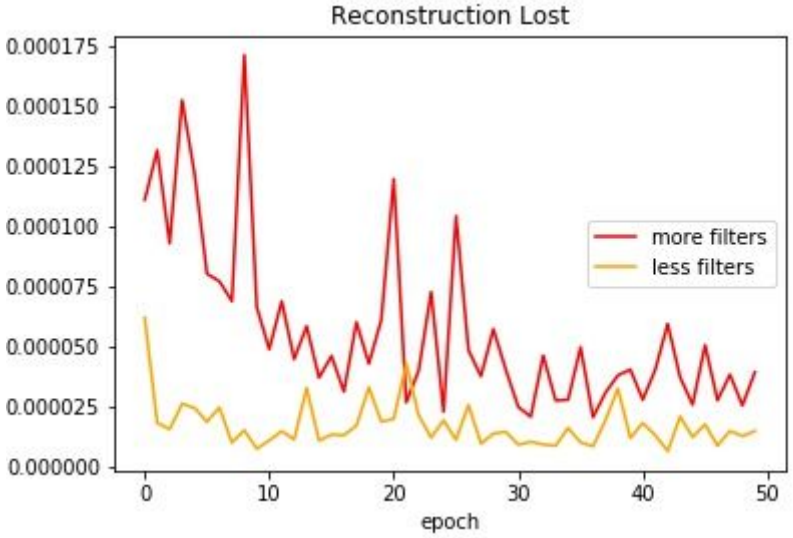
學號：B04303006 系級： 經濟四 姓名：劉傳筠

1. (1%) 請使用不同的Autoencoder model，以及不同的降維方式(降到不同維度)，討論其reconstruction loss & public / private accuracy。（因此模型需要兩種，降維方法也需要兩種，但clustrering不用兩種。）
   (a) Reconstruction Loss:



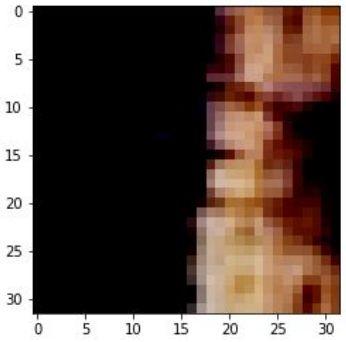
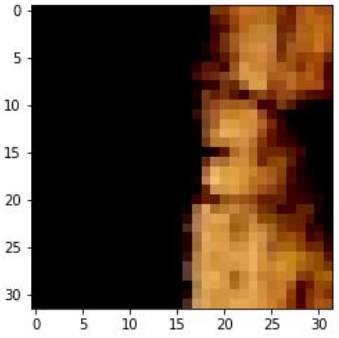
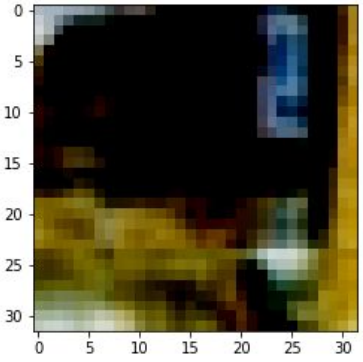
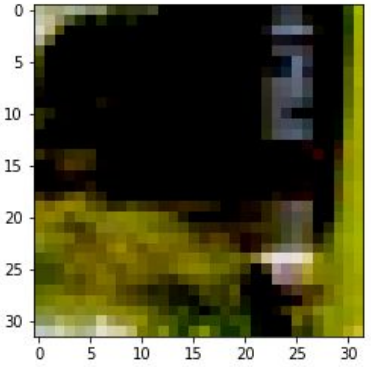
使用兩種不同的autoencoder: 第一種autoencoder每層的的filters較多(64,128,256)第二種autoencoder的filters較少(8,16,32)。觀察可以發現fulters較多會導致初期的reconstruction loss較高，應該是因為參數較多的原因。隨著epoch增加reconstruction loss皆逐漸下降，不過較多filters的仍然loss較高，應該是因為這裡epoch設較小的緣故。

(b) Private / Public Accuracy:

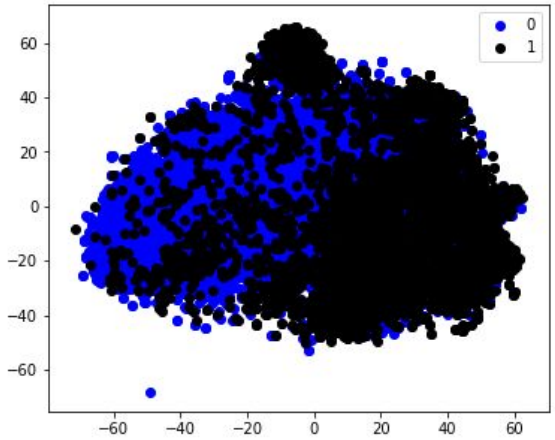
	較多channels	較少channels
w/ TSNE	0.82238 / 0.82259	0.83301 / 0.83629
wo/ TSNE	0.58333 / 0.58740	0.53539 / 0.54740

