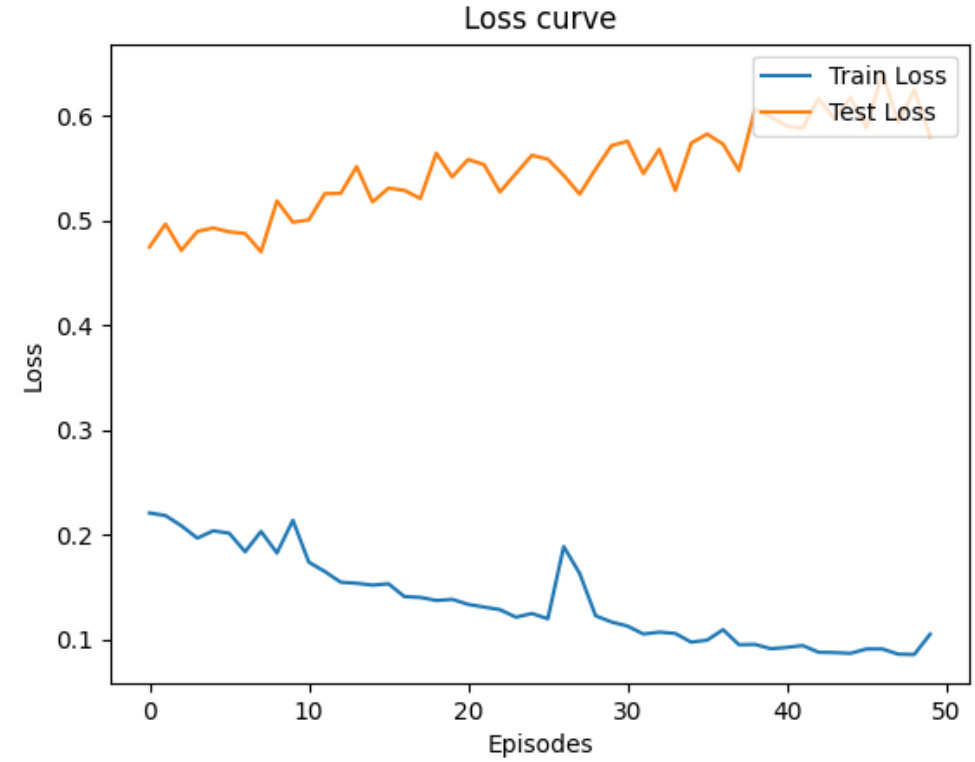


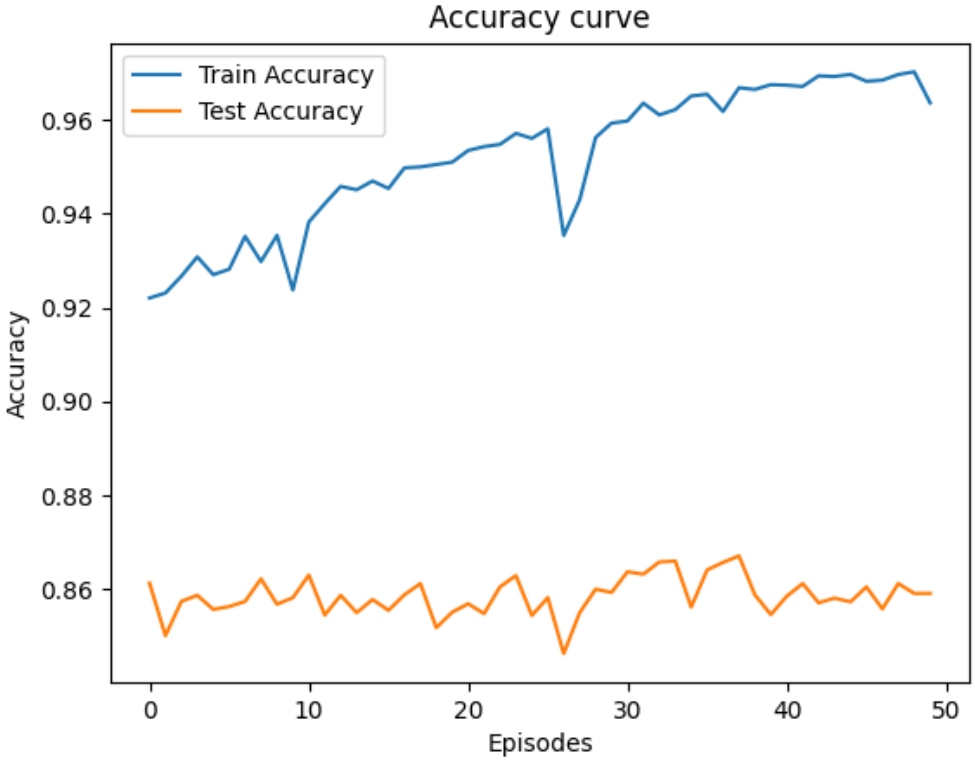
Small model trains on the half (initialized weights):



