IE 534 - Homework 5

Jacob Heglund - jheglun2

October 22, 2018

1 Implementation

My implementation is a mess. It sort of works, but god have mercy on anyone who tries to piece together the 6 (or so) python files I wrote to get all the required results for this homework. It took 35 hours to get a sort-of working implementation of the code, and I feel this assignment would have been much better suited as project with a two or three week long timeline. Due to the limited time we were given, I was only able to run my network for 14 epochs on ResNet18, but my code could easily (i.e. by changing 2 lines) be modified to run a more complicated model for more epochs. My code is set up to run for as many epochs required to achieve the 50 percent accuracy for this homework, but there wasn't enough time :(The accuracy I calculated was 8 percent, and this is shown up in the table 2.

The file training.py has all the stuff for training. I used a modified ResNet18 during my only substantial training run. The network has its fully connected layer removed, so we can extract the feature maps from an image. It also saves a checkpoint of the network, the extracted feature maps, and a list of all the query image classes and file-paths after every epoch. With this smaller network, I was able to complete 14 epochs, with each epoch taking 1.5 hours. The loss is included in Figure 1.

Validation.py has everything needed for getting accuracy from an epoch. The files are set up so that the data after an epoch is saved to disk, which validation.py could then load and be run in parallel. The numbers in the table below are the results of this file. My code is also basically set up to get the training accuracies for each epoch, but the computation would have gone beyond the deadline for this homework.

There are two versions of the dataloader for training and for getting results, and this is reflected in the file names. Writing a program to generate data was probably the most time-consuming part of this project. I implemented three different data generation schemes before settling for an online generator for each epoch. This can be found in tiny_imagenet_loader.py. Despite the file name, the program tiny_imagenet_triplet_generator.py does not generate triplets, but instead is used to give each training image a unique index for the dataloader. This index is then used by the getitem function, which generates a positive and negative image, then returns the triplet of the query, positive, and negative images.

2 Improvement

Describe at least one way in how you can improve the performance of your network

To improve my image comparison system's performance, I could train it for more epochs or use a deeper network (ResNet34, 50, 101, etc.). I could also do offline triplet sampling at the beginning of an epoch to decrease training time. A better sampling scheme could also be used. We used uniform sampling of images in the same class for positive images, and uniform sampling of negative classes and images for negative images in our implementation. However, the paper "Learning Fine-grained Image Similarity with Deep Ranking" used a more complicated triplet sampling scheme that improved performance. The paper describes that by choosing the positive image from the database based on a "relevance score", a better set of features can be learned with the triplet loss. For a negative example, they use an image in the same class, but of less relevance to the query than the positive image. This allows top results of the search to be even closer to the query. They also use a memory-efficient sampling scheme that

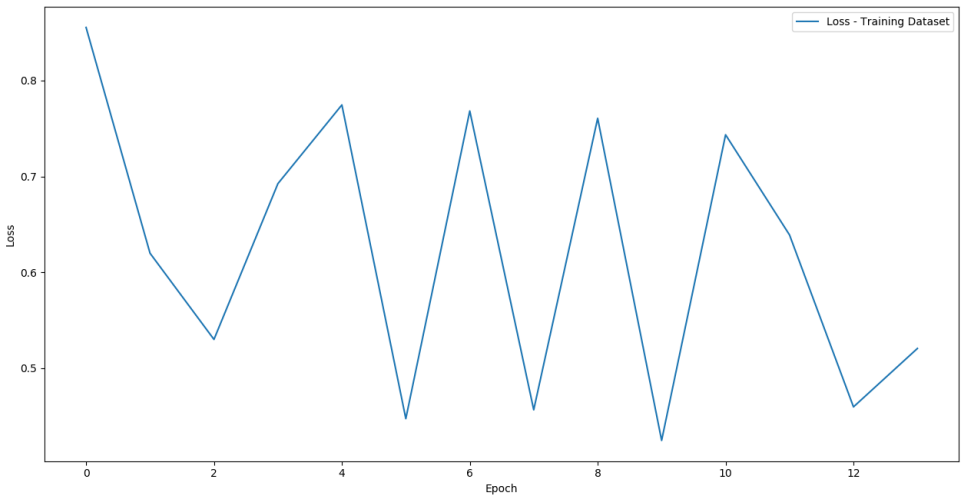

Figure 1: A nice monotonicly-ish decreasing training loss. This number is simply the final loss calculated in an epoch, and is not averaged. I'm not sure if that's the right way to do things, but I haven't read anything to the contrary.

