DSP 2021 fall
Project

姓名: 許顥騰
學號: R10922100

Part 1

1. I used:
   PyTorch / Lightning
   CPU / **CUDA**
   Colab / **PC** / Other:_____

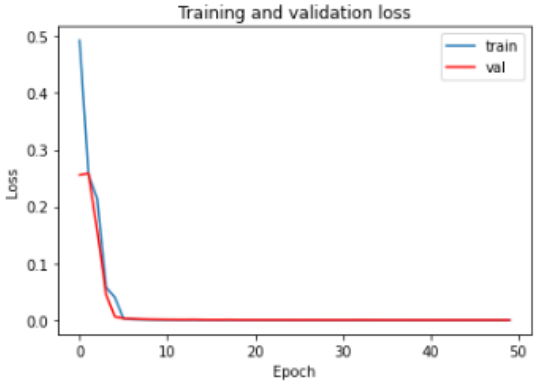
2. Kaggle result:

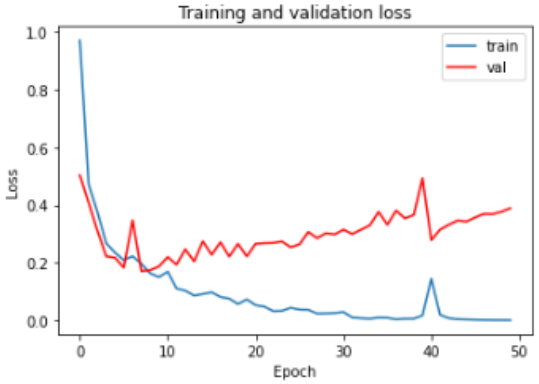
Accuracy	0.99666

3. Preprocessing details. (15pts)
   一開始先把 training data 給 shuffle，然後以 8:2 分成 training set 及 validation set。

   由於這份資料集沒那麼複雜，不用做太複雜的前處理就可以有很好的表現。在固定 model 跟參數下，我對兩個 channel 做兩種 feature scaling，分別是 maximum absolute scale 及 standardization。

   結果是 standardization(左圖)做出來的結果最好，training 跟 validation 的 loss 很快就降到趨近 0，反而用 maximum absolute scale(右圖)的 loss 在大概第 8 個 epoch 時 validation loss 就開始上升，看起來是發生 overfitting。





4. Method, training details (model arch., why did you select this arch?), experimental setting (hyper-parameters) (30pts):
   在上課中看到助教用 4 層 1d convolution 就能做到 100%正確率，所以我的嘗試方向是從 4 層開始簡化 model 複雜度，否則繼續增加層數只會更容易 overfitting。

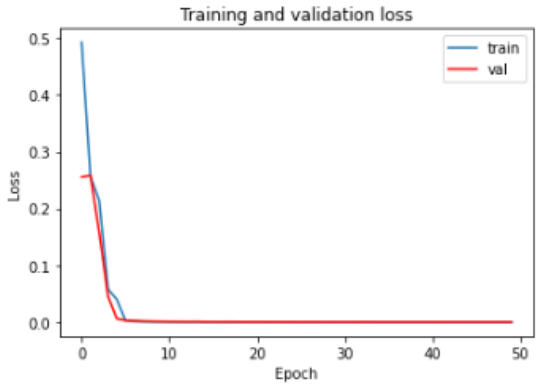
   這個問題的資料集是讓軸承旋轉然後用加速規和速度編碼器取樣，我覺得取樣出來的點可能會根據軸承轉速跟取樣頻率呈現一種類似週期訊號，所以我用較大的 kernel 跟 stride，也許可以抓出週期性的 pattern。