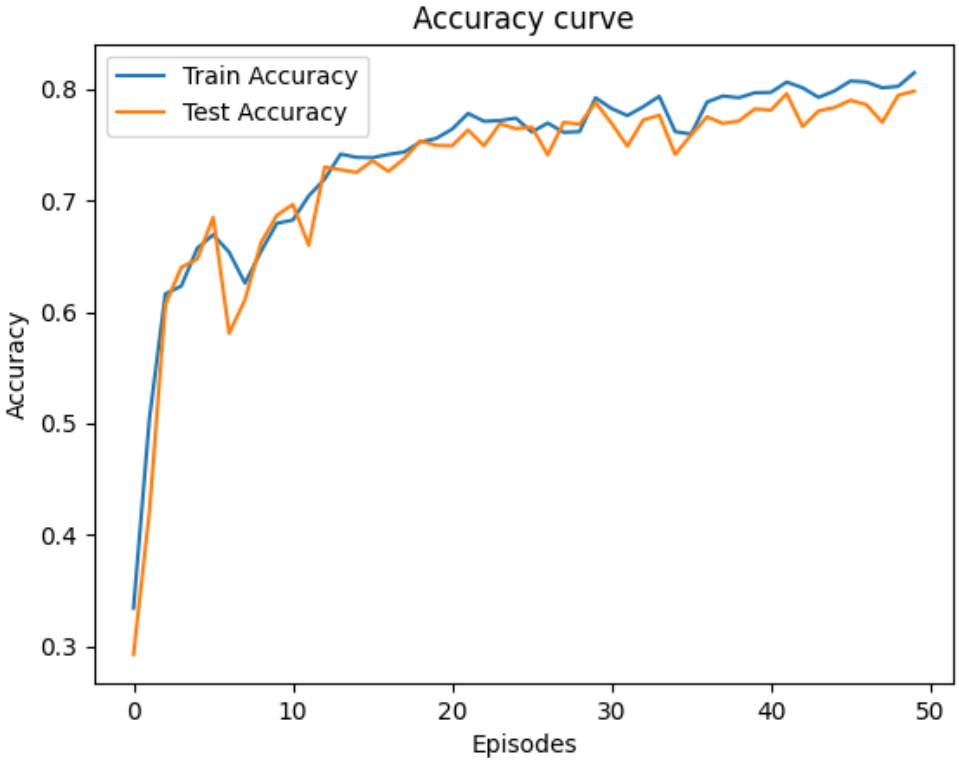


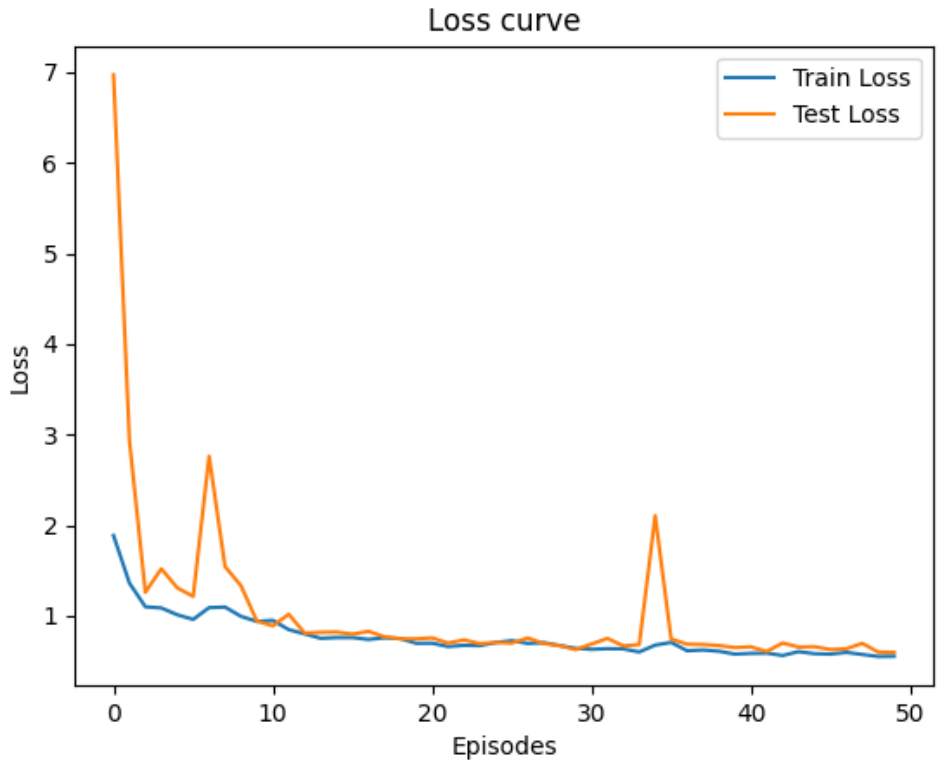
Small model trains on the all data (initialized weights):



