Image Similarity using Deep Ranking Project Report (IE 534/CS 598 Homework 5)

Name: Xinyan Yang NetID: xinyany2

1. Your code: Please see the attached file.

2. Report your accuracy

After 9 epochs, the test accuracy is 61.04%.

3. Describe your implementation

I have implemented this model using pytorch library. There are 3 major parts in the implementation:

- Triplet Sampling layer: Generation of the triplets.
- Feed the triplet data into the network.
- Define the loss function. This loss function would take the output embedding of the deep ranking model.

For the first part, I use the following method as the author of the *Medium* article suggested: Instead of sampling the triplets on the go, I output the triplet into a text file where each line will contain the triplet in the order of query image, positive image and the negative image.

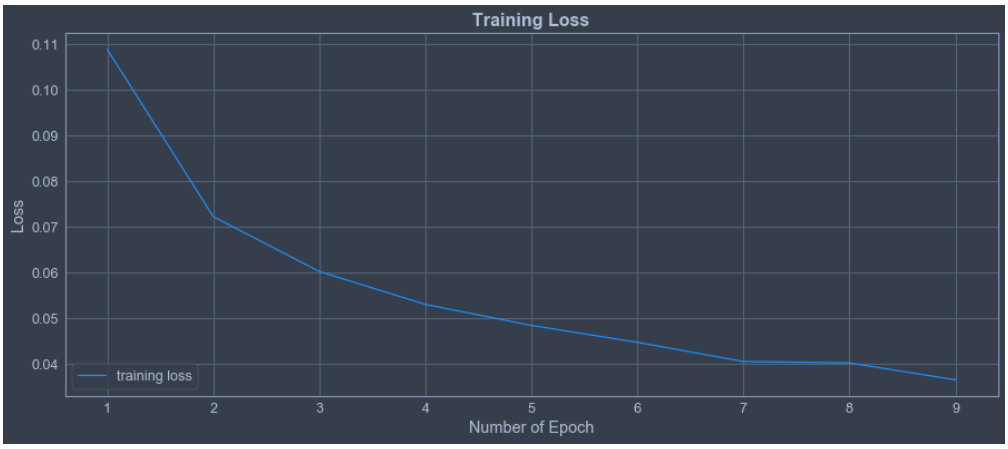
For the network architecture part, I use the Resnet50 as the convolutional model. At the fc layer, the dimension of the output features is 4096. Dropout with probability 0.6 is performed after that. Then I pass the convnet model into the Tripletnet which is a class object helps getting the feature embeddings of the triplet data.

For the loss function, I use the torch.nn.TripletMarginLoss which can directly create a criterion that measures the triplet loss given an input tensors x1, x2, x3 and a margin with a value greater than 0. This is used for measuring a relative similarity between samples.

In terms of the parameters, I use the SGD optimizer with learning rate equals to 0.001.

4. Quantitative results

- Show a plot of your training loss



- Include a table of similarity precision for both your training and test

  Train accuracy after 9 epochs: 63.25%

  Test accuracy after 9 epochs: 61.04%

5. Example of Query Results

Test image 1 tensor(0, device='cuda:0'): val_0.JPEG True label: n03444034

Top 10 ranked results along with their distances:

Distances: tensor([11.9521, 12.1200, 12.1633, 12.1710, 12.1825, 12.1919, 12.2410, 12.2813, 12.3324, 12.3605], device='cuda:0')

Paths to image: tensor([2106662307, 3814639314, 344403427, 406747267, 3444034118, 344403476, 3444034235, 3444034371, 2988304438, 3444034442], device='cuda:0')

Bottom 10 ranked results along with their distances:

Distances: tensor([29.1176, 28.0352, 27.1965, 26.5399, 26.3677, 26.3282, 26.2948, 25.6949, 25.6075, 25.5660], device='cuda:0')

Paths to image: tensor([2948072208, 2948072290, 7920052128, 381463924, 3544143295, 4456115316, 354414326, 354414364, 3544143264, 7715103257], device='cuda:0')

Test image 2 tensor(1, device='cuda:0'): val_1.JPEG True label: n04067472

Top 10 ranked results along with their distances:

Distances: tensor([11.5741, 11.9125, 11.9690, 12.1080, 12.1254, 12.1678, 12.2069, 12.2138, 12.2383, 12.2794], device='cuda:0')

Paths to image: tensor([3804744215, 9428293325, 3804744196, 436636774, 43663679, 3838899175, 4328186110, 4417672144, 284131555, 3424325261], device='cuda:0')

