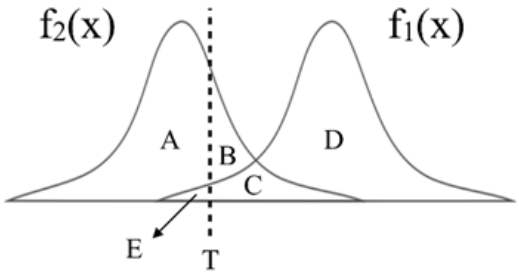
Up Computer Vision: from Recognition to Geometry
HW2

Name: 楊宗山　Department: 電信所　Student ID: r08942065

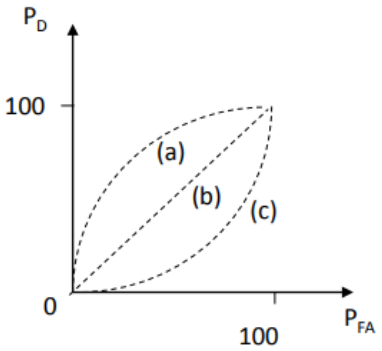
Problem 1

(a) Assume X is a continuous random variable that denotes the estimated probability of a binary classifier. The instance is classified as positive if $X > T$ and negative otherwise. When the instance is positive, X follows a PDF $f_1(x)$. When the instance is negative, X follows a PDF $f_2(x)$. Please specify which regions (A ~ E) represent the cases of *False Positive* and *False Negative*, respectively. Clearly explain why. (6%)



B, C: False positive; E: False negative
If x is greater than T, x will be labeled as f_1. As a result, false positive includes the area bounded by $f_2(x)$ and $x>T$.
Similarly, false negative area is bounded by $x<T$ and $f_1(x)$.

(b) There are three ROC curves in the plot below. Please specify which ROC curves are considered to have reasonable discriminating ability, and which are not. Also, please answer that under what circumstances will the ROC curve fall on curve (b)? (6%)



(a) is considered to have reasonable discriminating ability, and (c) is not.

If the distribution of $class_0$ is the same as $class_1$, the probability of detection and false alert will be the equal.

Problem 2

(a) Given a convolutional layer which contains:

n kernels with size $k*k*n_{in}$

padding size p,

stride size (s,s).

With input feature size $W*W*n_{in}$, calculate the size of output feature $W_{out}*W_{out}*n_{out}$.

(i) W_out=? (2%)

(ii) n_out =? (2%)

Ans: (i) $W_{out} = \lfloor \frac{W + 2*p - (k-1) - 1}{s} + 1 \rfloor$

(ii) n

(b) k=5, s=2, p=1, n=256, W=64, calculate the number of parameters in the convolutional layer (4%)

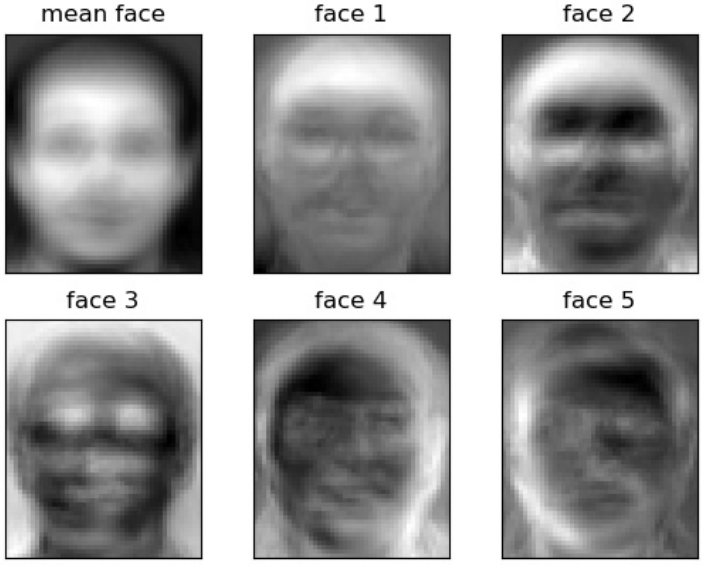
Ans: $(5 * 5 * n_{in}) * 256$

Problem 3 (report 35% + code 5%)