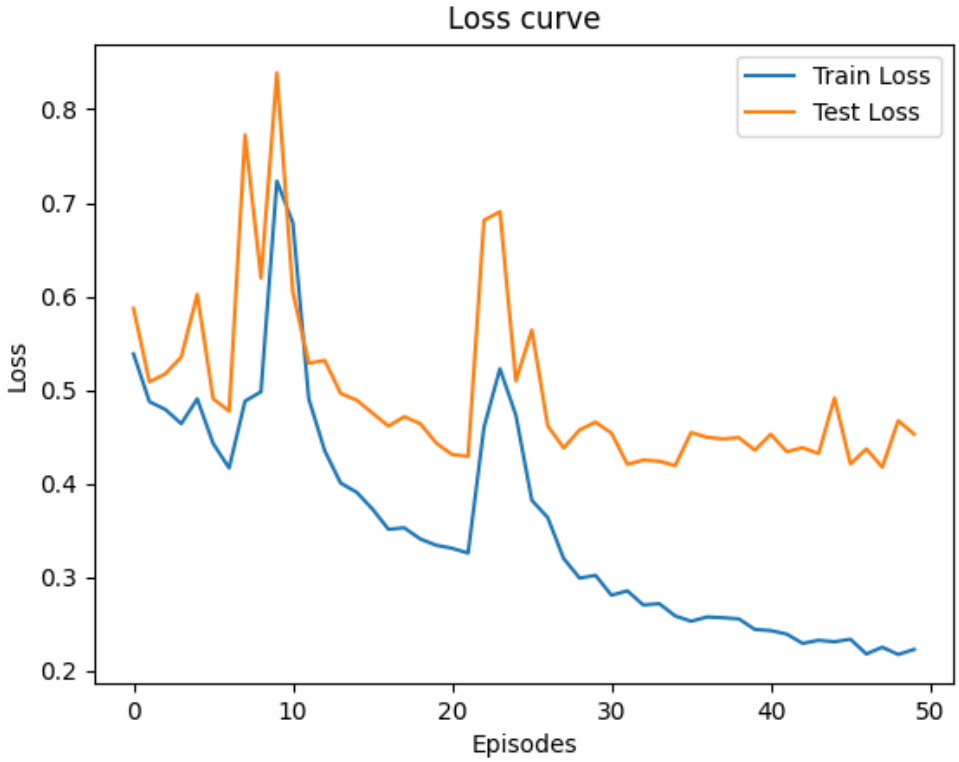


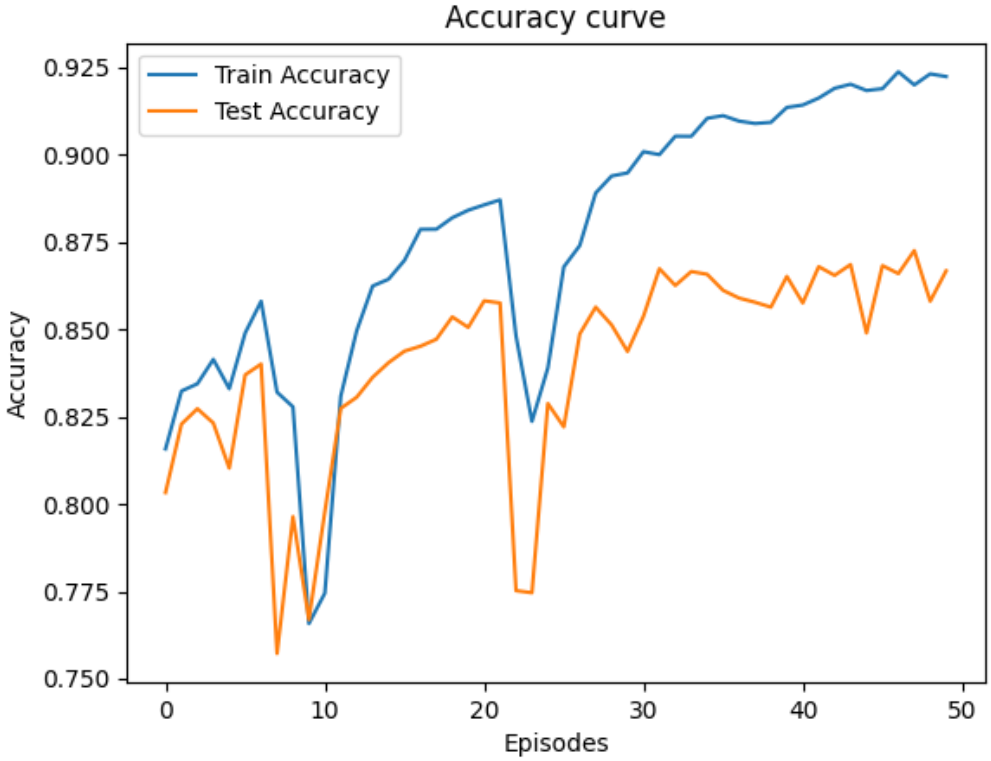
Big model trains on the sixteenth (initialized weights):



