學號：r12227118　系級：心理所碩二　姓名：張瑋宸

1. (1.5%) AutoEncoder model
   a. (0.5%) 貼上private submission所使用的AutoEncoder model程式碼。
   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。

a.

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

  # Encoder
  self.encoder = nn.Sequential(
   nn.Conv2d(3, 128, kernel_size=4, stride=2, padding=1),
   nn.BatchNorm2d(128),
   nn.ReLU(),
   nn.Conv2d(128, 128, kernel_size=4, stride=2, padding=1),
   nn.BatchNorm2d(128),
   nn.ReLU(),
   nn.Conv2d(128, 256, kernel_size=4, stride=2, padding=1),
   nn.BatchNorm2d(256),
   nn.ReLU(),
   nn.Conv2d(256, 512, kernel_size=4, stride=2, padding=1),
   nn.BatchNorm2d(512),
   nn.ReLU(),
   nn.Flatten(start_dim=1),  # Flatten to feed into the linear layer
   nn.Linear(512*4*4, 1024) # change to 2048
  )

  # Decoder
  self.decoder = nn.Sequential(
   nn.Linear(1024, 512*4*4),
   nn.Unflatten(dim=1, unflattened_size=(512, 4, 4)),  # Unflatten
   nn.ConvTranspose2d(512, 256, kernel_size=4, stride=2, padding=1),
   nn.BatchNorm2d(256),
   nn.ReLU(),
   nn.ConvTranspose2d(256, 128, kernel_size=4, stride=2, padding=1),
```

```
      nn.BatchNorm2d(128),
      nn.ReLU(),
      nn.ConvTranspose2d(128, 64, kernel_size=4, stride=2, padding=1),
      nn.BatchNorm2d(64),
      nn.ReLU(),
      nn.ConvTranspose2d(64, 3, kernel_size=4, stride=2, padding=1),
      nn.Sigmoid() # Use sigmoid to get pixel values between 0 and 1
    )

    # classifier head
    self.predictor = nn.Sequential(
      nn.Linear(1024, 1024),  # Latent space with 1024 features
      nn.ReLU(),
      nn.Dropout(0.3),
      nn.Linear(1024, 256),
      nn.ReLU(),
      nn.Dropout(0.3),
      nn.Linear(256, 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. 在 finetune 時 data是否有做data agmentation，上面是沒進行的public score，而下面是有進行data agmentation。可以看見有做的表現(ACC)比較好，原因可能是沒做data agmentation會使模型學到太侷限於某些樣本的特徵進行分類而不是比較robust的特徵，產生overfitting，在training 時分數很高但是validation就表現不好。

predict.csv Complete · now · w/ batch normalization, no agmentation	0.43925	☐
predict.csv Complete · 21h ago · w/ batch normalization	0.55025	☑

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)

