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：r14921A13　系級：電機所資安組碩一　姓名：鄭皓中

1. (1.5%) AutoEncoder
   a. (0.5%) Paste the **complete code** of the **AutoEncoder** used in your private submission.

```
class Autoencoder(nn.Module):
    def __init__(self):
        super(Autoencoder, self).__init__()

        # Encoder
        self.encoder = nn.Sequential(
            # Input: (B, 3, 64, 64)
            nn.Conv2d(3, 64, kernel_size=4, stride=2, padding=1),
            nn.BatchNorm2d(64),  # <-- 加入 BN
            nn.ReLU(),
            # -> (B, 64, 32, 32)

            nn.Conv2d(64, 128, kernel_size=4, stride=2, padding=1),
            nn.BatchNorm2d(128), # <-- 加入 BN
            nn.ReLU(),
            # -> (B, 128, 16, 16)

            nn.Conv2d(128, 256, kernel_size=4, stride=2, padding=1),
            nn.BatchNorm2d(256), # <-- 加入 BN
            nn.ReLU(),
            # -> (B, 256, 8, 8)

            nn.Conv2d(256, 512, kernel_size=4, stride=2, padding=1),
            nn.BatchNorm2d(512), # <-- 加入 BN
            nn.ReLU(),
            # -> (B, 512, 4, 4)

            nn.Flatten(start_dim=1),    # -> (B, 512*4*4)

            nn.Linear(512*4*4, 1024),
            nn.BatchNorm1d(1024), # <-- 扁平化後使用 1D BN
            nn.ReLU()             # <-- 建議在 latent vector 之前也加上 ReLU
            # -> (B, 1024)
        )
```

```python
# Decoder
self.decoder = nn.Sequential(
    # Input: (B, 1024)
    nn.Linear(1024, 512*4*4),
    nn.BatchNorm1d(512*4*4), # <-- 加入 1D BN
    nn.ReLU(),
    # -> (B, 512*4*4)

    nn.Unflatten(dim=1, unflattened_size=(512, 4, 4)), # -> (B, 512, 4, 4)

    nn.ConvTranspose2d(512, 256, kernel_size=4, stride=2, padding=1),
    nn.BatchNorm2d(256), # <-- 加入 BN
    nn.ReLU(),
    # -> (B, 256, 8, 8)

    nn.ConvTranspose2d(256, 128, kernel_size=4, stride=2, padding=1),
    nn.BatchNorm2d(128), # <-- 加入 BN
    nn.ReLU(),
    # -> (B, 128, 16, 16)

    nn.ConvTranspose2d(128, 64, kernel_size=4, stride=2, padding=1),
    nn.BatchNorm2d(64),  # <-- 加入 BN
    nn.ReLU(),
    # -> (B, 64, 32, 32)

    nn.ConvTranspose2d(64, 3, kernel_size=4, stride=2, padding=1),
    # -> (B, 3, 64, 64)

    nn.Sigmoid()  # 輸出層 (0-1), 不加 BN 或 ReLU
)
```

```python
    # classifier head
    self.predictor = nn.Sequential(
        # Input: (B, 1024)
        nn.Linear(1024, 1024),
        nn.BatchNorm1d(1024), # <-- 加入 1D BN
        nn.ReLU(),

        nn.Linear(1024, 10)   # 輸出層 (logits), 不加 BN 或 ReLU
        # -> (B, 10)
    )

def forward(self, x):
    # encode
    z = self.encoder(x)
    # decode
    x_prime = self.decoder(z)
    # classify
    y = self.predictor(z)
    return x_prime, y, z
```

b. (1.0%) Choose **one** optimization (loss function, data augmentation, etc.) you applied during the entire training process (including both pre-training and fine-tuning). Paste the **public scores** obtained **with** and **without** this optimization, compare the two results, and try to explain the reason for the difference.