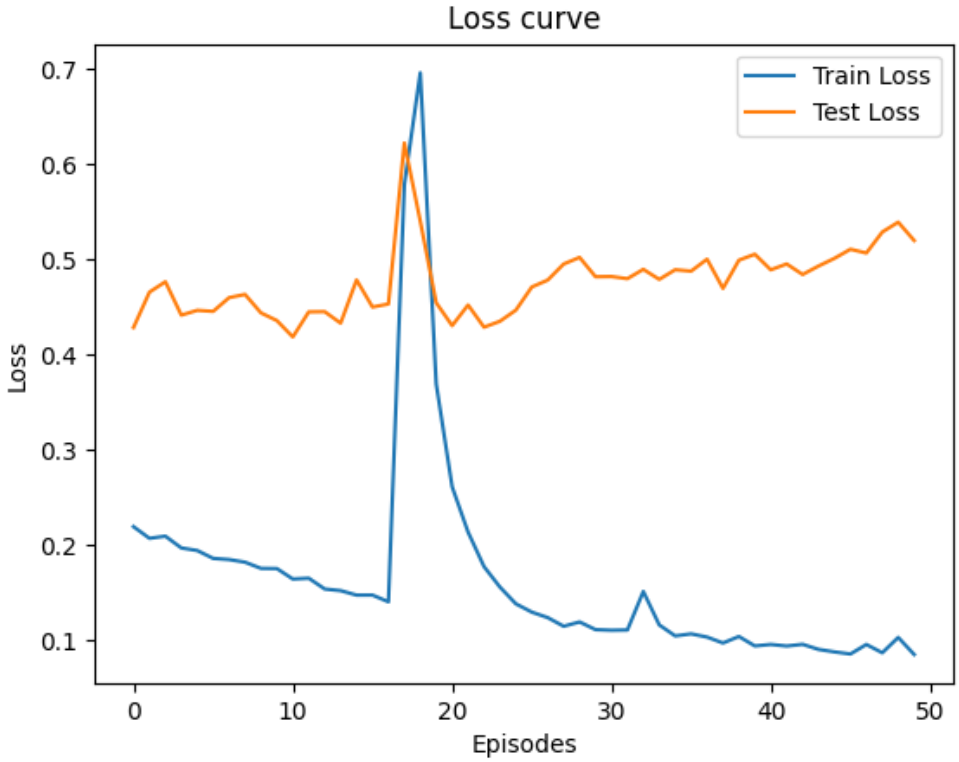


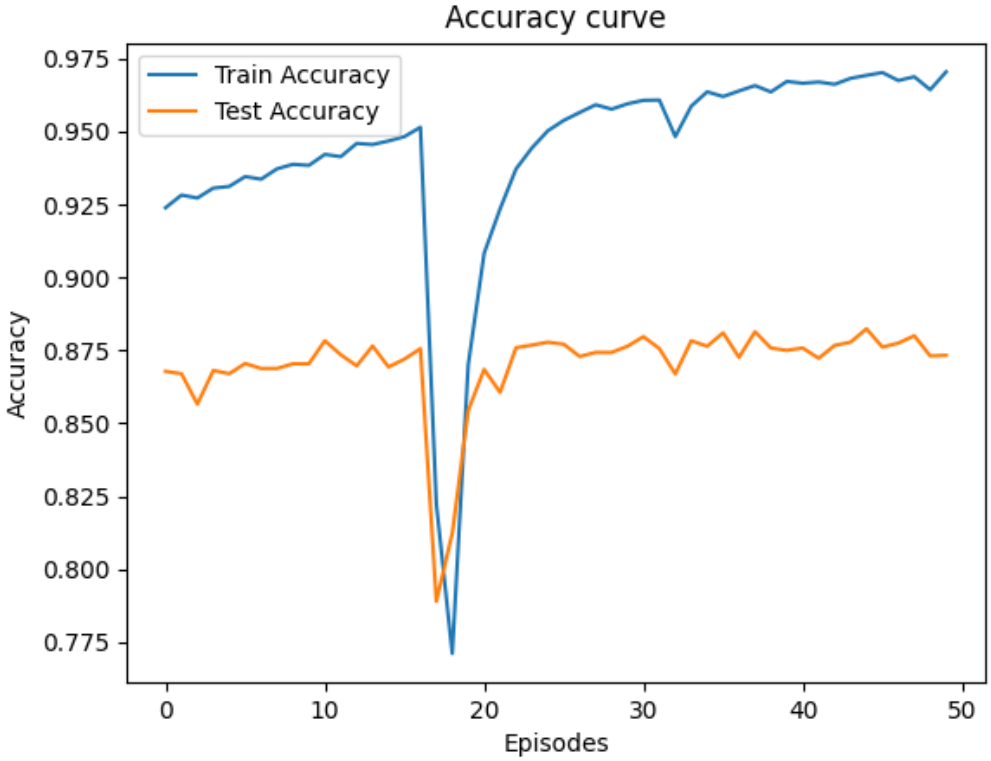
Big model trains on the half (initialized weights):



