Programming Exercise 4

Prepare your working environment

If you have installed following libraries in your python environment, you can skip this step.

```
pytorch
torchvision
numpy
matplotlib
pandas
scikit-learn
```

Otherwise, you can create a new Anaconda environment:

```
conda create -n dml python=3.6.5
conda activate dml
```

And follow the instruction of installing pytorch according to your device.

Then install other dependencies:

```
conda install numpy
conda install matplotlib
conda install pandas
conda install scikit-learn
```

Prepare your data

In this tutorial we will use the Fashion MNIST dataset which can be downloaded here. The dataset will download as a file named `fashion-mnist_train.csv`. Once downloaded, create a directory named `data` and put the csv file into that directory. The resulting directory structure should be `data/fashion-mnist_train.csv`.

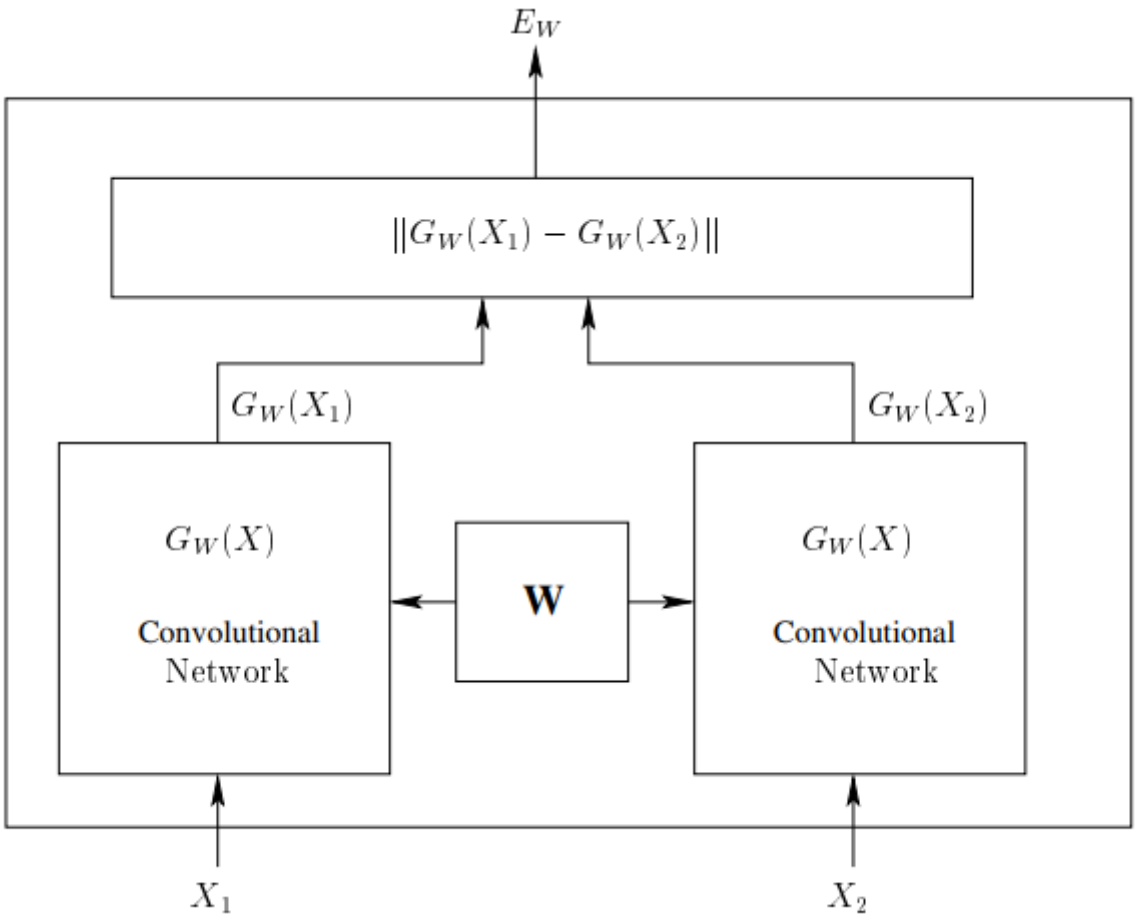
Introduction

In this tutorial, we will recreate part of the experiments from the paper "A unifying mutual information view of
metric learning: cross-entropy vs. pairwise losses" on Fashion MINIST, a small image classification dataset. Here are the main objectives:

- Implement a CNN classifier with cross entropy loss.
- Implement siamese CNN with contrastive loss, a classic deep metric learning network.
- Compare the learned feature embedding and image classification performance of both methods.