   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。

a. 如下

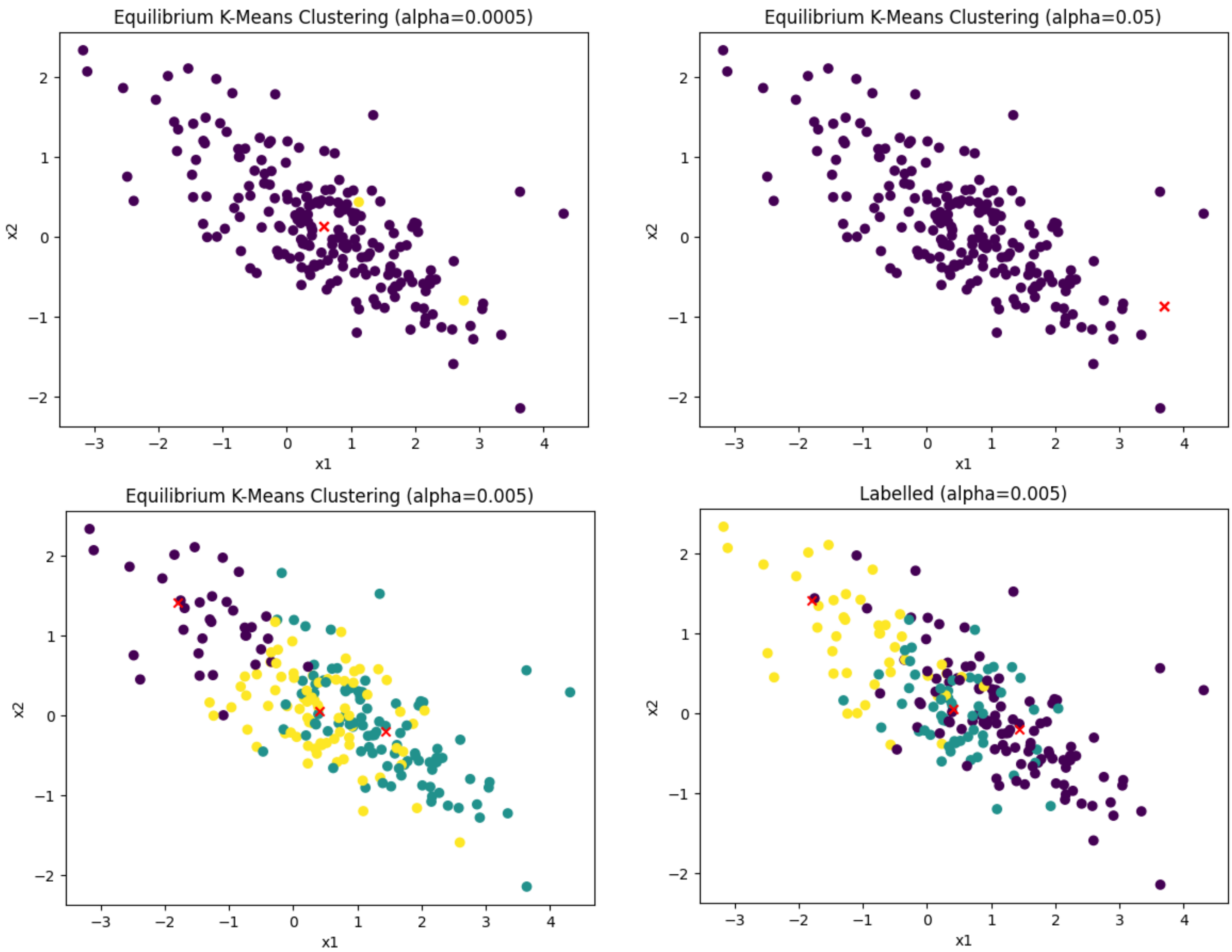
```
def Eq38_compute_weights(X, centroids, alpha):
  def distance(x1, x2):
    return (0.5 * np.linalg.norm(x1 - x2)**2)

  weights = np.zeros((X.shape[0], centroids.shape[0]))

  for n in range(X.shape[0]): # n_observation
    for k in range(centroids.shape[0]): # K Dimension
      d_kn = distance(X[n], centroids[k])
      numerator_1 = np.exp(-(alpha * d_kn))
      denominator = np.sum([np.exp(-(alpha * distance(X[n], centroids[i]))) for i in range(centroids.shape[0])])
      denominator += 1e-8 # avoid denominator=0
      numerator_2 = np.sum([distance(X[n], centroids[i]) * np.exp(-(alpha * distance(X[n], centroids[i]))) for i in range(centroids.shape[0])])
      weight = (numerator_1 / denominator) * (1 - (alpha * (d_kn - (numerator_2 / denominator))))
      weights[n, k] = weight
  return weights

def Eq39_update_centroids(X, weights):
  K = weights.shape[1]
  centroids = np.zeros((K, X.shape[1]))
  for k in range(K): # K
   # The weights for the current cluster k
   cluster_weights = weights[:, k]
   # Apply weights to the corresponding data points before summing
   numerator = np.sum(cluster_weights[:, np.newaxis] * X, axis=0)
   denominator = np.sum(weights[:, k]) + 1e-8 # avoid denominator=0
   centroids[k] = numerator / denominator
  return centroids
```

b. 如下圖所示，alpha=0.005 時候分群表現結果最好，再小或再大，centroid都會集中於一點，只是該點的位置不同。



使用Equilibrium K-Means Clustering再以t-SNE降維的視覺化結果。
左上、右上、左下：alpha參數不同時的 clustering 結果；右下：資料實際 label

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。

Ans:

還原的效果差不少，Anormal loss 比 Normal loss大，且還原與原圖差異甚大看不出原本樣子，Normal還看的出來車子的輪廓。原因可能是原先訓練的Autoencoder主要針對物品圖片進行還原，此模型在人臉(Anormal data)上找到的latent feature無法有效表現出原先的圖片，因此還原效果較差。

Output:

Anomaly loss: 0.05591103434562683
Normal loss : 0.010838131627274884