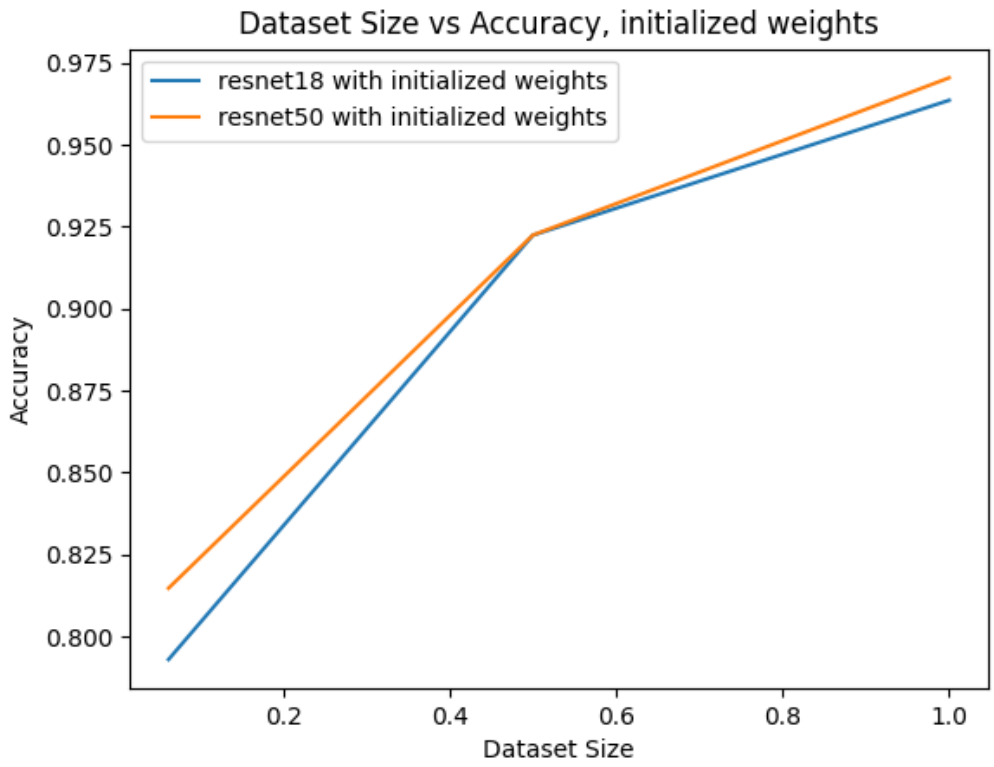


Big model trains on the all data (initialized weights):





Dataset Size vs Accuracy:



可以從 Dataset Size vs Accuracy 圖中觀察到，大模型與小模型相比，正確率都較高，因為使用了 ResNet with ImageNet initialized weights (weights="IMAGENET1K_V1")，使用 ImageNet 預訓練的權重可能有助於大型模型更好地捕捉數據集中的複雜特徵，這些權重可以包含對圖像中通用特徵的知識，並且可以通過微調或轉移學習來幫助模型更快地收斂到新的數據集上。
另外因為它們具有更豐富的模型容量和表徵能力，並且能夠更好地利用預訓練權重中的通用特徵知識。
從觀察 Big model 和 Small model 在不同 size data 訓練，可以看到在小 size data 訓練不容易出現 overfitting 的情況，而在越多 data 的訓練中越容易出現 overfitting。

Problems

3. (30%) Achieve the best performance given all training data using whatever model and training strategy. (You cannot use the model that was pretrained on CIFAR10)

