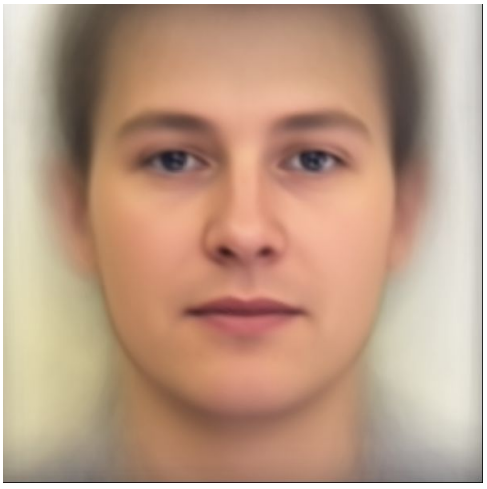
HW4

學號：B05902042 系級：資工二 姓名：林瑋毅

A. PCA of colored faces

A.1. (.5%) 請畫出所有臉的平均。



A.2. (.5%) 請畫出前四個 Eigenfaces，也就是對應到前四大 Eigenvalues 的 Eigenvectors。

以下由左至右分別為對應到第一至第四大特徵值的特徵向量。

A.3. (.5%) 請從數據集中挑出任意四個圖片，並用前四大 Eigenfaces 進行 reconstruction，並畫出結果。

以下由左至右分別為 [0-3].jpg 用前四大 eigenfaces 重建的結果。

A.4. (.5%) 請寫出前四大 Eigenfaces各自所佔的比重，請用百分比表示並四捨五入到小數點後一位。

以前四大 eigenfaces 對應的特徵值佔所有特徵值之和的大小可得其所佔的比重分別為 4.1%、2.9%、2.4%、2.2%。

B. Image clustering

B.1. (.5%) 請比較至少兩種不同的 feature extraction 及其結果。(不同的降維方法或不同的 cluster 方法都可以算是不同的方法)

方法一：Autoencoder + KMeans++

Autoencoder 由 pytorch 實作，架構如下：

```
AutoEncoder(
  (encoder): Sequential(
    (0): Linear(in_features=784, out_features=128, bias=True)
    (1): ReLU(inplace)
    (2): Linear(in_features=128, out_features=64, bias=True)
    (3): ReLU(inplace)
    (4): Linear(in_features=64, out_features=32, bias=True)
  )
  (decoder): Sequential(
    (0): Linear(in_features=32, out_features=64, bias=True)
    (1): ReLU(inplace)
    (2): Linear(in_features=64, out_features=128, bias=True)
    (3): ReLU(inplace)
    (4): Linear(in_features=128, out_features=784, bias=True)
    (5): Sigmoid()
  )
)
```

其中 optimizer 是預設參數的 Adam ，criterion 是預設參數的 MSELoss ，訓練了 100 個 epoch，並無使用 data augmentation。

KMeans++ 由 scikit-learn 的 KMeans 實作，參數如下：

- `algorithm='auto'`
- `copy_x=True`
- `init='k-means++'`
- `max_iter=300`
- `n_clusters=2`
- `n_init=10`
- `n_jobs=1`
- `precompute_distances='auto'`
- `random_state=0`
- `tol=0.0001`
- `verbose=0`