Siamese CNN & Contrastive Loss

The Siamese framework with contrastive loss as the training objective is firstly proposed in the paper: Learning a similarity metric discriminatively, with application to face verification. It comprises two identical networks (CNN in our case) with the same weights and one cost module. The input to the system is a pair of images and a label (indicating whether the two images are from the same class). The images are passed through the sub-networks, yielding two outputs which are passed to the cost module. The loss function combines the label with two model outputs and calculate a distance in the feature space for model optimization.



Contrastive loss is firstly proposed in the paper: Dimensionality Reduction by Learning an Invariant Mapping. The loss encourages the model to learn a mapping from
high to low dimensional space that maps similar input vectors to nearby points on the output manifold and dissimilar vectors to distant points. Let $\vec{X}_1$, $\vec{X}_2$ be a pair of input vectors shown to the model. Let Y be a binary label assigned to this pair. $Y = 0$ if $\vec{X}_1$ and $\vec{X}_2$ are similar, and $Y = 1$ if they are dissimilar. Define the parameterized distance function to be learned D_W between $\vec{X}_1$ and $\vec{X}_2$ as the euclidean distance between the outputs of G_W, and then we can write the actual form of contrastive loss.

$$D_W(\vec{X}_1, \vec{X}_2) = \left\| G_W(\vec{X}_1) - G_W(\vec{X}_2) \right\|_2$$

$$L_{CT}(W, Y, \vec{X}_1, \vec{X}_2) = (1 - Y)\frac{1}{2}(D_W)^2 + (Y)\frac{1}{2}\left\{max(0, m - D_W)\right\}^2$$

where $m > 0$ is a margin. The margin defines a radius around $G_W(\vec{X})$. Dissimilar pairs contribute to the loss function only if their distance is within this radius. The contrastive term involving dissimilar pairs is crucial. Simply minimizing $D_W(\vec{X}_1, \vec{X}_2)$ over the set of all similar pairs will usually lead to a collapsed solution, since D_W and the loss L could then be made zero by setting G_W to a constant.

Standard CNN for Classification & Cross Entropy Loss

A standard CNN for classification usually consists of one branch of the above siamese network. The model take one image as input and predict a vector of logits whose dimension is the same as the number of classes. Then cross entropy loss is used as an objective to train the model.

Cross Entropy (CE) loss measures the performance of a classification model whose output is a probability value between 0 and 1. CE loss increases as the predicted probability diverges from the actual label. In binary classification, where the number of classes M equals 2, Binary Cross Entropy (BCE) can be calculated as:

$$L_{BCE} = -(y \log(p) + (1 - y) \log(1 - p))$$

If $M > 2$ (i.e., multiclass classification), we calculate a separate loss for each class label per observation and sum the result:

$$L_{CE} = -\sum_{c=1}^{M} y_{o,c} \log(p_{o,c})$$

where log is the natural log, y is the binary indicator (0 or 1) if class label c is the correct classification for observation o, and p is the predicted probability observation o is of class c.

Coding Time

You can paste the following code blocks into your own python scirpt to run the experiments.

Import dependencies

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.utils.data import Dataset,DataLoader
from sklearn.model_selection import train_test_split
from sklearn import manifold
import random
import os
import time
from sklearn.neighbors import KNeighborsClassifier
from sklearn import metrics
```

Set random seed for reproducibility, set device, and set time counter.

