說明：請各位使用此template進行Report撰寫，如果想要用其他排版模式也請註明題號以及題目內容（請勿擅自更改題號），最後上傳前，請務必轉成PDF檔，並且命名為report.pdf，否則將不予計分。

學號：R12945060 系級：生醫電資所碩二 姓名：羅佳蓉

1. (1.5%) AutoEncoder model

   a. (0.5%) 貼上private submission所使用的AutoEncoder model程式碼。

```
# Residual Block class
class ResidualBlock(nn.Module):
    def __init__(self, in_channels, out_channels):
        super(ResidualBlock, self).__init__()
        self.conv1 = nn.Conv2d(in_channels, out_channels, kernel_size=3, stride=1, padding=1)
        self.bn1 = nn.BatchNorm2d(out_channels)
        self.relu = nn.ReLU(inplace=True)
        self.conv2 = nn.Conv2d(out_channels, out_channels, kernel_size=3, stride=1, padding=1)
        self.bn2 = nn.BatchNorm2d(out_channels)
        self.skip = nn.Conv2d(in_channels, out_channels, kernel_size=1, stride=1, padding=0)
    def forward(self, x):
        identity = self.skip(x)
        out = self.conv1(x)
        out = self.bn1(out)
        out = self.relu(out)
        out = self.conv2(out)
        out = self.bn2(out)
        out += identity
        out = self.relu(out)
        return out
# Modified Autoencoder class
class Autoencoder(nn.Module):
    def __init__(self):
        super(Autoencoder, self).__init__()
        # Encoder
        self.enc1 = ResidualBlock(3, 64)
        self.enc2 = ResidualBlock(64, 128)
        self.enc3 = ResidualBlock(128, 256)
        self.enc4 = ResidualBlock(256, 512)
        self.pool = nn.MaxPool2d(2, 2)
        # Bottleneck
        self.bottleneck = ResidualBlock(512, 1024)
        # Decoder with skip connections
        self.upconv4 = nn.ConvTranspose2d(1024, 512, kernel_size=2, stride=2)
        self.dec4 = ResidualBlock(1024, 512)
        self.upconv3 = nn.ConvTranspose2d(512, 256, kernel_size=2, stride=2)
        self.dec3 = ResidualBlock(512, 256)
        self.upconv2 = nn.ConvTranspose2d(256, 128, kernel_size=2, stride=2)
        self.dec2 = ResidualBlock(256, 128)
        self.upconv1 = nn.ConvTranspose2d(128, 64, kernel_size=2, stride=2)
```

```
        self.dec1 = ResidualBlock(128, 64)
        # Final convolution
        self.final_conv = nn.Conv2d(64, 3, kernel_size=1)
        self.sigmoid = nn.Sigmoid()  # Use sigmoid to get pixel values between 0 and 1
        # Classifier head with Dropout for regularization
        self.predictor = nn.Sequential(
            nn.Linear(1024 * 4 * 4, 1024),  # Adjust according to your latent space size
            nn.ReLU(),
            nn.Dropout(0.5),  # Add dropout to reduce overfitting
            nn.Linear(1024, 10)
        )
    def forward(self, x):
        # Encoder
        e1 = self.enc1(x)
        e2 = self.enc2(self.pool(e1))
        e3 = self.enc3(self.pool(e2))
        e4 = self.enc4(self.pool(e3))
        # Bottleneck
        b = self.bottleneck(self.pool(e4))
        b_flat = b.view(b.size(0), -1)
        # Decoder with skip connections
        d4 = self.upconv4(b)
        d4 = torch.cat((d4, e4), dim=1)
        d4 = self.dec4(d4)
        d3 = self.upconv3(d4)
        d3 = torch.cat((d3, e3), dim=1)
        d3 = self.dec3(d3)
        d2 = self.upconv2(d3)
        d2 = torch.cat((d2, e2), dim=1)
        d2 = self.dec2(d2)
        d1 = self.upconv1(d2)
        d1 = torch.cat((d1, e1), dim=1)
        d1 = self.dec1(d1)
        # Final output
        x_prime = self.final_conv(d1)
        x_prime = self.sigmoid(x_prime)
        y = self.predictor(b_flat)
        return x_prime, y, b_flat
```

b. (1.0%) 選擇一個你在整個訓練過程中(包含pretraining/finetuning)所做的優化(loss function, augmentation, training scheme, …)。貼上使用/未使用這個調整的public分數，比較這兩個分數並嘗試說明原因。

使用狀況	Public 分數
使用 augmentation	0.11225
未使用 augmentation	0.09825

augmentation:隨機水平翻轉、隨機垂直翻轉、隨機旋轉、 隨機裁剪、顏色抖動、高斯模糊

- 提升泛化能力與減少過擬合

未使用資料增強時，模型容易過擬合，對訓練集表現好但對測試集效果差。增強技術增加了訓練數據的多樣性，使模型更具泛化能力，降低對特定特徵的依賴。

- 模擬真實場景的變化

資料增強（如水平翻轉、旋轉、顏色抖動）模擬了真實場景中的變化，使模型在面對不同情況的測試資料時更具適應能力，從而提升 Public 分數。

- 有效利用數據

增強技術使同一張影像以多種形式呈現，增加了數據量，使模型學習更普遍的特徵，對未見過的資料有更好的預測能力。

- 增強技術的平衡

適當選擇增強組合很重要，過度增強會導致模型難以學習穩定特徵。因此，增強中引入隨機性（如隨機高斯模糊）有助於平衡多樣性與穩定性。

2. (1.5%) Equilibrium K-means algorithm (ref: https://arxiv.org/pdf/2402.14490)
   a. (0.5%) 貼上相關程式碼(Eq38_compute_weights, Eq39_update_centroids)