結果如下：private / public = 0.71512 / 0.71522

方法二：PCA + KMeans++

先直接把原始資料壓平成 784 維，再用 PCA 降成 32 維，最後用與方法一相同的方法進行 cluster。

結果如下：private / public = 0.51739 / 0.51663

可以發現，方法一 (autoencoder) extract features 的能力較方法二

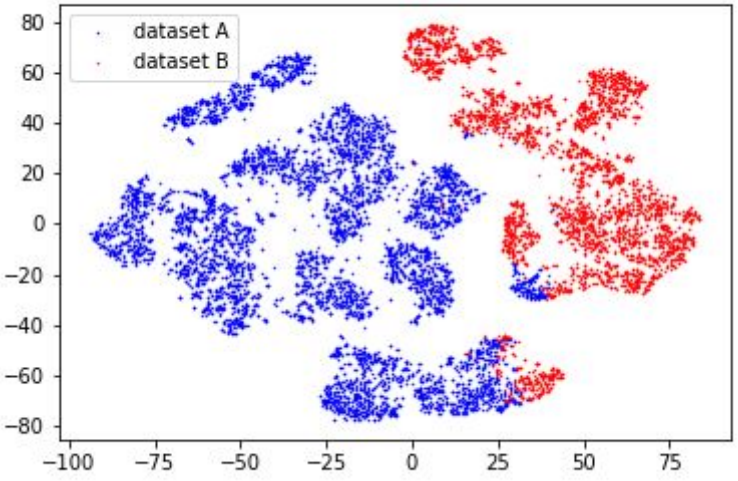
(PCA) 好。

B.2. (.5%) 預測 visualization.npy 中的 label，在二維平面上視覺化 label 的分佈。

此題與 B.3. 題降維 (feature extraction) 的方法都與 B.1. 題的第一個方法相同，不再重述。視覺化 feature 的方法則是採用 TSNE，實作上採用 scikit-learn 的 TSNE，參數為

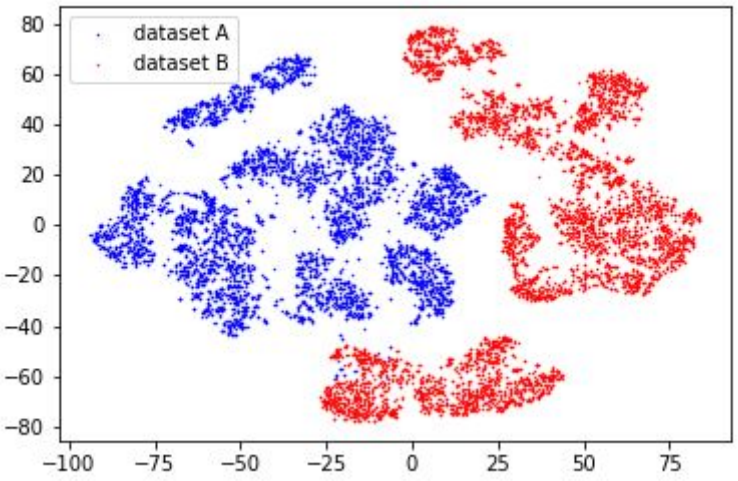
- `angle=0.5`
- `early_exaggeration=12.0`
- `init='pca'`
- `learning_rate=200.0`
- `method='barnes_hut'`
- `metric='euclidean'`
- `min_grad_norm=1e-07`
- `n_components=2`
- `n_iter=1000`
- `n_iter_without_progress=300`
- `perplexity=30.0`
- `random_state=None`
- `verbose=0`

結果如下：



B.3. (.5%) visualization.npy 中前 5000 個 images 跟後 5000 個 images 來自不同 dataset。請根據這個資訊，在二維平面上視覺化 label 的分佈，接著比較和自己預測的 label 之間有何不同。

視覺化之後如下：



可以發現有部分的 label 使用 kmeans clustering 會分類錯誤 (x=[-25,25], y=[-80,-40]的那一群)，原因可能是 dataset 有些圖片在 feature extract 後還是太過相近，導致 Kmeans 沒辦法正確地 cluster。

C. Ensemble learning

C.1. (1.5%) 請在hw1/hw2/hw3的task上擇一實作ensemble learning，請比較其與未使用ensemble method的模型在 public/private score 的表現並詳細說明你實作的方法。（所有跟ensemble learning有關的方法都可以，不需要像hw3的要求硬塞到同一個model中）

此題使用的 ensemble 方法為 bagging，先分別訓練了四個架構如附錄的 model，預測時，分別用四個 model 計算每個 label 的機率，再求四個 model 預測機率的平均值，並取最大的那個 label 做為預測結果。