降維使用PCA(200 components)->FactorAnalysis(50 componenets)->(TSNE(2 components))->MinibatchKmeans(2 clusters)，其中一種有用tsne一種沒有。根據kaggle的private & public分數，有用tsne明顯好很多，不過多和少的channels就沒有像有沒有tsne的差距這麼巨大。

2. (1%) 從dataset選出2張圖，並貼上原圖以及經過autoencoder後reconstruct的圖片。
使用較多filters的model(64,128,256)，效果還不錯，雖然顏色會改變但是形狀都有抓到。

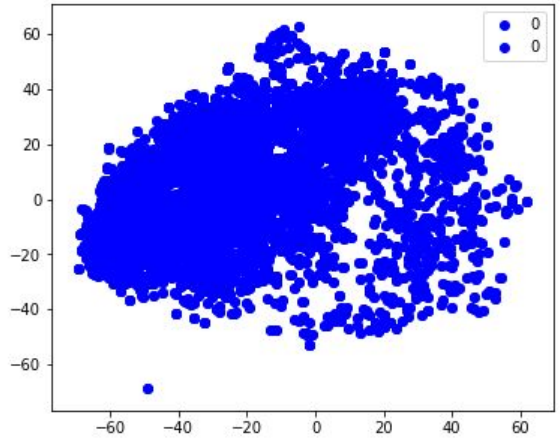
Reconstructed	Original
	
	

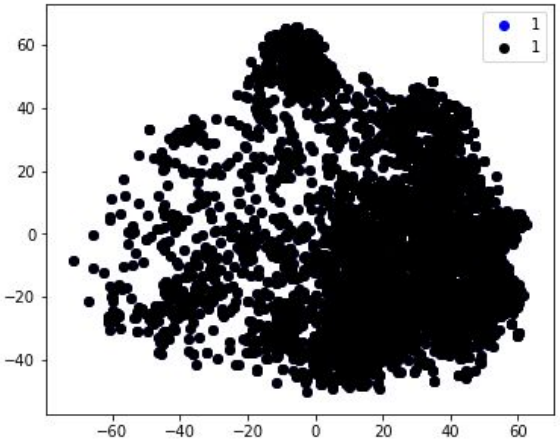
3. (1%) 在之後我們會給你dataset的label。請在二維平面上視覺化label的分佈。
True label:



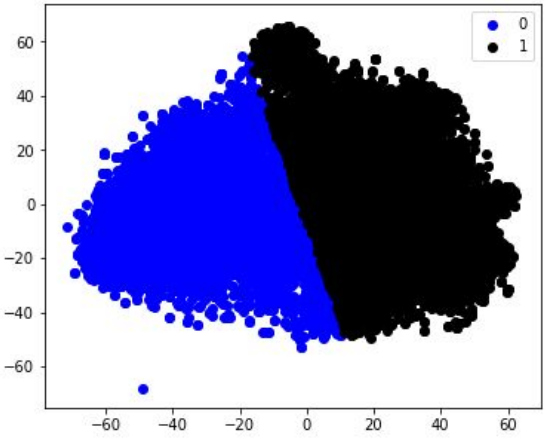
看起來好像沒有分得很開...不過彼此覆蓋的部份較少。用minibatch kmeans分群之後正確率約為83.3(both private & public, 無條件捨去至小數點第二位)

個別：





用MiniBatch Kmeans分群：



4. (3%)Refer to math problem https://drive.google.com/file/d/1e_IDAV2yv0YEhIuVWpDdaH4Pzz5s1p2P/view?fbclid=IwAR0tO9NRxK9JZeUDNdawNuSbGTvqI7niuMX3Kkk9arauC8O6p6iJc7oMz84

ML HW4 Handwritten. ◆ date/ 11/29 ◆ page/ 1

1. (a) (無條件捨去至小數第三位)

[-0.616, -0.588, -0.522]

[-0.678, 0.734, -0.027]

[0.399, 0.337, -0.852]

(b) (無條件捨去至小數第一位)

(-3.36, 0.70, -1.48)

(-9.78, 3.02, 0.03)

(-13.61, 6.53, -2.41)

(-7.94, 5.06, -1.16)

(-12.37, 6.83, 5.02)

(-7.19, -1.83, 3.29)

(-14.96, -0.47, +1.36)

(-7.08, 3.81, 3.04)

(-12.86, -3.95, 0.97)

(-16.30, 1.10, 1.74)

(c) 6.064

2. (a) • $x^T A^T A x = (Ax)^T(Ax) = \|Ax\|^2 \geq 0$, $\forall x \in R^n$ and

$y^T A A^T y = (A^T y)^T(A^T y) = \|A^T y\|^2 \geq 0$, $\forall y \in R^m$

Also $(A^T A)^T = A^T A$ & $(AA^T)^T = AA^T$

∴ They're both symmetric positive semi-definite

∴ Their eigen values are non-negative.