withaug.csv 0.61275
Complete · 2h ago

woaug.csv 0.45650
Complete · now

上面是with data augmentation 的public score，下圖則是只使用Resize及ToTensor的public score，透過觀察調整model時的Train跟Valid acc可以發現Train acc是一路上升到0.98，但Valid acc則是卡在0.5左右，這應該是overfitting的部分，因為我們沒有對data進行任何的augmentation，導致模型直接背圖片特徵而沒有關注整體，所以Train學得非常好但是Valid卻很糟糕。經過了Data augmentation調整圖片的狀態，包含翻轉、色彩變換、隨機仿射，讓model沒有辦法直接背圖片特徵，而是需要關注整體後，Train跟Valid的acc就一起緩慢上升，而非Overfitting下Train跟Valid差距懸殊的狀況。

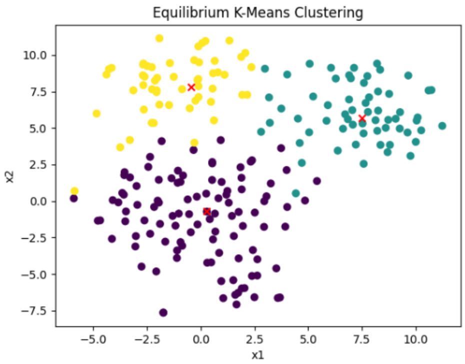
2. (1.5%) **Equilibrium K-means Algorithm** (ref: https://arxiv.org/pdf/2402.14490)

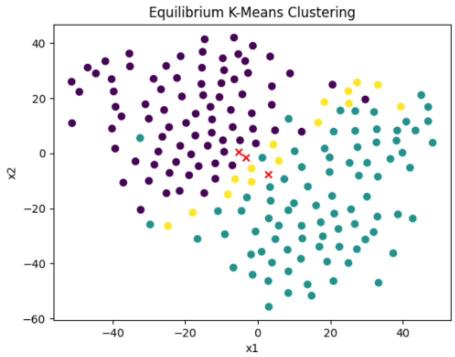
a. (0.5%) Paste the relevant code sections (**Eq38_compute_weights**, **Eq39_update_centroids**).

```python
def Eq38_compute_weights(X, centroids, alpha):
    #==== TODO: Compute the weights for each data point (refer to Eq. 38) ====#

    # X: (N, D) - N 個資料點, D 維特徵
    # centroids: (K, D) - K 個中心點, D 維特徵

    # 1. 計算 d_kn = 0.5 * ||x_n - c_k||^2
    # (N, 1, D) - (1, K, D) -> (N, K, D) -> sum over D -> (N, K)
    dist = 0.5 * np.sum(np.square(X[:, np.newaxis, :] - centroids[np.newaxis, :, :]), axis=2)

    # 2. 計算 e^(-alpha * d_kn)
    exp_dist = np.exp(-alpha * dist) # (N, K)

    # 3. 計算分母 sum_i(e^(-alpha * d_in))
    sum_exp_dist = np.sum(exp_dist, axis=1, keepdims=True) # (N, 1)

    # 4. 計算第一項 (softmax)
    term1 = exp_dist / sum_exp_dist # (N, K)

    # 5. 計算括號中的 d_kn
    term2_d_kn = dist # (N, K)

    # 6. 計算括號中的 weighted average d_in
    # (N, K) * (N, K) -> sum over K -> (N,) -> (N, 1)
    weighted_avg_dist = np.sum(dist * exp_dist, axis=1, keepdims=True) / sum_exp_dist # (N, 1)

    # 7. 計算括號中的 [d_kn - avg_d]
    term2 = 1 - alpha * (term2_d_kn - weighted_avg_dist) # (N, K)

    # 8. 總權重
    weights = term1 * term2 # (N, K)

    #========================================================================#
    return weights
```

```python
def Eq39_update_centroids(X, weights):
    #==== TODO: Update the centroids (refer to Eq. 39) ====#
    # X: (N, D)
    # weights: (N, K)

    # 1. 計算分母: sum_n(w_kn) for each k
    sum_weights = np.sum(weights, axis=0) # (K,)

    # 2. 計算分子: sum_n(w_kn * x_n) for each k
    # (K, N) @ (N, D) -> (K, D)
    weighted_sum_X = weights.T @ X

    # 3. 相除 (K, D) / (K, 1)
    centroids = weighted_sum_X / sum_weights[:, np.newaxis]


    #=======================================================#
    return centroids
```

b. (1.0%) Adjust the value of **α (alpha)** until the centroids are separ ated and the sample size among the three clusters is approximatel y **2:1:1**. Then, using **10×** and **0.1×** of that α value, paste the corresponding three images and compare them.





