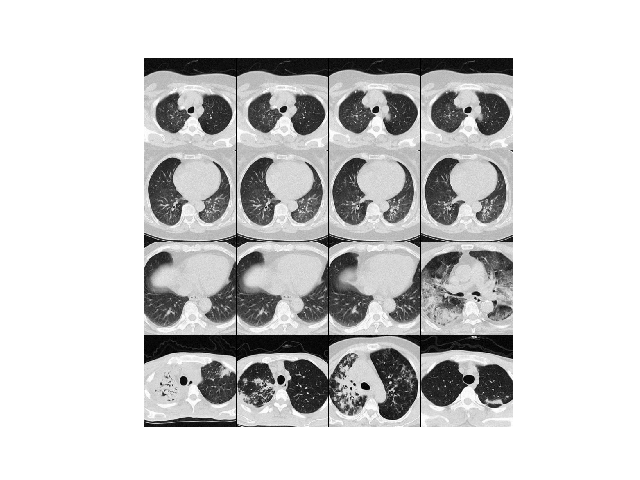
**Please keep it, as the dataset is not public.**

**Part 1. Retrieval (4p)**

n\_queries = 4  
xp\_idx0 = np.random.choice(np.where(labels == 0)[0])  
xp\_idx1 = np.random.choice(np.where(labels == 1)[0])  
xp\_idx2 = np.random.choice(np.where(labels == 2)[0])  
xp\_idx3 = np.random.choice(np.where(labels == 3)[0])  
  
queries = [xp\_idx0, xp\_idx1, xp\_idx2, xp\_idx3]  
queries\_metric = metric[queries, :]  
  
euc\_dist = torch.cdist(queries\_metric, metric, p=2)  
cloeset\_50\_value, cloeset\_50\_pos = torch.topk(euc\_dist, 28, dim=1, largest=False, sorted=True, out=None)

plt\_Compare = plt  
plt\_Compare.clf()  
f, axarr = plt\_Compare.subplots(n\_queries, 1, gridspec\_kw={'wspace': 0, 'hspace': 0})  
for i in range(n\_queries):  
 ax = axarr[i]  
 # img\_t = torch.cat((anchors[queries[i], :, :, :].unsqueeze(0), anchors[cloeset\_50\_pos[i, :], :, :, :]))  
 img\_t = anchors[cloeset\_50\_pos[i, 0:4], :, :, :]  
  
 plot\_out = torchvision.utils.make\_grid(img\_t, nrow=4, normalize=True, padding=1)  
 image\_transposed = plot\_out.numpy().transpose((1, 2, 0))  
 ax.imshow(image\_transposed)  
 ax.axis('off')  
plt\_Compare.savefig(args.experiment\_name + '\_Comparring.png')  
f.show()

But I failed to draw this figure with heatmaps.

**Calculating the accuracy, precision, recall and mAP:**

predict\_label = torch.cat(([labels[cloeset\_50\_pos[0, 1:]], labels[cloeset\_50\_pos[1, 1:]], labels[cloeset\_50\_pos[2, 1:]], labels[cloeset\_50\_pos[3, 1:]]]), 0)  
labels = th\_delete(labels, queries)  
torch.sum(predict\_label == labels)  
accuracy\_rate = torch.sum(predict\_label == labels)/len(labels)

### Part 2. Metric Learning (6p)

I used the resnet18 as the base network.

**Choose the proper sequence of dataset:**

x = self.inputs[idx]  
t = self.targets[idx]  
xp\_idx = np.random.choice(np.where(self.targets == t)[0])  
xn\_idx = np.random.choice(np.where(self.targets != t)[0])  
xp = self.inputs[xp\_idx]  
xn = self.inputs[xn\_idx]  
return x, xp, xn, t

**Get the output of three images sets -> anchors, positives and negatives:**

anchors = anchors.to(device)  
positives = positives.to(device)  
negatives = negatives.to(device)  
  
anc\_metric = model(anchors)  
pos\_metric = model(positives)  
neg\_metric = model(negatives)  
  
loss = criterion(anc\_metric, pos\_metric, neg\_metric)

**Define the criterion as Tripletloss, and the margin value:**

class TripletLoss(nn.Module):  
 def \_\_init\_\_(self, margin=0.2):  
 super(TripletLoss, self).\_\_init\_\_()  
 self.margin = margin  
  
 def forward(self, a, p, n):  
 loss = torch.norm(a - p, p=2, dim=1) - torch.norm(a - n, p=2, dim=1) + self.margin  
 loss = F.relu(loss).mean()  
 return loss

**Plot the t-SNE, confusion matrix and roc curve from test.py script:**