Ans:

```
class Net(nn.Module):
  def __init__(self):
    super(Net, self).__init__()
    self.conv1 = nn.Conv2d(3, 128, 5, padding=2)
    self.conv2 = nn.Conv2d(128, 128, 5, padding=2)
    self.conv3 = nn.Conv2d(128, 256, 3, padding=1)
    self.conv4 = nn.Conv2d(256, 256, 3, padding=1)
    self.pool = nn.MaxPool2d(2, 2)
    self.bn_conv1 = nn.BatchNorm2d(128)
    self.bn_conv2 = nn.BatchNorm2d(128)
    self.bn_conv3 = nn.BatchNorm2d(256)
    self.bn_conv4 = nn.BatchNorm2d(256)
    self.bn_dense1 = nn.BatchNorm1d(1024)
    self.bn_dense2 = nn.BatchNorm1d(512)
    self.dropout_conv = nn.Dropout2d(p=0.25)
    self.dropout = nn.Dropout(p=0.5)
    self.fc1 = nn.Linear(256 * 8 * 8, 1024)
    self.fc2 = nn.Linear(1024, 512)
    self.fc3 = nn.Linear(512, 10)

  def conv_layers(self, x):
    out = F.relu(self.bn_conv1(self.conv1(x)))
    out = F.relu(self.bn_conv2(self.conv2(out)))
    out = self.pool(out)
    out = self.dropout_conv(out)
    out = F.relu(self.bn_conv3(self.conv3(out)))
    out = F.relu(self.bn_conv4(self.conv4(out)))