```
start = time.time()

def set_seed(seed):
    """Set seed"""
    random.seed(seed)
    np.random.seed(seed)
    torch.manual_seed(seed)
    if torch.cuda.is_available():
        torch.cuda.manual_seed(seed)
        torch.cuda.manual_seed_all(seed)
        torch.backends.cudnn.deterministic = True
        torch.backends.cudnn.benchmark = False
    os.environ["PYTHONHASHSEED"] = str(seed)

device = 'cuda' if torch.cuda.is_available() else 'cpu'

set_seed(2022)
```

Preprocess csv data.

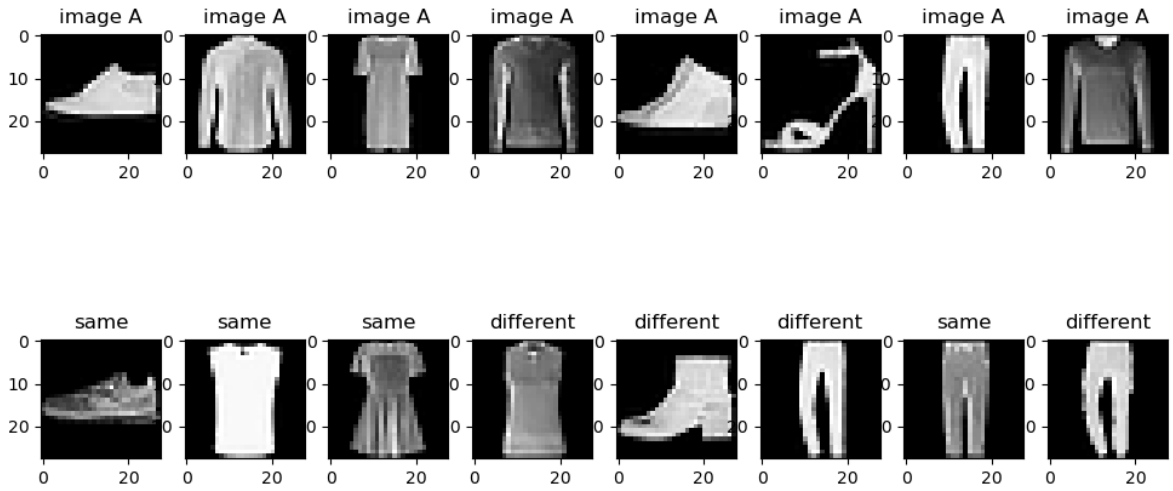
```
data_train = pd.read_csv('data/fashion-mnist_train.csv')
print(data_train.head())

X_full = data_train.iloc[:,1:]
y_full = data_train.iloc[:,:1]
x_train, x_test, y_train, y_test = train_test_split(X_full, y_full, test_size = 0.05)
x_train = x_train.values.reshape(-1, 1, 28, 28).astype('float32') / 255.
x_test = x_test.values.reshape(-1, 1, 28, 28).astype('float32') / 255.
print(y_train.label.unique())
print(np.bincount(y_train.label.values), np.bincount(y_test.label.values))
```

Create the PyTorch dataset and dataloader.

```
class mydataset(Dataset):
    def __init__(self, x_data, y_data):
        self.x_data = x_data
        self.y_data = y_data.label.values

    def __len__(self):
        return len(self.x_data)

    def __getitem__(self, idx):
        img1 = self.x_data[idx]
        y1 = self.y_data[idx]
        if np.random.rand() < 0.5:
            idx2 = np.random.choice(np.arange(len(self.y_data))[self.y_data == y1], 1)
        else:
            idx2 = np.random.choice(np.arange(len(self.y_data))[self.y_data != y1], 1)
        img2 = self.x_data[idx2[0]]
        y2 = self.y_data[idx2[0]]
        label = [0] if y1 == y2 else [1]
        return torch.FloatTensor(img1), torch.FloatTensor(img2), torch.FloatTensor(label), y1


train_dataset = mydataset(x_train, y_train)
train_dataloader = DataLoader(dataset = train_dataset, batch_size=128)
val_dataset = mydataset(x_test, y_test)
val_dataloader = DataLoader(dataset = val_dataset, batch_size=128)
```

Visualize data.