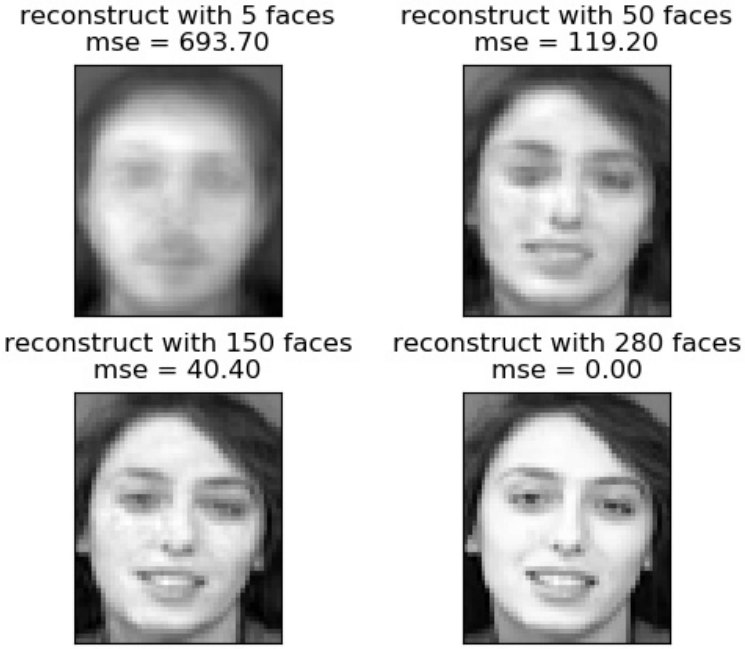
(a) PCA (15%)

In this task, you need to implement PCA from scratch, which means you cannot call PCA function directly from existing packages.

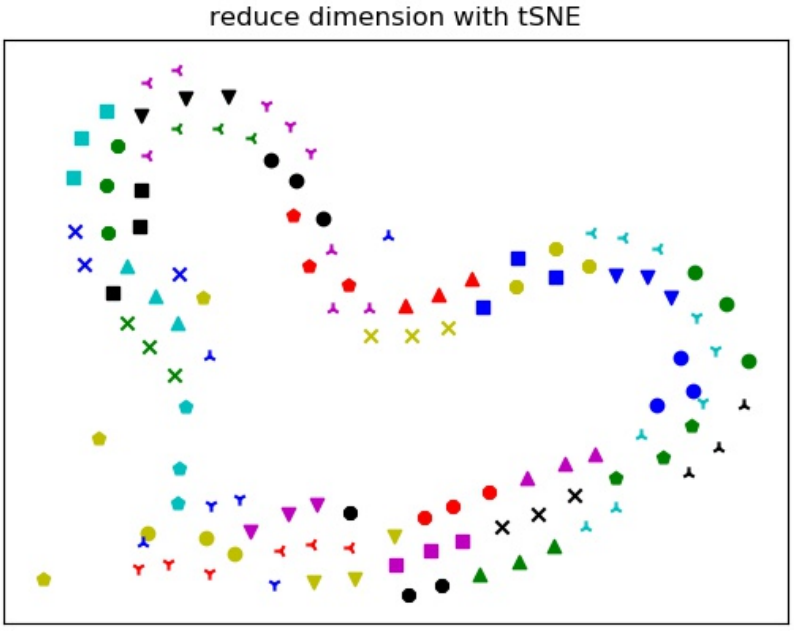
1. Perform PCA on the training data. Plot the mean face and the first five eigenfaces and show them in the report. (5%)



2. Take $\boldsymbol{person}_8\_\ \boldsymbol{image}_6$, and project it onto the above PCA eigenspace. Reconstruct this image using the first n = { 5, 50, 150, all } eigenfaces. For each n, compute the mean square error (MSE) between the reconstructed face image and the original $\boldsymbol{person}_8\_\ \boldsymbol{image}_6$. Plot these reconstructed images with the corresponding MSE values in the report. (5%)