    out = self.pool(out)
    out = self.dropout_conv(out)
    return out

  def dense_layers(self, x):
```

```
        out = F.relu(self.bn_dense1(self.fc1(x)))
        out = self.dropout(out)
        out = F.relu(self.bn_dense2(self.fc2(out)))
        out = self.dropout(out)
        out = self.fc3(out)
        return out

    def forward(self, x):
        out = self.conv_layers(x)
        out = out.view(-1, 256 * 8 * 8)
        out = self.dense_layers(out)
        return out
```

我定義了自己的網路，初始化所有的網絡層。該模型包括兩個部分：

- **卷積層(conv1, conv2, conv3, conv4）和全連接層（fc1, fc2, fc3）。**
- **卷積層使用 nn.conv2d 來定義，並指定了輸入通道數、輸出通道數、卷積核大小和填充。**
- **最大池化層使用 nn.MaxPool2d，其目的是減小特徵圖的空間尺寸。**
- **正規化層（nn.BatchNorm2d 和 nn.BatchNorm1d）用於批次正規化，有助於穩定訓練過程。**
- **丟棄層（nn.Dropout 和 nn.Dropout2d）用於減小過擬合風險。**

定義了卷積層 conv_layers:

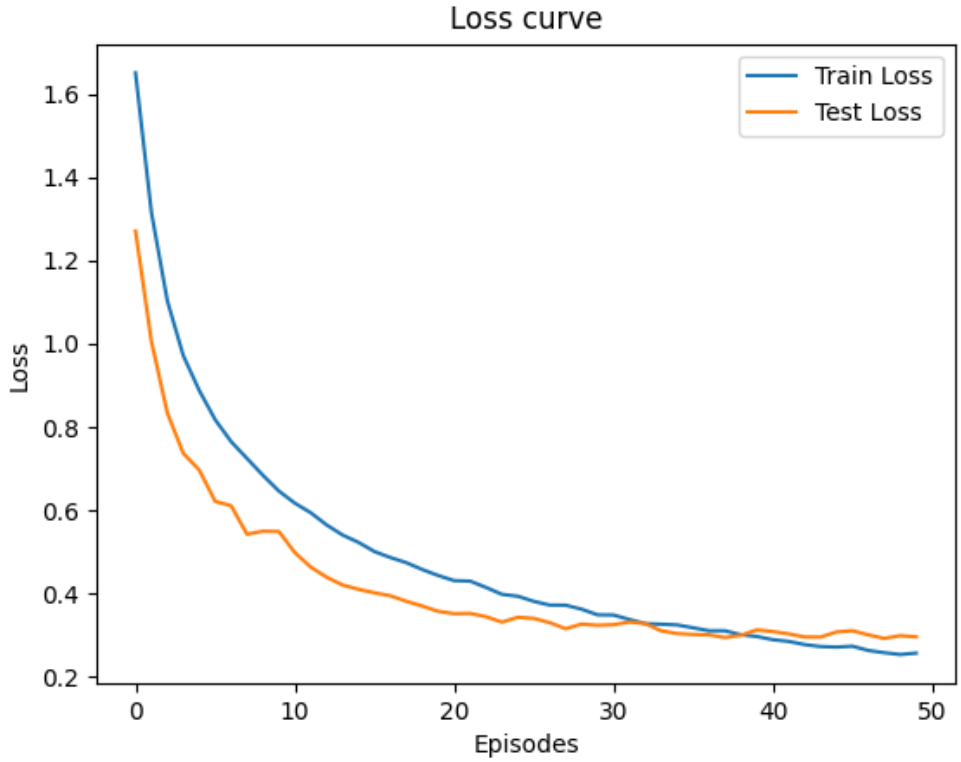
- **這個函數定義了模型的卷積部分。它接收輸入 x，通過一系列的卷積、批次正規化、激活和池化操作來提取特徵。**
- **每個卷積層後面都有 ReLU 激活函數，用於引入非線性。**
- **在每個卷積層之後，都有批次正規化操作，以穩定訓練過程。**
- **在某些層之後，使用丟棄層以減小過擬合風險。**
- **這個函數返回提取的特徵。**