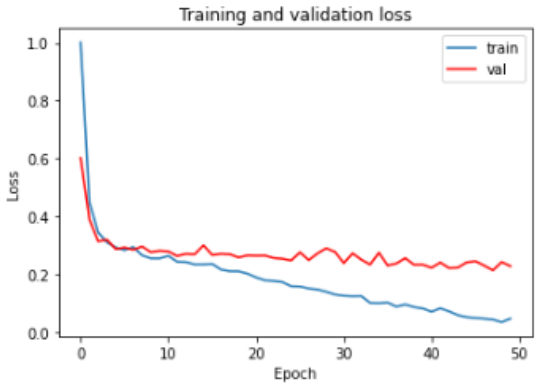
   最後的 model 架構如下圖，optimizer 使用 Adam，batch size = 32，learning rate = 0.005，總共 50 個 epoch，並記錄當前最低的 validation loss，只要有進步就保存該 epoch 的 model。

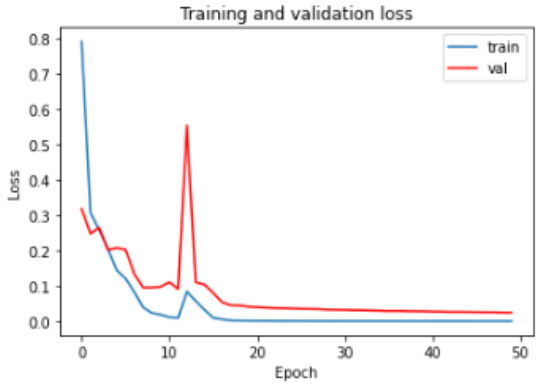
```python
class Network(nn.Module):
    def __init__(self) -> None:
        super(Network, self).__init__()

        self.conv = nn.Sequential(
            nn.Conv1d(2, 4, 64, 32),
            nn.ReLU(),
            nn.Conv1d(4, 8, 64, 32),
            nn.ReLU()
        )

        self.linear = nn.Sequential(
            nn.Linear(112, 3)
        )

    def forward(self, x):
        x = self.conv(x)
        x = x.view(x.size(0), -1)
        x = self.linear(x)
        return x

    def loss(self, pred, target):
        criterion = nn.CrossEntropyLoss()
        loss = criterion(pred, target)
        return loss
```

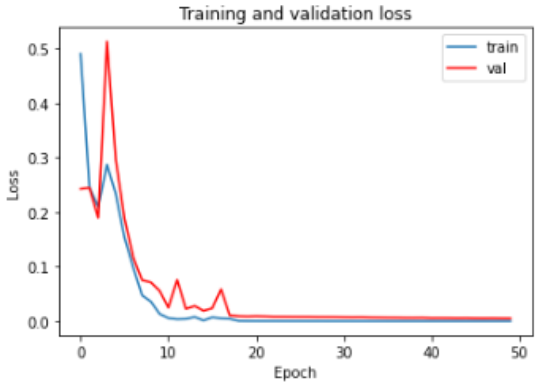
這個 model 在 public score 只得到 99.66%，所以我有嘗試做一些 layer 的微調。
(1) 試不同的 activation layer：Sigmoid、Tanh、LeakyReLU。
(2) 在 Conv1d 和 ReLU 間加上 BatchNormalization。

換不同 activation layer 的 loss，ReLU(左上)、Sigmoid(右上)、Tanh(左下)、LeakyReLU(右下)

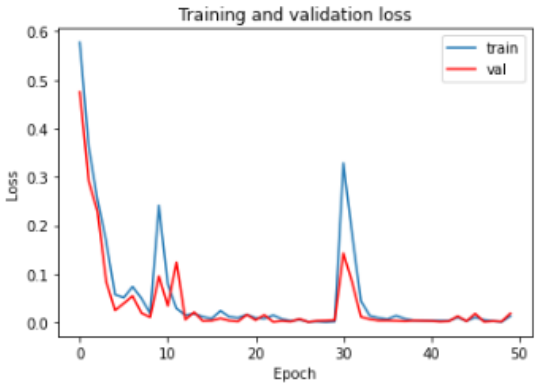








加上 BatchNormalization layer



這幾個不同的嘗試下來，都沒有進步反而都退步了。

5. What have you learned (Interesting Findings and Special Techniques)? (30pts)

這次作業是我第一次認真用 Pytorch 寫完一整個訓練的流程，以前都是寫好玩的，所以一開始寫出幾個 bug，以為 train 出來的表現不錯，結果上傳 kaggle 後不到 simple baseline。

