DSP 2021 fall
Project

姓名: 黃奕誠
學號: R10922136

Part 1

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

2. Kaggle result:

Accuracy	1.000

3. Preprocessing details. (15pts)
   以助教提供在 sample code 中的 norm 為基礎，我作了以下三種嘗試:
   1, 完全不做 normalization，直接以 raw data 作為訓練資料
   2, 使用 sklearn 的 StandardScaler
   3, 使用 sklearn 的 StandardScaler，同時也做 data augmentation，將原有的 data 加上微量的 noise
   由實驗結果來看，(2)的表現是最好的。推測是因為有經過 normalization 的 data 在神經網路訓練的過程更加穩定。而(3)則多了幾個超參數需要調整，另外，這個 task 以結果而言，training data 的數量也相當充足，因此效益不佳。

4. Method, training details (model arch., why did you select this arch?), experimental setting (hyper-parameters) (30pts):

   模型的定義如下圖。主要由四層 Conv1D 與一層 FC Layer 組成，中間加入了三層 ReLU 作為 activation function。此架構參考自助教於課程中的示範，預測的表現也相當優異。
   其他設定如下:
   Batch size: 48
   Epoch: 30
   Optimizer: Adam
   Learning rate: 3e-3

```python
class MyDSPNet(pl.LightningModule):
    def __init__(self):
        super().__init__()
        # define your arch
        self.encoder = nn.Sequential(
            nn.Conv1d(2, 20, kernel_size=13, stride=7), #2285 * 20
            nn.ReLU(),
            nn.Conv1d(20, 40, kernel_size=11, stride=7), #326 * 40
            nn.ReLU(),
            nn.Conv1d(40, 80, kernel_size=9, stride=5), #64 * 80
            nn.ReLU(),
            nn.Conv1d(80, 160, kernel_size=7, stride=5), #12 * 160
        )
        self.clf = nn.Linear(1920, 3)
    def forward(self, x):
        # define your forward
        x = self.encoder(x)
        x = x.view(x.size(0), -1)
        output = self.clf(x)
        return output
    def configure_optimizers(self):
        optimizer = torch.optim.Adam(self.parameters(), lr=3e-3)
        return optimizer
    def training_step(self, train_batch, batch_idx):
        x, y = train_batch
        output = self(x)
        loss = F.cross_entropy(output, y)
        self.log('train_loss', loss)
        return loss
```

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

1, 資料的前處理與訓練超參數的重要性不亞於架構本身。本 task 是資料量較小且分類相對容易的問題，而即便如此，前處理的有無、處理方式的不同也能明確的在預測準確率上體現出差異。超參數中特別是 learning rate 亦會大幅影響預測準確率，他與 epoch 也需一起調整以避免未收斂或者 overfit 的問題。

2, 訓練資料充分的神經網路模型的能力比我預期的更加強大(主要是與 task2 相比)。Task2 的部分我另外嘗試了 Autoencoder, VAE, PCA, one class SVM 等方式以試圖找出 anomaly，但這些做法始終難以超越直接將少數 anomaly 加入 training data 訓練起來的神經網路。

Part 2

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

Accuracy	0.54666

2. Preprocessing details.

   (a)Task1 的(2)

   (b)以 Task1 的(2)為基礎，另外生成了 500 筆的隨機資料，並將他們的 label 設定為”3”

   以結果而言，(a)搭配上 AutoEncoder 有最佳的表現，但(b)搭配 one class SVM 的表現與其相差不遠。推測是因為資料隱含的 pattern 難以利用 Normal distribution 簡單 sample 出的資料替代，但這些隨機生成的資料仍有幫助 one class SVM 分類的功能。

3. Method, training details (model arch., why did you select this arch?), experimental setting (hyper-parameters).

   (A) one class SVM

   將(b)中隨機產生的資料加入原有的 training data 之中當作假的 anomaly data，並使用了 sklearn.svm.OneClassSVM，將 nu 設定為 0.35。此 Model 是用來檢測 data 是否為 anomaly，如是則 predict label 為 3，如否則將資料輸入 Task1 的分類 model，predict label 為[0 or 1 or 2]。

   (B) AutoEncoder

   資料前處理為(a)。建立一個 AutoEncoder 如下圖，目標是讓這個 AutoEncoder 能還原出與 input x 盡量相似的資料 x_hat，並且計算 mse_loss(x_hat, x)。如果 test data 算出的 mse_loss 超過 threshold 值，則其 predict label 為 3，如 mse_loss 小於 threshold 則將資料輸入 Task1 的分類 model，predict label 為[0 or 1 or 2]。

```python
class AutoEncoder(nn.Module):
    def __init__(self):
        super().__init__()
        self.encoder = nn.Sequential(
            nn.Conv1d(2, 20, kernel_size=13, stride=7), #2285 * 20
            nn.BatchNorm1d(20),
            nn.ReLU(),
            nn.Conv1d(20, 40, kernel_size=11, stride=7), #326 * 40
            nn.BatchNorm1d(40),
            nn.ReLU(),
            nn.Conv1d(40, 80, kernel_size=9, stride=5), #64 * 80
            nn.BatchNorm1d(80),
            nn.ReLU(),
            nn.Conv1d(80, 160, kernel_size=7, stride=5), #12 * 1
            #nn.ReLU(),
            #nn.Conv1d(160, 1, kernel_size=1, stride=1),
        )
        self.decoder = nn.Sequential(
            #nn.ConvTranspose1d(1, 160, kernel_size=1, stride=1, output_padding=0),
            #nn.ReLU(),
            nn.ConvTranspose1d(160, 80, kernel_size=7, stride=5, output_padding=2),
            nn.BatchNorm1d(80),
            nn.ReLU(), #64 * 80
            nn.ConvTranspose1d(80, 40, kernel_size=9, stride=5, output_padding=2),
            nn.BatchNorm1d(40),
            nn.ReLU(), #326 * 40
            nn.ConvTranspose1d(40, 20, kernel_size=11, stride=7, padding=1, output_padding=1),
            nn.BatchNorm1d(20),
            nn.ReLU(), #2285, 20
            nn.ConvTranspose1d(20, 2, kernel_size=13, stride=7, padding=1, output_padding=1),
            #16000, 2
        )

    def forward(self, x):
        x = x.view(-1, x.size(-2), x.size(-1))
        encoded = self.encoder(x)
        decoded = self.decoder(encoded)

        return decoded
```

4. What have you learned (Interesting Findings and Special Techniques)?
   Anomaly detection 是一個意外困難的題目。嘗試了 NN、PCA 皆效果不彰，one class SVM 似乎應該加上 feature transformation 才能讓他的分類更精準。AutoEncoder 的 threshold 相當難以制定，Anomaly data 跟 test data 的 mse_loss 的分布十分接近，可能需要多 submit 幾次才能找到較好的數值。另外我也嘗試使用 VAE，但或許是因為有東西寫錯，VAE 似乎訓練不起來。