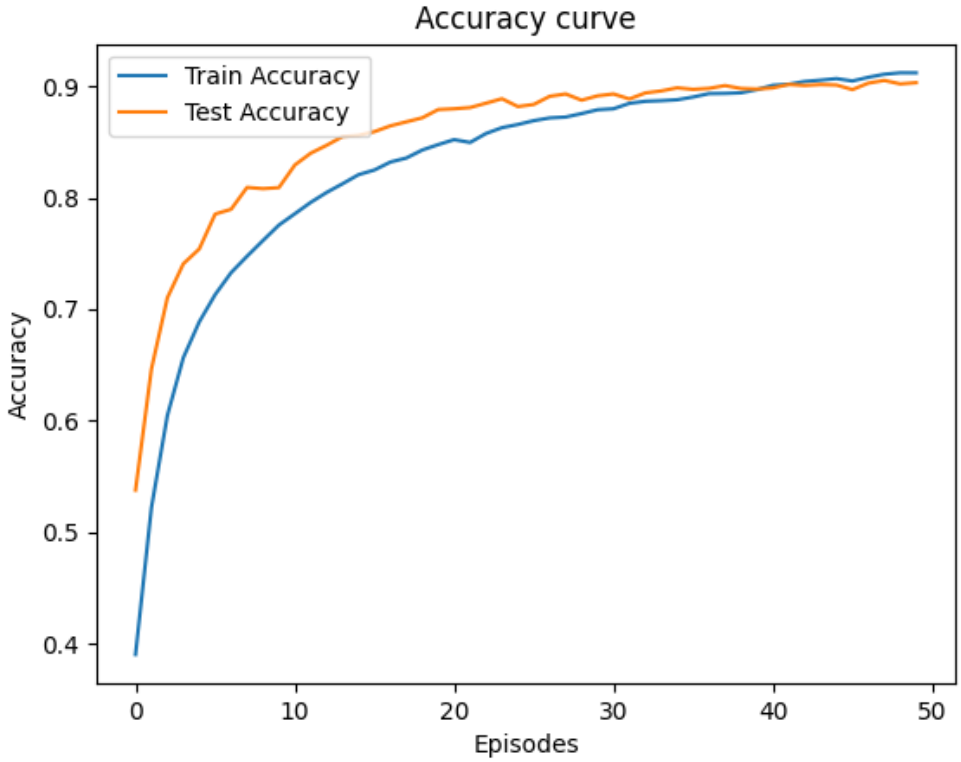
定義了卷積層 dense_layers:

- **這個函數定義了模型的全連接部分。它接收卷積部分提取的特徵 x，並通過一系列的全連接層、批次正規化、激活和丟棄操作來執行分類。**
- **每個全連接層後面都有 ReLU 激活函數。**
- **在某些層之後，使用丟棄層以減小過擬合風險。**
- **最終的全連接層 fc3 的輸出大小為 10，用於分類 10 類的圖像數據。**

定義 forward 函數:

這個函數定義了模型的正向傳播過程。它首先通過卷積部分 **conv_layers** 提取特徵，然後通過全連接部分 **dense_layers** 執行分類，最終返回模型的輸出。





可以觀察到這麼網路模型在大數據訓練下，並不會與前面的 ResNet18 和 ResNet50 一樣會出現 overfitting 情況，並且在測試及上的正確率可以到達 91%。