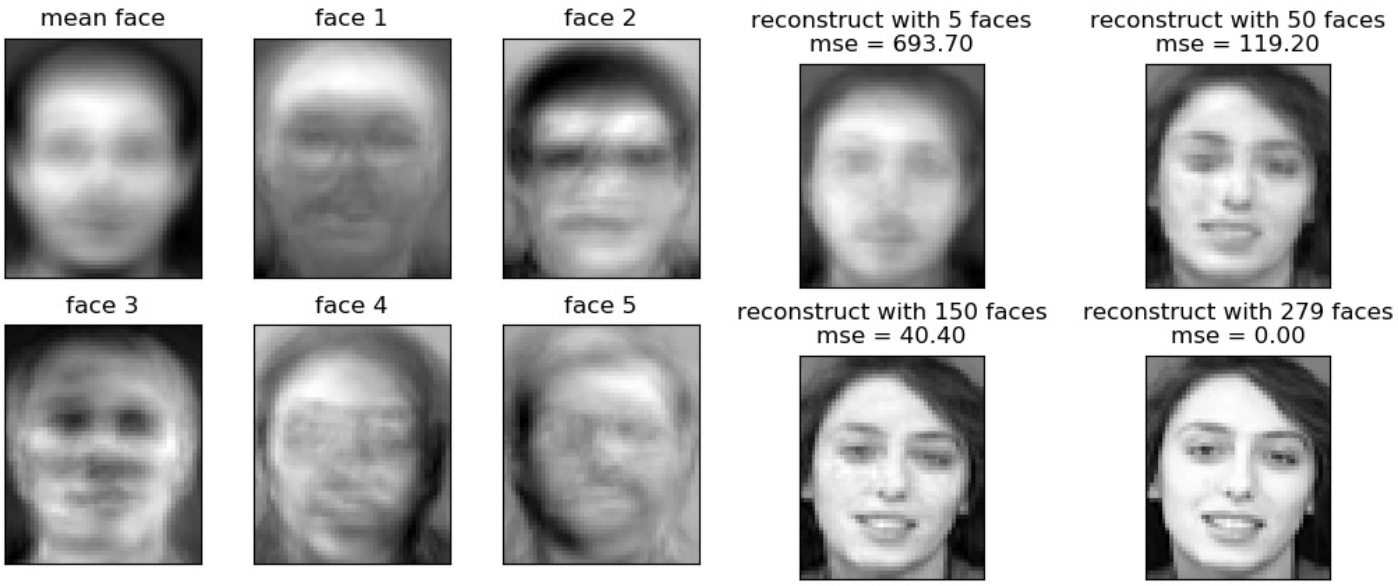


3. Reduce the dimension of the image in testing set to dim = 100. Use t-SNE to visualize the distribution of test images. (5%)



4. (bonus 5%) Implement the Gram Matrix trick for PCA. Compare the two reconstruction images from standard PCA/Gram Matrix process. If the results are different, please explain the reason. Paste the main code fragment(screenshot) on the report with discussion.

```python
def gram_pca(data):
    # flatten
    data = data.reshape(data.shape[0], -1)
    mean = np.mean(data, axis=0)
    data = data - mean # data shape(n_data, 46*56)
    # Calculate the eigen value for the gram matrix
    eigen_val, eigen_vec = np.linalg.eigh(np.dot(data, data.T))
    eigen_vec = np.dot(data.T, eigen_vec).T # eigen_vec shape(n_data, 46*56))
    for i in range(eigen_vec.shape[1]):
        eigen_vec[:, i] = eigen_vec[:, i] / np.sqrt(eigen_val)
    return mean , eigen_vec[::-1, :]
```



The reconstruction process is almost the same. The slight difference between the decomposition is the number of eigenvectors. In Gram Matrix trick for PCA, there are only $n-1$ eigenvectors due to the constraint of mean vector so that we can only use 279 face for reconstruction.