```
for idx,(img1, img2, target, _) in enumerate(train_dataloader):
    fig, axs = plt.subplots(2, img1.shape[0] - 120, figsize = (12, 6))
    for idx, (ax1, ax2) in enumerate(axs.T):
        ax1.imshow(img1[idx, 0, :, :].numpy(), cmap='gray')
        ax1.set_title('image A')
        ax2.imshow(img2[idx, 0, :, :].numpy(), cmap='gray')
        ax2.set_title('{}'.format('same' if target[idx, 0] == 0 else 'different'))
    plt.savefig('visualize_data.png')
    break
```



Define Siamese CNN

For this experiment, we implement both the siamese network and CE-based classification network using the same PyTorch class. The boolean flag `is_contrastive` determines whether the network is siamese or not. When `is_contrastive=False` , the forward layers (defined in function `forward()`) consist only of the first branch of the siameses network. In this way, this architecture can be trained with both contrastive loss and cross entropy loss.

```
class siamese(nn.Module):
    def __init__(self, is_contrastive):
        super(siamese, self).__init__()
        self.feature_net = nn.Sequential(
            nn.Conv2d(1, 8, kernel_size=3, padding=1, stride=1),
            nn.ReLU(inplace=True),
            nn.BatchNorm2d(8),
            nn.Conv2d(8, 8, kernel_size=3, padding=1, stride=1),
            nn.ReLU(inplace=True),
            nn.BatchNorm2d(8),
            nn.MaxPool2d(2),
            nn.Conv2d(8, 16, kernel_size=3, padding=1, stride=1),
            nn.ReLU(inplace=True),
            nn.BatchNorm2d(16),
            nn.Conv2d(16, 16, kernel_size=3, padding=1, stride=1),
            nn.ReLU(inplace=True),
            nn.BatchNorm2d(16),
            nn.MaxPool2d(2),
            nn.Conv2d(16, 1, kernel_size=3, padding=1, stride=1),
            nn.ReLU(inplace=True)
        )
        self.linear1 = nn.Linear(49, 49)
        self.linear2 = nn.Linear(49, 10)
        self.is_contrastive = is_contrastive

    def forward_once(self, x):
        x = self.feature_net(x)
        x = x.view(x.shape[0], -1)
        x = self.linear1(x)
        x = self.linear2(x)
        return x

    def forward(self, x1, x2):
        output1 = self.forward_once(x1)
        if not self.is_contrastive:
            # Only one forward pass is needed for computing cross entropy loss.
            self.emb1 = output1
            return output1
        else:
            # We need to perform another forward pass to get one more feature for computing contrastive
            output2 = self.forward_once(x2)
            return output1, output2
```

Define Contrastive Loss.

```
def contrastive_loss(pred1, pred2, target):
    MARGIN = 2.5
    euclidean_dis = F.pairwise_distance(pred1, pred2)
    target = target.view(-1)
    loss = (1-target) * torch.pow(euclidean_dis, 2) + target * torch.pow(torch.clamp(MARGIN - euclidean
    loss = torch.mean(loss)
    return loss
```

Define Cross Entropy Loss.

```
cross_entropy_loss = nn.CrossEntropyLoss()
```

Initialize models and optimizors.

```
model_ct = siamese(is_contrastive=True).to(device)
model_ce = siamese(is_contrastive=False).to(device)

optimizor_ct = torch.optim.Adam(model_ct.parameters(), lr=0.001)
optimizor_ce = torch.optim.Adam(model_ce.parameters(), lr=0.001)
```

Training with contrastive loss

```
print('\n-----------Training with Contrastive Loss-----------')

for e in range(5):
    history = []
    for idx, (img1, img2, target, _) in enumerate(train_dataloader):
        img1 = img1.to(device)
        img2 = img2.to(device)
        target = target.to(device)

        pred1, pred2 = model_ct(img1, img2)
        loss = contrastive_loss(pred1, pred2, target)

        optimizor_ct.zero_grad()
        loss.backward()
        optimizor_ct.step()

        loss = loss.detach().cpu().numpy()
        history.append(loss)
        train_loss = np.mean(history)

    history = []
    with torch.no_grad():
        for idx, (img1, img2, target, _) in enumerate(val_dataloader):
            img1 = img1.to(device)
            img2 = img2.to(device)
            target = target.to(device)