使用單個 base learner 的結果如下：

private / public : 0.69322 / 0.71524

使用四個 base learner 的結果如下：

private / public : 0.71106 / 0.73390

可見進行 ensemble 可以有更好的準確率。

附錄：

C題使用的 base learner 架構如下，其中 optimizer 使用 SGD (lr=.1, momentum=.9, nesterov=True)，訓練了 180 個 epoch：

```
WideResNet(
  (init): Sequential(
```

```
      (0): Conv2d(1, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True)
      (2): ReLU(inplace)
)
(layer1): Sequential(
      (0): WideBlock(
      (block1): Sequential(
      (0): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True)
      (1): ReLU(inplace)
      )
      (block2): Sequential(
      (0): Conv2d(16, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
      (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True)
      (2): ReLU(inplace)
      (3): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      )
      (downsample): Sequential(
      (0): Conv2d(16, 128, kernel_size=(1, 1), stride=(2, 2))
      )
      )
      (1): WideBlock(
      (block1): Sequential(
      (0): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True)
      (1): ReLU(inplace)
      )
      (block2): Sequential(
      (0): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True)
      (2): ReLU(inplace)
      (3): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      )
      )
      (2): WideBlock(
      (block1): Sequential(
      (0): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True)
      (1): ReLU(inplace)
      )
      (block2): Sequential(
      (0): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True)
      (2): ReLU(inplace)
      (3): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      )
      )
)
(layer2): Sequential(
      (0): WideBlock(
      (block1): Sequential(
```

```
      (0): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True)
      (1): ReLU(inplace)
      )
      (block2): Sequential(
      (0): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
      (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True)
      (2): ReLU(inplace)
      (3): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      )
      (downsample): Sequential(
      (0): Conv2d(128, 256, kernel_size=(1, 1), stride=(2, 2))
      )
      )
      (1): WideBlock(
      (block1): Sequential(
      (0): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True)
      (1): ReLU(inplace)
      )
      (block2): Sequential(
      (0): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True)
      (2): ReLU(inplace)
      (3): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      )
      )
      (2): WideBlock(
      (block1): Sequential(
      (0): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True)
      (1): ReLU(inplace)
      )
      (block2): Sequential(
      (0): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True)
      (2): ReLU(inplace)
      (3): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      )
      )
)
(layer3): Sequential(
      (0): WideBlock(
      (block1): Sequential(
      (0): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True)
      (1): ReLU(inplace)
      )
      (block2): Sequential(
      (0): Conv2d(256, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
      (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True)
      (2): ReLU(inplace)
```

```
            (3): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
            )
            (downsample): Sequential(
            (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2))
            )
            )
            (1): WideBlock(
            (block1): Sequential(
            (0): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True)
            (1): ReLU(inplace)
            )
            (block2): Sequential(
            (0): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
            (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True)
            (2): ReLU(inplace)
            (3): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
            )
            )
            (2): WideBlock(
            (block1): Sequential(
            (0): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True)
            (1): ReLU(inplace)
            )
            (block2): Sequential(
            (0): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
            (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True)
            (2): ReLU(inplace)
            (3): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
            )
            )
      )
      (end): Sequential(
            (0): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True)
            (1): ReLU(inplace)
            (2): AvgPool2d(kernel_size=6, stride=1, padding=0, ceil_mode=False,
    count_include_pad=True)
      )
      (fc): Sequential(
            (0): Linear(in_features=512, out_features=7, bias=True)
            (1): LogSoftmax()
      )
    )
```

另外，訓練時的 data augmentation 包括:

```
transforms.RandomResizedCrop(48, scale=(0.9, 1), ratio=(0.9, 1.1)),
transforms.ColorJitter(brightness=0.1, contrast=0.1, saturation=0.1, hue=0.1),
transforms.RandomHorizontalFlip(),
transforms.RandomCrop(size=46, padding=2),
```

```
transforms.ToTensor()
```