Runtime Information

PCA (`np.linalg.svd`) : 0.11090517044067383 (s)

Gram Matrix PCA : 0.09964679718017578 (s)

get nearly 10% faster than original

(b) k-NN (10%)

To apply the k-nearest neighbors (k-NN) classifier to recognize the testing set images, please determine the best k and n values by 3-fold cross-validation.

For simplicity, the choices for such hyper-parameters are:

k = {1, 3, 5} and n = {3, 10, 39}.

Please show the cross-validation results and explain your choice for (k, n). Also, show the recognition rate on the testing set using your hyper-parameter choice.

	k=1, n=3	k=3, n=3	k=5, n=3
fold1 acc	0.6500	0.4875	0.4500
fold2 acc	0.6250	0.6000	0.5125
fold3 acc	0.7000	0.5917	0.4917
	k=1, n=10	k=3, n=10	k=5, n=10
fold1 acc	0.9125	0.7000	0.6875
fold2 acc	0.9125	0.7875	0.7750
fold3 acc	0.8917	0.7917	0.6917
	k=1, n=39	k=3, n=39	k=5, n=39
fold1 acc	0.9500	0.8125	0.7375
fold2 acc	0.9250	0.8375	0.8000
fold3 acc	0.9333	0.8250	0.7750

fold1 = $person_{1\sim40}$, $image_{6\sim7}$;
fold2 = $person_{1\sim40}$, $image_{1\sim2}$;
fold3 = $person_{1\sim40}$, $image_{3\sim5}$

I chose (k=1, n=39) as my hyper-parameters, and the testing accuracy is *0.9667.*

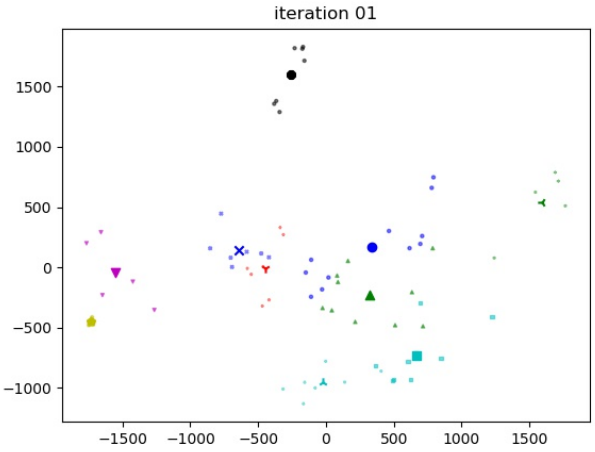
(c) K-means (10%)

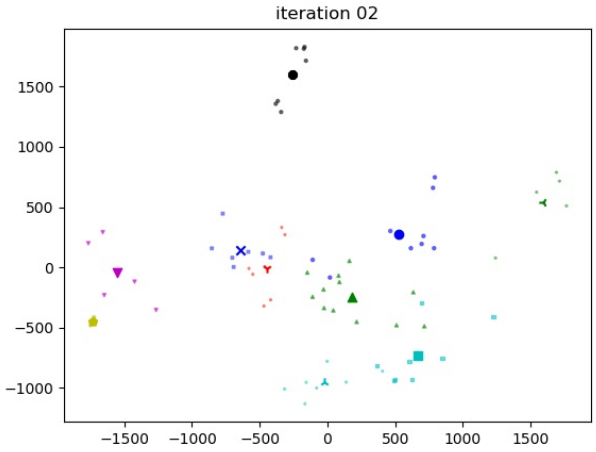
Reduce the dimension of the images in the first 10 class of training set to dim = 10 using PCA. Implement the k-means clustering method to classify the these images.