一個 bug 是做 feature scaling 的時候應該要用 training set 學到的 scaling 用在 testing set 上，我一開始是用 testing set 來做 scaling。另一個是 validation loss 算錯，所以訓練出來的 model 是有問題的。

另外一個有趣的發現是在 model 的建構子中如果有宣告多的 layer 沒有用在 forward 中的話會對訓練產生些微的差異。這是我在寫 part 2 中遇到的，建構子中放了幾個沒有的 layer 便一直無法還原實驗結果，最後查了一下可能遇到的問題是建構子中放了沒有使用到的 layer 跟不同 layer 宣告的順序會影響每個 layer 的 weight initialization 的結果。

Part 2

1. Explain your method and result in detail. (7pts)

Accuracy	0.82

我使用 one class svm 做 anomaly detection，但 svm 需要二維資料做訓練，所以我的想法是用 task 1 中的 1d convolution 把每一個 sample 的兩個 channel 變成一個 channel。

做法是訓練一層 1d convolution layer 和一層 fully connected layer 的 model，並取出最後訓練完的 convolution kernel 來套用在每個 sample 上。由於 task 1 的 model 表現不錯，所以我仿造整個訓練流程。

把訓練出來的 kernel 套用在 data set 上後就可以訓練 one class svm。拿 training set 做訓練，anomaly set 用來評估 svm 有沒有分類好，以此來調整 svm 參數。最後訓練完便可對 testing set 做分類，分類出是不是 anomaly。

由於還須對 non-anomaly data 做分類，我直接使用 task 1 訓練好的 model 做分類，如果這個 sample 被 one class svm 判斷成 anomaly data，則直接給他 label 3。

整個流程大致是：

(1) Shuffle training data，照 8:2 分成 training set 跟 validation set。
(2) 對兩個 channel 做 standardization，由 training set 學到的 mean 跟 variance 套用在其他 set 上。
(3) 用 training set 訓練只有一層 1d convolution 的 model，並用 validation set 評估表現。(optimizer 使用 Adam，batch size = 32，learning rate = 0.005，總共 50 個 epoch)
(4) 取出訓練好的 1d convolution layer 的 kernel 並套用在 training、anomaly、testing set 上，得到一組二維的資料。

(5) 用經過 kernel 轉換過後的 training set 來訓練 one class svm，訓練完後用經過 kernel 轉換的 anomaly data 評估表現，然後調整參數，最後 predict testing data。理想情況是 training data 都被判斷成 non-anomaly，anomaly data 都被判斷成 anomaly。

(6) Load 在 task 1 訓練好的 model，predict 被 one class svm 判斷為 non-anomaly 的 testing data，anomaly data 則直接 label 成 3。

```python
class Network(nn.Module):
    def __init__(self, task1=True) -> None:
        super(Network, self).__init__()

        if task1:
            self.conv = nn.Sequential(
                nn.Conv1d(2, 4, 64, 32),
                nn.ReLU(),
                nn.Conv1d(4, 8, 64, 32),
                nn.ReLU()
            )
            self.linear = nn.Sequential(
                nn.Linear(112, 3)
            )
        else:
            self.conv = nn.Sequential(
                nn.Conv1d(2, 1, 64, 32),
                nn.ReLU()
            )
            self.linear = nn.Sequential(
                nn.Linear(499, 3)
            )

    def forward(self, x):
        x = self.conv(x)
        x = x.view(x.size(0), -1)
        x = self.linear(x)
        return x

    def loss(self, pred, target):
        criterion = nn.CrossEntropyLoss()
        loss = criterion(pred, target)
        return loss
```

這次實驗的結果是意外產生的，在流程中的第 3 步中決定使用的 model 時，原本是打算用最低的 validation loss 的 model，在右上圖第 26 個 epoch，但一開始在測試這方法是否可行的時候不小心用到最後一個 epoch 的 model，在調整 one class svm 的參數後結果 public score 得到 0.82。之後修改使用最低 validation loss 的 model 最好的結果只到 0.8。