            pred1, pred2 = model_ct(img1, img2)
            loss = contrastive_loss(pred1, pred2, target)

            loss = loss.detach().cpu().numpy()
            history.append(loss)
            val_loss = np.mean(history)
    print(f'Epoch: {e}, train_loss: {train_loss}, val_loss: {val_loss}')

torch.save(model_ct.state_dict(), 'siamese_ct.pth')
```

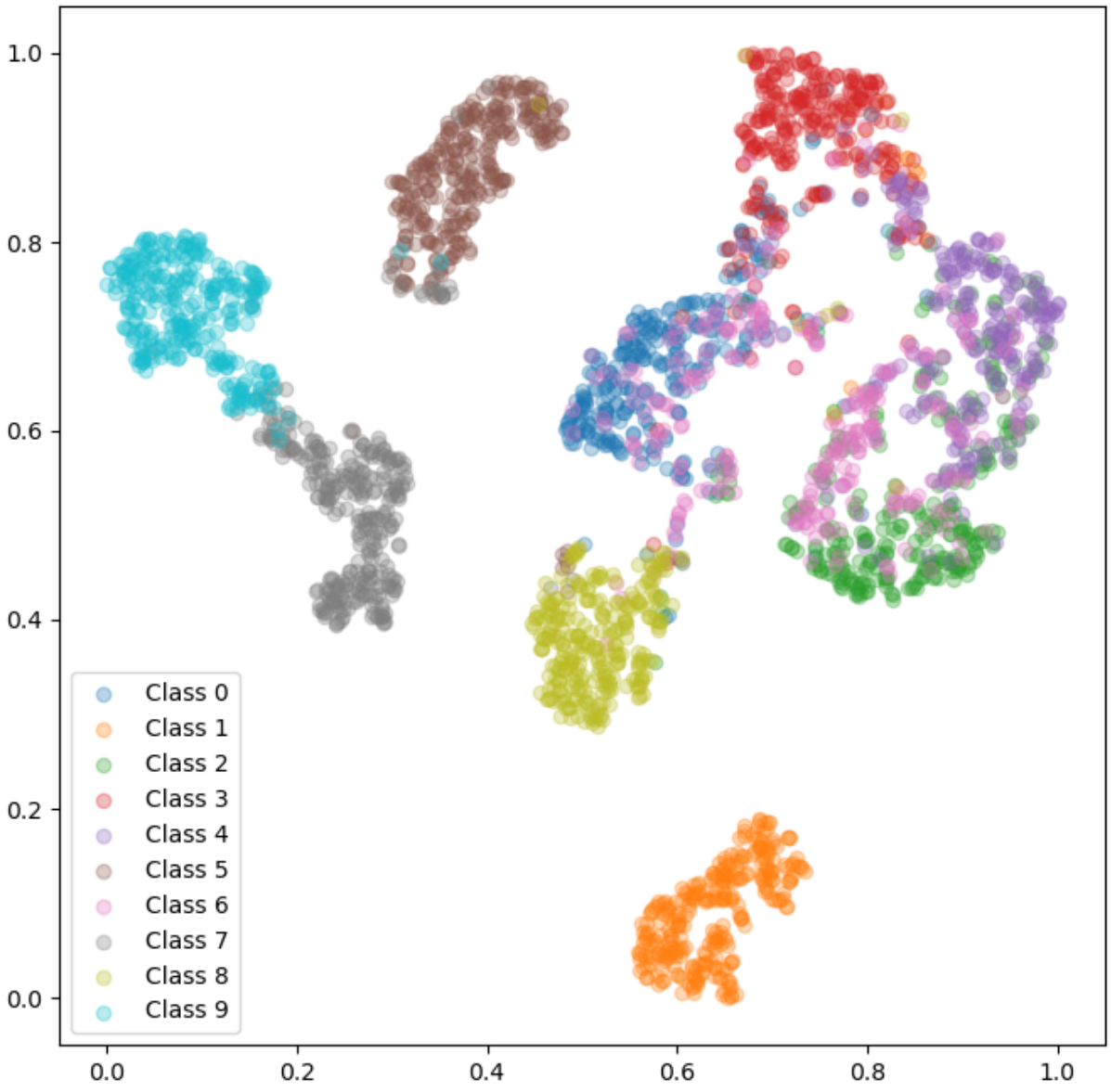
Get validation set embeddings. Only `pred1` is needed since it already can traverse the whole validation set.