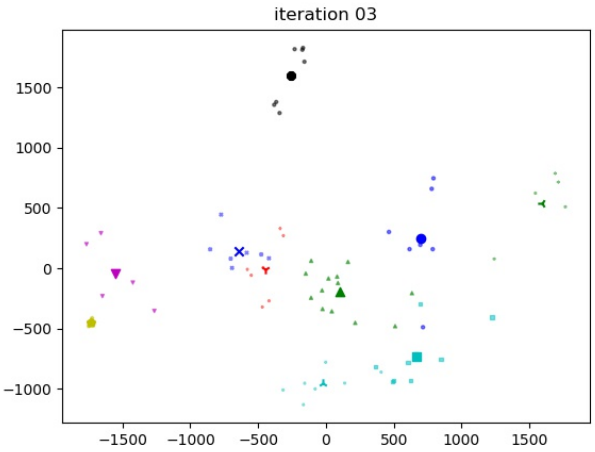
1. Please use weighted Euclidean distance to implement the k-means clustering. The weight of the best 10 eigenfaces is

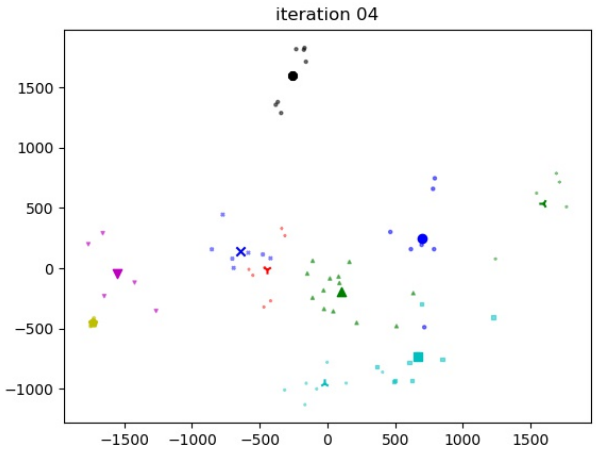
   [0.6, 0.4, 0.2, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1]

2. Please visualize the features and the centroids to 2D space with the first two eigenfaces for each iteration in k-means clustering (up to 5 images). (5%)









3. Compare the results of K-means and ground truth. (5%)

```
====================
group: 00
3_5, 5_1, 5_2, 5_3, 5_4, 5_5, 5_6, 5_7,
====================
group: 01
8_1, 8_2, 8_3, 8_4, 8_5, 8_6, 8_7,
====================
group: 02
3_1, 3_2, 3_3, 3_4, 3_6, 3_7, 4_1, 4_2, 4_3, 4_4, 4_5, 4_6, 4_7,
====================
group: 03
6_1, 6_2, 6_3, 6_4, 6_5, 6_6, 6_7,
====================
group: 04
10_3, 10_4, 10_5, 10_6, 10_7,
====================
group: 05
7_1, 7_2, 7_3, 7_4, 7_5, 7_6, 7_7,
====================
group: 06
10_1, 10_2,
====================
group: 07
9_1, 9_2, 9_3, 9_4, 9_5, 9_6, 9_7,
====================
group: 08
1_1, 1_3, 2_1, 2_2, 2_3, 2_4, 2_5, 2_6, 2_7,
====================
group: 09
1_2, 1_4, 1_5, 1_6, 1_7,
====================
```

Most of the pictures are well-grouped. However, $person_3$ is mistook with $person_4$, and some images of $person_{10}$ can't be grouped properly.
I guess that $person_3$ and $person_4$ can't be separated well because of their similar haircut. Some pictures of $person_{10}$ blur the hair parts with background and they are also tough cases for human beings.

Problem 4 (report 30% + baseline 10%)

(a) MNIST Classification (20%)

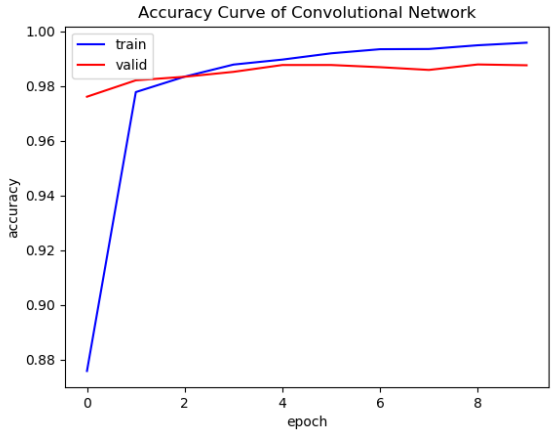
1. Build a CNN model and a Fully-Connected Network and train them on the MNIST dataset. Show the architecture of your models and the correspond parameters amount in the report. (5%)