Bottom 10 ranked results along with their distances:

Distances: tensor([28.9235, 27.8497, 26.8350, 26.5663, 26.1584, 25.7323, 25.5902, 25.4655, 25.1713, 24.9658], device='cuda:0')

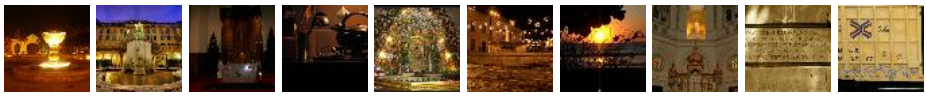
Paths to image: tensor([2948072208, 2948072290, 381463924, 7920052128, 4456115316, 3544143295, 354414326, 354414364, 3544143264, 4456115350], device='cuda:0')

Test image 3 tensor(2, device='cuda:0'), val_2.JPEG True label: n04070727

Top 10 ranked results along with their distances:

Distances: tensor([10.3919, 10.4663, 10.6796, 10.9117, 10.9459, 10.9558, 10.9806, 11.0134, 11.0604, 11.0864], device='cuda:0')

Paths to image: tensor([3388043355, 3388043443, 2699494483, 4597913468, 453210622, 3388043281,
942829360, 2699494182, 2892201354, 3393912232], device='cuda:0')



Bottom 10 ranked results along with their distances:

Distances: tensor([26.3849, 25.3916, 24.5291, 24.2252, 24.2009, 24.1391, 24.1024, 24.0671, 24.0340, 23.9614],
device='cuda:0')

Paths to image: tensor([381463924, 7920052128, 228140687, 2281406122, 2948072208, 2281406153,
2281406125, 2281406242, 2281406348, 4596742330], device='cuda:0')

Test image 4 tensor(3, device='cuda:0'), val_3.JPEG True label: n02808440

Top 10 ranked results along with their distances:

Distances: tensor([8.7431, 8.8800, 9.5022, 9.6354, 9.7613, 9.7678, 9.7962, 9.7988, 10.0083, 10.0317],
device='cuda:0')

Paths to image: tensor([280844061, 2808440365, 280844021, 2808440489, 2808440117, 2808440390,
2808440274, 2808440406, 2808440333, 2808440261], device='cuda:0')

Bottom 10 ranked results along with their distances:

Distances: tensor([28.8025, 27.7905, 27.1915, 27.0040, 26.5592, 26.4674, 26.3916, 26.3191, 25.9538, 25.9392],
device='cuda:0')

Paths to image: tensor([2948072208, 2948072290, 381463924, 7920052128, 354414326, 4259630128,
3544143264, 4456115316, 354414364, 3544143295], device='cuda:0')

Test image 5 tensor(5, device='cuda:0'), val_5.JPEG True label: n04399382

Top 10 ranked results along with their distances:

Distances: tensor([9.9989, 10.0392, 10.1045, 10.4005, 10.6391, 10.7119, 10.7388, 10.7740, 10.7810, 10.7822], device='cuda:0')]

Paths to image: tensor([439938297, 4399382454, 4399382101, 4399382346, 4399382393, 439938238, 4399382450, 4399382219, 4399382496, 4399382326], device='cuda:0')]

Bottom 10 ranked results along with their distances:

Distances: tensor([29.6265, 28.6166, 27.6417, 27.1355, 26.6626, 26.3702, 26.3134, 26.2934, 26.1022, 26.0856], device='cuda:0')]

Paths to image: tensor([2948072208, 2948072290, 4456115316, 381463924, 4456115350, 7920052128, 4532670483, 4456115474, 3599486210, 2948072320], device='cuda:0')]

6. Improvement

The improvement can be done by fining the sampling strategy. Since we are interested in the figures that most "relevant" to our query. We need to define a relevance score, as suggested by the paper to ensure we can get more positive samples for the given query image.

For every picture pi , it has a class label ci . We define the relevance for pi and pj as

$$r_{i,j} = \begin{cases} \text{some pre-calculated number}, & c_i = c_j, i \neq j \\ 0, & c_i \neq c_j, i \neq j \end{cases}$$

Then we sample the positive sample p+ for pi with following probability from the category ci

$$P(p_i^+) \propto \min\{T_i, r_{i,i^+}\},$$

where Ti is a non-negative threshold.

Also, there are two types of negative samples, in-class-negative and out-class-negative .

For out-class-negative p− , just sample uniformly at random while for in-class-negative , using

$$P(p^-) \propto \min\{T_i, r_{i,i^-}\}$$