Left : alpha = 0.1, the three clusters are 100 : 55 : 45

Right : alpha = 0.01

在使用alpha = 1 的過程中出現了error message: TSNE does not accept missing values encoded as NaN natively。由這個錯誤訊息可以發現此時Equilibrium-K-means Algorithm產生了NaN的中心點，導致分群失敗。

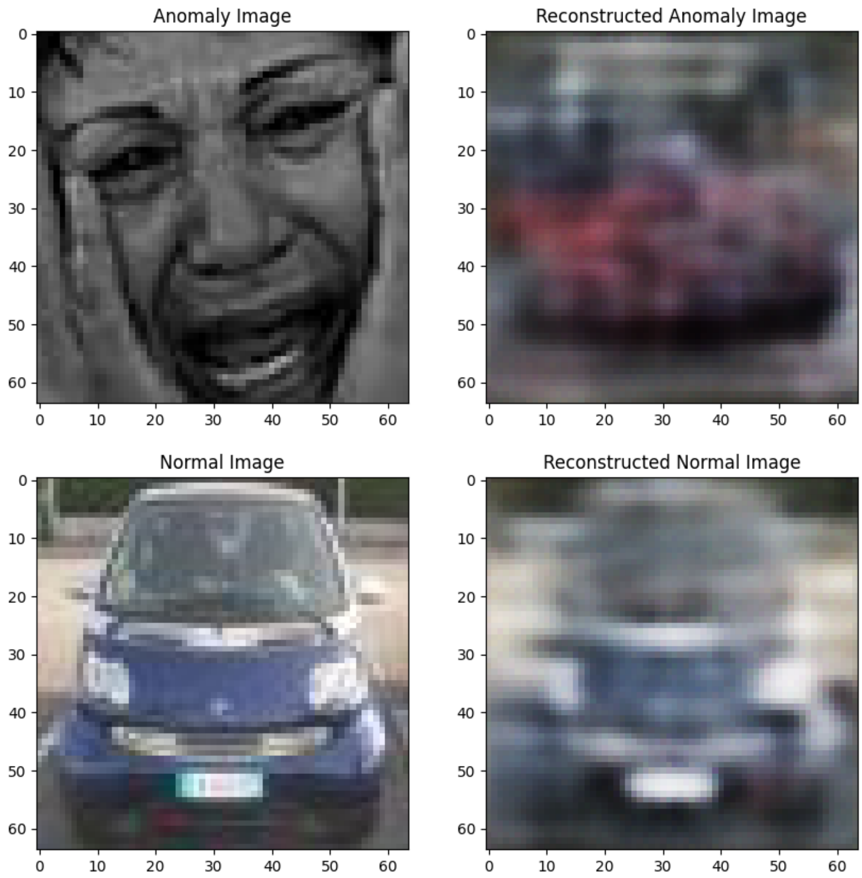
而由上面的圖也可以發現當alpha=0.01的時候三個群的中心點非常接近，可以看出當alpha過小的時候會導致分群上面混在一起，無法清楚地進行clustering。

3. (1%) **Anomaly Detection**

Paste the **loss values** and the **corresponding images** from the results. (Choose **one** of the following options to answer.)

- If the loss and reconstruction quality differ significantly between **normal** and **anomalous** images, try to explain the reason.
- If the loss and reconstruction quality are **similar** (i.e., the model fails to distinguish anomalies), try to explain the reason.

Finally, use your **pre-trained model** or **fine-tuned model** to run the **last cell** in the given .ipynb, observe the reconstruction results, and explain your observations.



Anomaly loss: 0.055664725601673126

Normal loss : 0.014214593281466 81

由於Anomaly loss跟Normal loss的差距很大，這可能是因為Autoencoder對於汽車這一個class已經非常熟悉，所以在判讀一般汽車圖片時可以很好的去Decode並重新生成，但當遇到非熟悉的class(Ex：人像)的時候，他則是會試圖去將其以汽車的class進行還原，產生極大的loss。透過這個方式我們可以判讀產生極大的loss的部分可能是Anomaly的圖片。