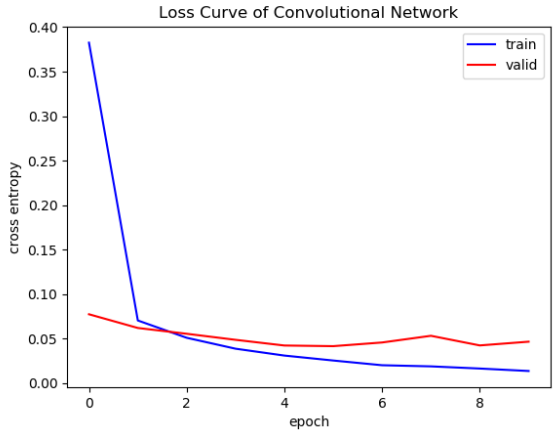
```
class ConvNet(nn.Module):
    # with 44,426 parameters
    def __init__(self):
        super(ConvNet, self).__init__()
        self.conv1 = nn.Conv2d(1, 6, kernel_size=(5, 5), stride=(1, 1))
        self.conv2 = nn.Conv2d(6, 16, kernel_size=(5, 5), stride=(1, 1))
        self.fc1 = nn.Linear(in_features=256, out_features=120, bias=True)
        self.fc2 = nn.Linear(in_features=120, out_features=84, bias=True)
        self.fc3 = nn.Linear(in_features=84, out_features=10, bias=True)
```

```
class Fully(nn.Module):
    # with 109,386 parameters
    def __init__(self):
        super(Fully, self).__init__()
        self.fc1 = nn.Linear(in_features=28*28, out_features=128, bias=True)
        self.fc2 = nn.Linear(in_features=128, out_features=64, bias=True)
        self.fc3 = nn.Linear(in_features=64, out_features=10, bias=True)
```

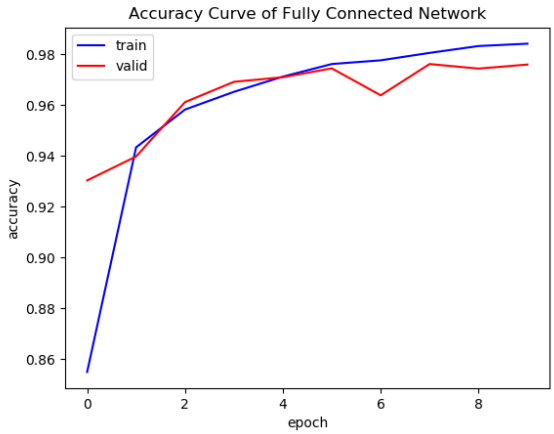
2. Report your training / validation accuracy, and plot the learning curve (loss, accuracy) of the training process. (figure 5%+ baseline 5%)

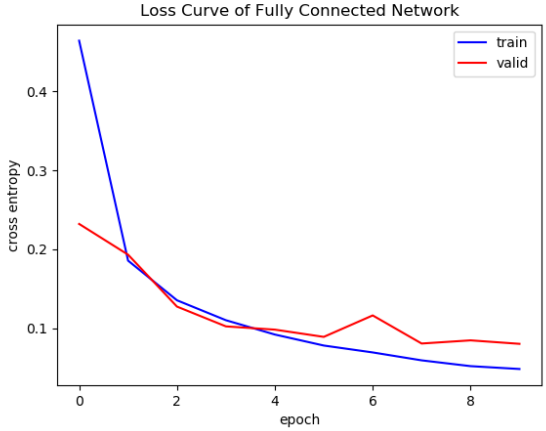
Convolution Network: acc (train / valid) = (0.994 / 0.982)





Fully Connected: acc (train / valid) = (0.984 / 0.976)





3. Compare the results of both models and explain the difference. (5%)

The performance of convolutional network is better than fully connected network with less parameters in this experiment.
Convolution is more suitable than fully connected layers on the image tasks because the output of convolution preserves the space information from the input data. Besides, convolution shares filters across different part of input, which reduce the amount of parameters compared to fully connected layer.