```
x = []
y = []
with torch.no_grad():
    for idx, (img1, img2, target, y1) in enumerate(val_dataloader):
        img1 = img1.to(device)
        img2 = img2.to(device)

        pred1, pred2 = model_ct(img1, img2)

        x.append(pred1.detach().cpu().numpy())
        y.append(y1.detach().cpu().numpy())

X = np.concatenate(x, axis=0)
y = np.concatenate(y, axis=0)
y = y.reshape(-1)
```

Visualize Results of Contrastive Loss via t-SNE

t-distributed stochastic neighbor embedding (t-SNE) is a statistical method for visualizing high-dimensional data by giving each datapoint a location in a two or three-dimensional map. We can compare the embedding results of pairwise learning and discriminative learning through t-SNE visualization.

```
tsne = manifold.TSNE(n_components=2)
X_tsne = tsne.fit_transform(X)
print("Feature dimension is {}. \
      Dimension after reduction is {}".format(X.shape[-1], X_tsne.shape[-1]))

x_min, x_max = X_tsne.min(0), X_tsne.max(0)
X_norm = (X_tsne - x_min) / (x_max - x_min)
plt.figure(figsize=(8, 8))
for i in range(10):
    plt.scatter(X_norm[y == i][:, 0], X_norm[y == i][:, 1], alpha=0.3, label=f'Class {i}')
plt.legend()
plt.savefig('visualize_results_ct.png')
```



Get training set embeddings.

```
x_r = []
y_r = []
with torch.no_grad():
    for idx, (img1, img2, target, y1) in enumerate(train_dataloader):
        img1 = img1.to(device)
        img2 = img2.to(device)

        pred1, pred2 = model_ct(img1, img2)

        x_r.append(pred1.detach().cpu().numpy())
        y_r.append(y1.detach().cpu().numpy())

X_r = np.concatenate(x_r, axis=0)
y_r = np.concatenate(y_r, axis=0)
y_r = y_r.reshape(-1)
```

Evaluation

We will use K-Nearest Neighbors (KNN) algorithm (with Euclidean distance as metric) to evaluate the performance of our model, i.e, the learnt embeddings.

```
knn = KNeighborsClassifier(n_neighbors=1)
knn.fit(X_r, y_r)
y_pred = knn.predict(X)

def Metrics(y_real, y_pred):
    acc = metrics.accuracy_score(y_real, y_pred)
    prec = metrics.precision_score(y_real, y_pred, average='macro')
    rec = metrics.recall_score(y_real, y_pred, average='macro')
    f = metrics.f1_score(y_real, y_pred, average='macro')

    print("The average scores for all classes:")
    print("\nAccuracy:  {:.3f}%".format(acc * 100))  # (TP+TN)/Total / number of classes
    print("Precision: {:.3f}%".format(prec * 100))  # TP/(TP+FP) / number of classes
    print("Recall:    {:.3f}%".format(rec * 100))  # TP/(TP+FN) / number of classes
    print("F-measure: {:.3f}%".format(f * 100))  # 2 * (prec*rec)/(prec+rec) / number of classes

    return acc

acc = Metrics(y, y_pred)
```

Training with Cross Entropy Loss

Following the same procedures, we train the model using standard cross entropy loss in order to compare with contrastive loss.