• Next, show they share the same eigen values

Let λ be an eigen value of $A^T A$,

$$A^T A x = \lambda \cdot x$$

both multiply by A from left:

$$A \cdot A^T (Ax) = \lambda (Ax)$$

∴ For each eigen value of $A^T A$ (corresponding to eigen vector x), $A A^T$ has the same eigen value (corresponding to Ax). ∴ They share the same non-zero eigenvalues

(b) ∵ Σ is positive semi-definite.
It can be diagonalized to $C\cdot\Lambda^{-1}C^T$
∵ $C\cdot\Lambda^{-1}\cdot C^T = C\cdot\Lambda^{-\frac{1}{2}}\cdot(C\cdot\Lambda^{-\frac{1}{2}})^T$
∴ $C\cdot\Lambda^{-\frac{1}{2}}$ is the $(X-\mu)$ in our case.
∴ given Σ & μ, we can derive X. ※

(c) $\Phi \in R^{m\times k}$
$\min Tr[\Phi^T\Sigma\Phi]$
s.t. $\Phi^T\Phi = I$

$Tr[\Phi^T\Sigma\Phi]$
$= Tr[\Phi^T\underbrace{(I-W)(I-W)^T}_{\Sigma}\Phi]$
$= Tr[\Phi^T(I-W)[\Phi^T(I-W)]^T]$
$= \sum_i \| y_i - \sum_j W_{ij} y_j \|^2$ in LLE

The solution of the problem is obtained from the set of eigen vectors associated with the k smallest non-zero eigen values of Σ (or $(I-W)(I-W^T)$).

<pf> $\min \sum_j \phi_j^T\Sigma\phi_j$
s.t. $\sum\phi_j^T\phi_j = k$
$(\Phi^T\Phi = I_k)$
Lagrange: $L = \sum_j \phi_j^T\Sigma\phi_j - \lambda(\sum_j \phi_j^T\phi_j - k)$

$\frac{\partial L}{\partial \phi_j} = 2\Sigma\phi_j - 2\lambda\phi_j = 0$ ⎫
$\frac{\partial L}{\partial \lambda} = \sum_j \phi_j^T\phi_j - k = 0$ ⎭ $\Rightarrow \Sigma\Phi = \lambda\Phi$

Φ: eigen vectors to minimize $Tr(\Phi^T\Sigma\Phi)$
⇒ choose smallest k eigen vectors.

PCA as class slides

The function below is refined from this article.

In [3]:
```
def pca_calc(X, k):
    m, n = np.shape(X)
    covX = np.cov(X.T)   #计算协方差矩阵
    featValue, featVec=  np.linalg.eig(covX)  #求解协方差矩阵的特征值和特征向量
    index = np.argsort(-featValue) #按照featValue进行从大到小排序
    finalData = []
    if k > n:
        print("k must lower than feature number")
        return
    else:
        #注意特征向量时列向量，而numpy的二维矩阵(数组)a[m][n]中，a[1]表示第1行值
        selectVec = np.matrix(featVec.T[index[:k]]) #所以这里需要进行转置
        finalData = X * selectVec.T
        reconData = finalData * selectVec
    return finalData, reconData, selectVec
```

In [4]:
```
finalData, reconData, selectVec = pca_calc(X, 2)
```

In [5]:
```
selectVec
```

Out[5]:
```
matrix([[-0.6165947 , -0.58881629, -0.52259579],
        [-0.67817891,  0.73439013, -0.02728563]])
```

In [6]:
```
finalData
```

Out[6]:
```
matrix([[ -3.36201464,   0.70874446],
        [ -9.78988804,   3.02597728],
        [-13.61894165,   6.53257419],
        [ -7.94010395,   5.06051399],
        [-12.37159312,   6.83599606],
        [ -7.19402383,  -1.83697744],
        [-14.96324467,  -0.47405978],
        [ -7.0829102 ,   3.81329871],
        [-12.86219784,  -3.95173109],
        [-16.30109667,   1.10550298]])
```

In [7]:
```
reconData
```

Out[7]:
```
matrix([[ 1.59234485,  2.50010393,  1.73763614],
        [ 3.98423909,  7.98669341,  5.03358854],
        [ 3.96711319, 12.81651271,  6.93895612],
        [ 1.46389215,  8.39165409,  4.01138555],
        [ 2.99223039, 12.30488361,  6.27881797],
        [ 5.6815963 ,  2.88690031,  3.80968963],
        [ 9.54775466,  8.46245739,  7.83266364],
        [ 1.78118612,  6.97098184,  3.59745077],
        [10.61074365,  4.67135929,  6.82955587],
        [ 9.30144096, 10.41022174,  8.48872009]])
```

In [8]:
```
diff = (reconData - X)
loss = 0
for i in range(10):
    loss += diff[i] * diff[i].T
loss / 10
```

Out[8]:
```
matrix([[6.06438305]])
```