(b) Face Recognition (20%)

1. Extract image feature using pytorch pretrained alexnet and train a KNN classifier to perform human face recognition on the given dataset. Report the validation accuracy. (5%)

   Accuracy = 0.6212

   I extracted the 256-dimension feature from the `AlexNet`, and reduced the latent into 100-dimension by `PCA`. I trained the `kNN` classifier with `n_neighbors=1`.

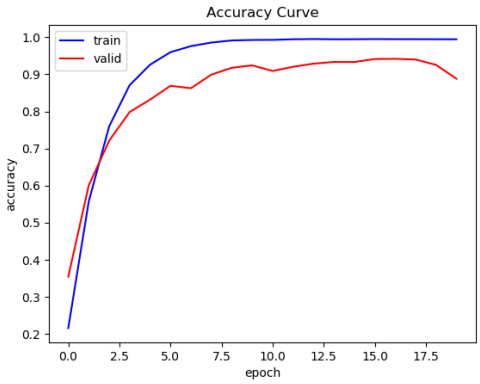
2. Build your own model and train it on the given dataset to surpass the accuracy of previous stage. Show and explain the architecture of your model and report the validation accuracy. Paste the main code fragment(screenshot). (5%)

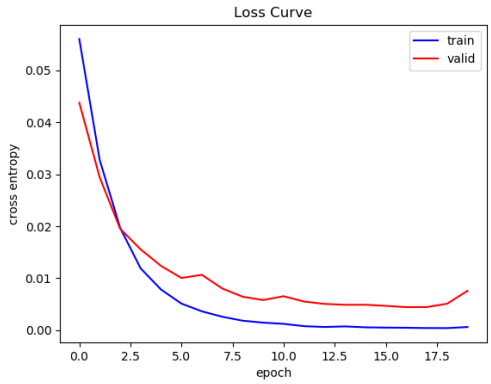
```python
class Residual_Block(nn.Module):
    def __init__(self, in_c, out_c):
        super(Residual_Block, self).__init__()
        self.feature = nn.Sequential(
            nn.Conv2d(in_c, out_c, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(out_c),
            nn.LeakyReLU(inplace=True),
            nn.Conv2d(out_c, out_c, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(out_c),
            )
        if in_c != out_c:
            self.shortcut = nn.Sequential(nn.Conv2d(in_c, out_c, kernel_size=1, padding=0, bias=False),
                                          nn.BatchNorm2d(out_c),)
        else:
            self.shortcut = nn.Sequential()

    def forward (self,x):
        residual = self.shortcut(x)
        x = self.feature(x)
        x = nn.functional.leaky_relu(residual + x, inplace=True)
        return x
```

```python
class resnet(nn.Module):
    def __init__(self, widen_factor=4):
        super(resnet, self).__init__()
        self.widen_factor = widen_factor
        self.conv_1 = nn.Conv2d(3, 64, kernel_size=3, stride=1, padding=1, bias = False)
        self.res_block1 = Residual_Block(64, 64)
        self.res_block2 = Residual_Block(64, 64)
        self.res_block3 = Residual_Block(64, 128)
        self.res_block4 = Residual_Block(128, 128)
        self.res_block5 = Residual_Block(128, 256)
        self.res_block6 = Residual_Block(256, 256)
        self.fc_1 = nn.Linear(256*3*3, 100)

    def forward(self, x):
        x = self.conv_1(x)                          #(B, 16, H, W)
        x = self.res_block1(x)
        x = F.max_pool2d(self.res_block2(x), 2)     #(B, 64, H/2, W/2)
        x = self.res_block3(x)
        x = F.max_pool2d(self.res_block4(x), 4)     #(B, 128, H/8, W/8)
        x = self.res_block5(x)
        latent = F.avg_pool2d(self.res_block6(x), 4) #(B, 256, H/32, W/32)
        x = latent.view(latent.size(0), -1)
        x = self.fc_1(x)
        return x, latent
```

I built a shallow `ResNet` by my self-defined residual block. Due to the property of `BatchNorm`, greater batch size achieved better performance on this task. The accuracy was 0.9946 / 0.9417 (train / valid).
The accuracy of `kNN classifier (n_neighbors=1)` was 0.8545.