```
print('\n-----------Training with Cross Entropy Loss-----------')

for e in range(5):
    history = []
    for idx, (img1, img2, _, y1) in enumerate(train_dataloader):
        img1 = img1.to(device)
        img2 = img2.to(device)
        lbl = torch.LongTensor(y1)
        lbl = lbl.to(device)

        pred1 = model_ce(img1, img2)
        loss = cross_entropy_loss(pred1, lbl)

        optimizor_ce.zero_grad()
        loss.backward()
        optimizor_ce.step()

        loss = loss.detach().cpu().numpy()
        history.append(loss)
        train_loss = np.mean(history)

    history = []
    with torch.no_grad():
        for idx, (img1, img2, _, y1) in enumerate(val_dataloader):
            img1 = img1.to(device)
            img2 = img2.to(device)
            lbl = torch.LongTensor(y1)
            lbl = lbl.to(device)

            pred1 = model_ce(img1, img2)
            loss = cross_entropy_loss(pred1, lbl)

            loss = loss.detach().cpu().numpy()
            history.append(loss)
            val_loss = np.mean(history)
    print(f'Epoch: {e}, train_loss: {train_loss}, val_loss: {val_loss}')

torch.save(model_ce.state_dict(), 'siamese_ce.pth')
```

Get validation set embeddings.

```
x = []
y = []
y_pred = []
with torch.no_grad():
    for idx, (img1, img2, target, y1) in enumerate(val_dataloader):
        img1 = img1.to(device)
        img2 = img2.to(device)

        out = model_ce(img1, img2)
        # _, predicted = torch.max(out.data, 1)
        pred1 = model_ce.emb1

        x.append(pred1.detach().cpu().numpy())
        y.append(y1.detach().cpu().numpy())
        # y_pred.append(predicted.detach().cpu().numpy())

X = np.concatenate(x, axis=0)
y = np.concatenate(y, axis=0)
y = y.reshape(-1)
# y_pred = np.concatenate(y_pred, axis=0)
# y_pred = y_pred.reshape(-1)
```

Visualiza results via t-SNE.

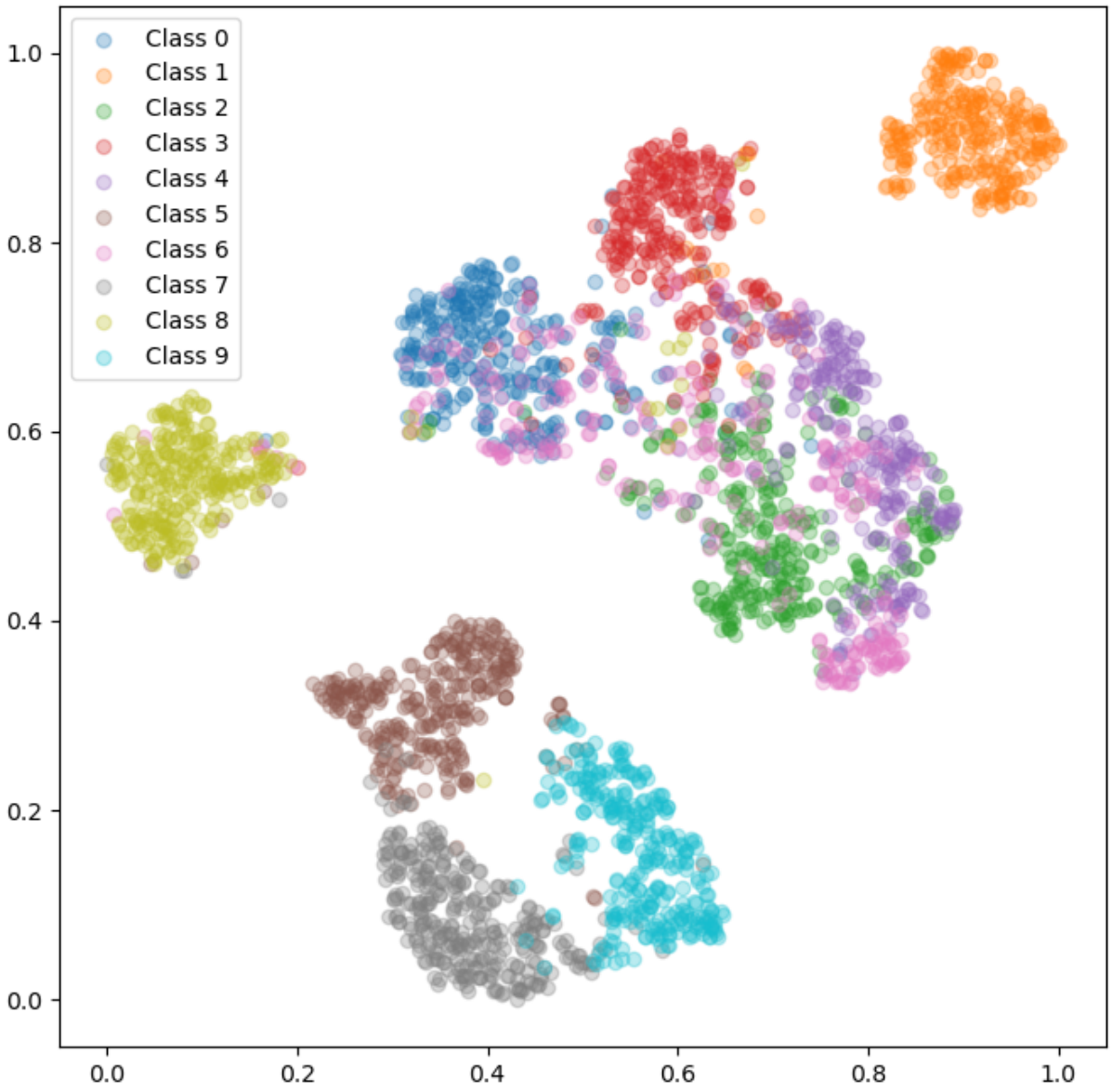
```
tsne = manifold.TSNE(n_components=2)
X_tsne = tsne.fit_transform(X)
print("Feature dimension is {}. \
      Dimension after reduction is {}".format(X.shape[-1], X_tsne.shape[-1]))