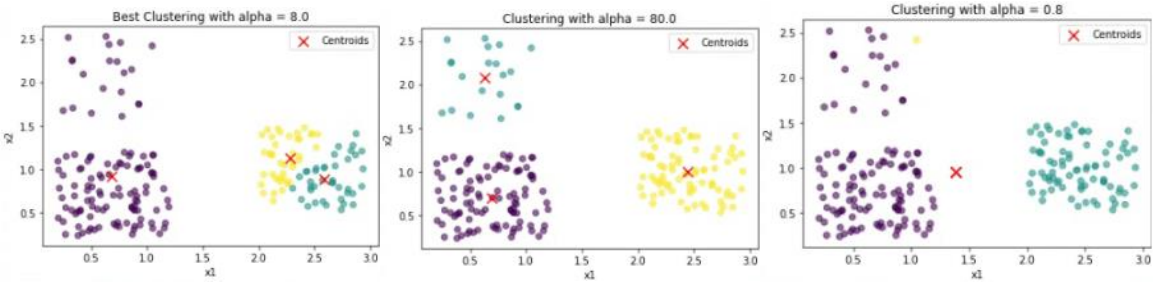
Eq38_compute_weights 函數：

```
def Eq38_compute_weights(X, centroids, alpha):
    distances = np.linalg.norm(X[:, np.newaxis] - centroids, axis=2)  # (n_samples, k_centroids)
    weights = np.exp(-alpha * distances)
    weights /= np.sum(weights, axis=1, keepdims=True)
    return weights
```

Eq39_update_centroids 函數：

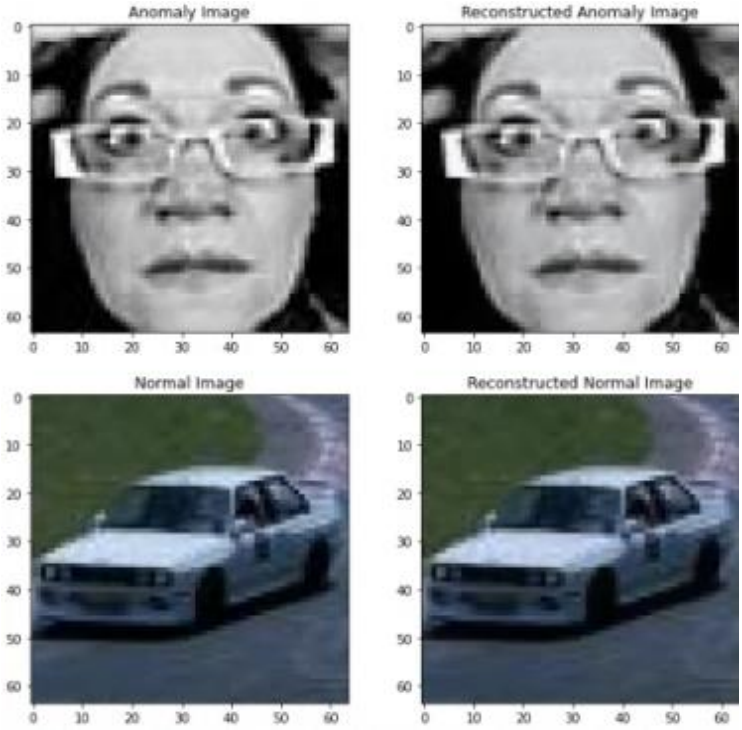
```
def Eq39_update_centroids(X, weights):
    centroids = (weights.T @ X) / np.sum(weights, axis=0)[:, np.newaxis]
    return centroids
```

   b. (1.0%) 調整alpha的數值，直到centroids分開，並且三個分群的樣本數比例大約2:1:1。再使用10x, 0.1x的數值，貼上這三個數值對應的圖片。



3. (1%) Anomaly detection
   a. 貼上執行結果的loss、圖片。(下面選一個做即可)
      i. 如果正常/異常圖片的loss跟還原的效果差很多，嘗試解釋原因。
      ii. 如果正常/異常圖片的loss跟還原的效果差不多(無法分辨anomaly)嘗試解釋原因。

iii. 使用你的pretrained model或是finetune model跑最後一個儲存格，觀察還原的效果並嘗試解釋原因。



Anomaly loss: 0.0019174122717231512

Normal loss : 0.00022390130632427626

損失差異明顯：異常圖片的損失（0.001917）明顯大於正常圖片的損失(0.000224)，這意味著模型對異常圖像的重建能力較差，因而產生較高的重建誤差。

原因：

模型訓練過程只包含正常數據：自編碼器主要是用正常圖片進行訓練，因此它擅長於重建正常類別的圖像，能夠有效地學習到正常圖像的特徵和分佈。

異常數據與正常數據的差異：異常圖片包含了模型未見過的特徵和數據分佈，導致模型無法有效地對其進行重建。這導致異常圖像的重建質量較差，進而產生較高的損失值。

特徵提取的局限性：由於模型沒有學習到異常類別的特徵，它無法很好地捕捉到異常圖像中的細節特徵，因此在還原時出現更多失真。