Epoch	Validation Accuracy (percent)
0	.8
1	1
2	1.8
3	2.0
4	2.4
5	2.8
6	3.8
7	4.4
8	4.9
9	5.5
10	6.3
11	6.6
12	6.9
13	8.0

Table 1: Accuracy-by-epoch for ResNet18 for a 14 epoch training run

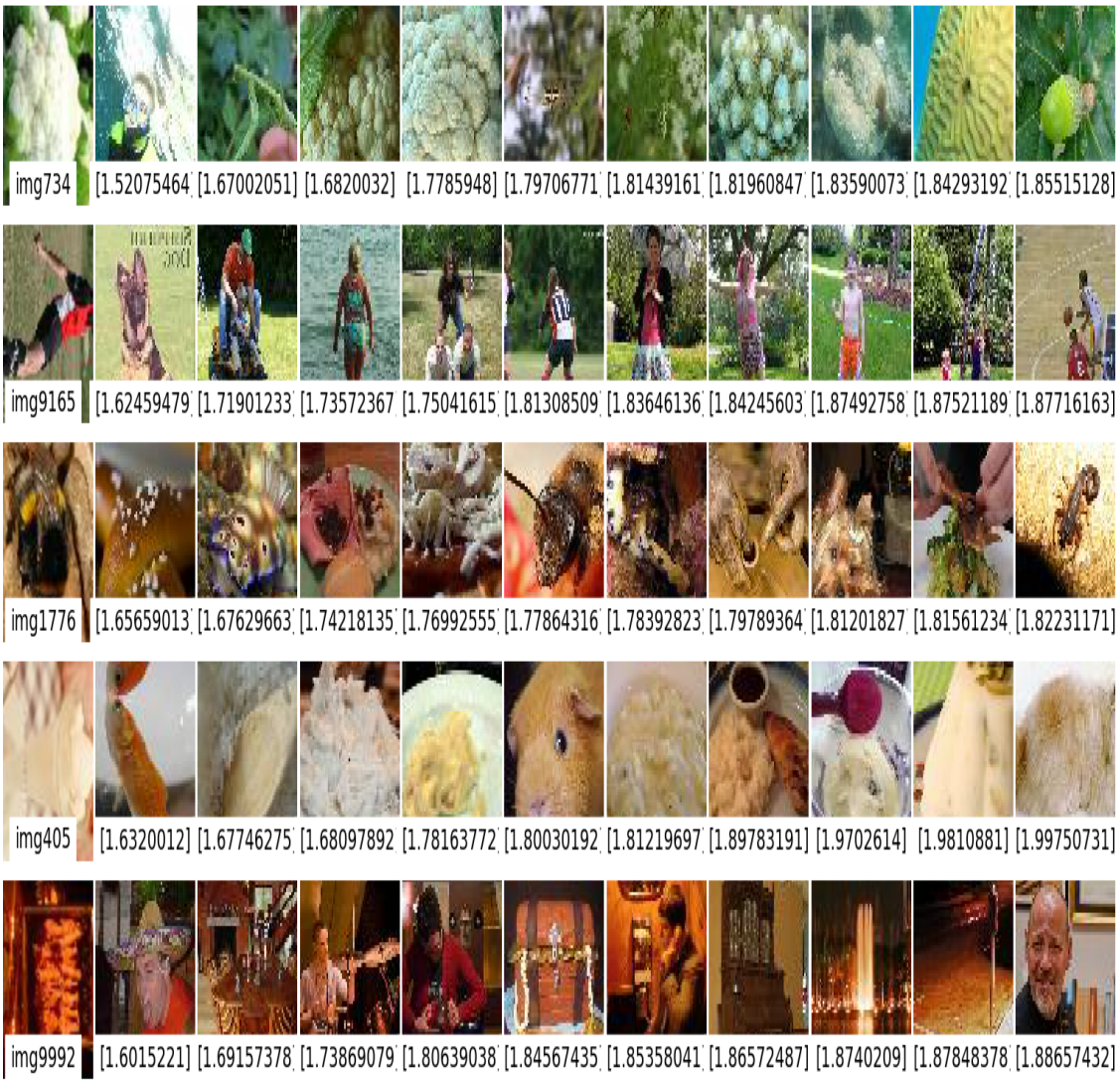

Figure 2: These are the nearest 10 images to the query image in the left-most column. The Euclidean distance between the query and selected images are also shown.

Figure 3: These are the furthest 10 images to the query image in the left-most column. The Euclidean distance between the query and selected images are also shown. Apparently ambulances don't look that similar to about anything!