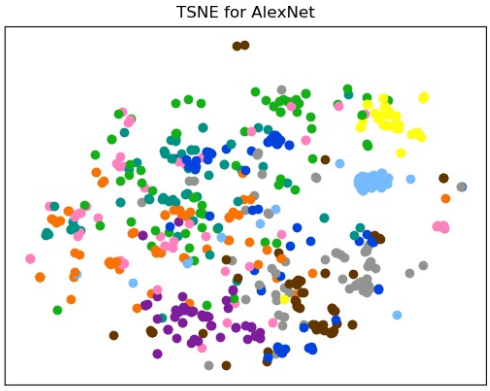
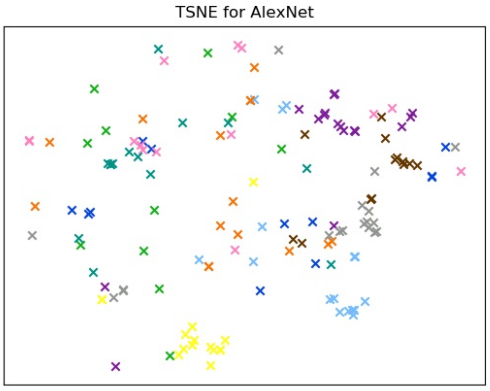
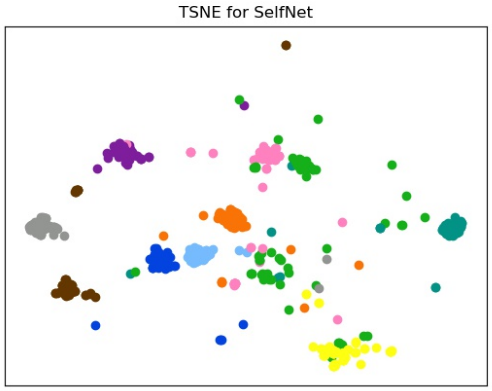
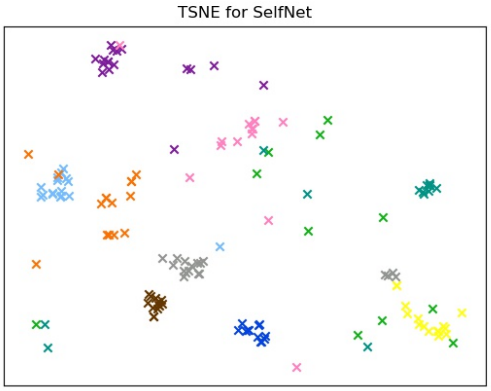
3. Plot the learning curve of your training process(training/validation loss/ accuracy) in stage 2, explain the result you observed. (5%)





The accuracy was 0.9946 / 0.9417 (train / valid). As you can see in the figures, the model is overfitting on the training set at the end of training process. I chose the model from the 16th epoch as my feature for `kNN`.

4. Pick the first 10 identities from dataset. Visualize the features of training/ validation data you extract from pretrained alexnet and your own model to 2D space with t-distributed stochastic neighbor embedding (t-SNE), compare the results and explain the difference. (5%)

feature from `AlexNet` (train)	feature from `AlexNet` (valid)
	
feature from `ResNet` (train)	feature from `ResNet` (valid)
	

The `TSNE` result from `ResNet` is grouped better than the features from AlexNet. The most popular reason is that the latent is from a deeper layer in network which can represent high-level features in computer vision.