x_min, x_max = X_tsne.min(0), X_tsne.max(0)
X_norm = (X_tsne - x_min) / (x_max - x_min)
plt.figure(figsize=(8, 8))
for i in range(10):
    plt.scatter(X_norm[y == i][:, 0], X_norm[y == i][:, 1], alpha=0.3, label=f'Class {i}')
plt.legend()
plt.savefig('visualize_results_ce.png')
```

Get training set embeddings.

```
# reference
x_r = []
y_r = []
with torch.no_grad():
    for idx, (img1, img2, target, y1) in enumerate(train_dataloader):
        img1 = img1.to(device)
        img2 = img2.to(device)

        out = model_ce(img1, img2)
        pred1 = model_ce.emb1

        x_r.append(pred1.detach().cpu().numpy())
        y_r.append(y1.detach().cpu().numpy())

X_r = np.concatenate(x_r, axis=0)
y_r = np.concatenate(y_r, axis=0)
y_r = y_r.reshape(-1)
```

Evaluation via KNN.

```
knn = KNeighborsClassifier(n_neighbors=1)
knn.fit(X_r, y_r)
y_pred = knn.predict(X)

acc_ce = Metrics(y, y_pred)
```



Calculate running time of the script.

```
end = time.time()

print(f'Running Time: {(end - start)/60.0:.1f} mins')
```

Exercises

1. The reading material claim that with careful tuning, training with cross entropy loss can achieve better performance than with those pairwise losses. What do you observe from the experiments above? Do you agree with the conclusion from the reading material and why?
2. Change the sample size of training set and compare the sample efficiency of both methods.
3. Try to implement another pairwise loss and compare its result with CE and contrastive loss.
4. Report the recall at k (e.g. k = 1 to 5) performance and draw a plot.
   (Hint: In our setting, the knn classifier can output estimated probabilities of each test sample with respect to each of the 10 classes by using `knn.predict_proba()` function. Take recall at 5 for an example, let's say k = 5 and the ground truth label of a test sample is 1, if the top 5 highest predicted probabilities by the KNN corresponds to a list of predicted classes [0, 2, 3, 5, 1] which includes the ground truth label 1, the we say this prediction is correct. Then for all test samples, we apply the same procedures and finally calculate recall at 5.)

Reference

1. Learning a similarity metric discriminatively, with application to face verification.
2. Dimensionality Reduction by Learning an Invariant Mapping.
3. A unifying mutual information view of metric learning: cross-entropy vs. pairwise losses.