Image Similarity using Deep Ranking

Arkin, Dharawat Priyanka Balakumar Satwik Singh

May 5, 2019

1 Introduction

Finding images that are similar to a query image, is an indispensable function for modern image search engines. An effective image similarity metric is at the core of finding similar images. In this report we detail the process of learning image similarity using Deep Ranking. For the intents and purpose of this project we consider categorical similarity of images and not fine-grained similarity of images i.e: for a query that is an image of a 'black car', the retrieved image has to be a 'car'(of the same class). However in case of fine-grained similarity as detailed in [1] the retrieved image has to be a 'black car'.

2 Overview

Since we are considering only categorical similarity of images i.e: images that belong to the same class we need to define our model, our loss and our ranking with respect to that. To do this we consider a pairwise ranking model. To train the model we will use triplets that consist of a query image, an in-class image or positive image and an out-class image or negative image. We do not consider in-class negative images since we are only training for categorical similarity.

3 Similarity in Images

Since we are considering a pairwise ranking model we need to mathematically define the similarity between 2 images, x and y . We define this as:

$$D(f(x), f(y)) = \|f(x) - f(y)\|_2^2$$

Where $f(.)$ is a function that takes in an image and calculates a linear embedding in an R^d and $D(.,.)$ is the euclidean distance between 2 points.

Based on this, given a triplet (q, p, n) where we have query, positive and negative image respectively we say that the triplet is correctly ranked if:

$$D(f(q), f(p)) < D(f(q), f(n)) \quad \forall (q, p, n)$$

In other words, the distance between the negative embedding and query embedding should be more than the distance between positive embedding and query embedding.

4 Network Architecture

Our entire network architecture can be divided into three parts:(1) triplet sampling layer, (2) deep network and (3) a ranking layer. We adopt the architecture in [1] but at each layer we introduce some modifications.

4.1 Triplet Sampling

Triplet sampling is important since this determines what data gets fed into the network and what it gets trained on. In [1] they used an online triplet sampling layer, however due to resource and time constraints we pre-sampled all our triplets.

We used a uniform sampling algorithm, for every query image we considered n in-class images and n out-class images. We test models with $n = 1$ and $n = 2$. Also note that the dataset with $n = 2$ has 4 times the samples present in $n = 1$.

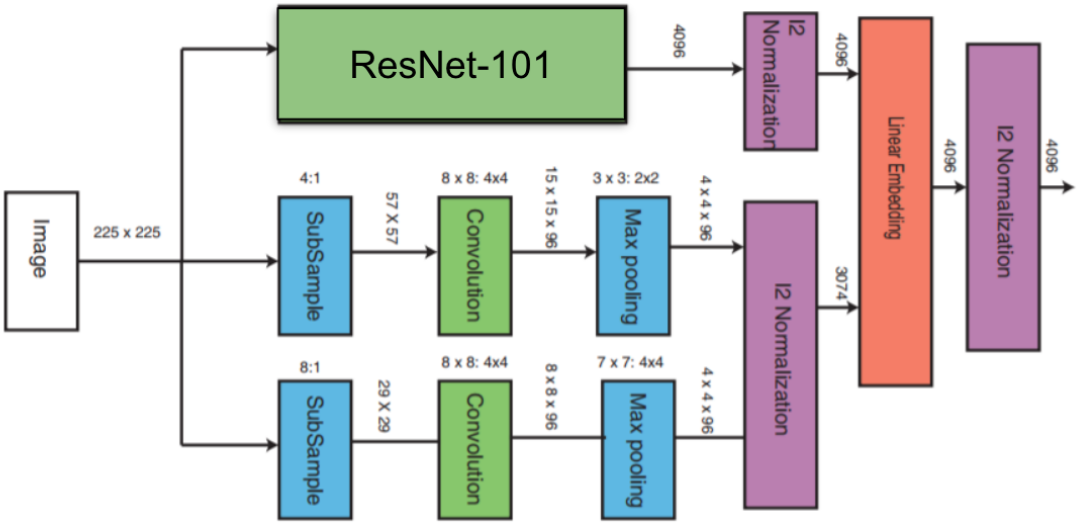
4.2 ConvNet and Sub-sampling layer

Every image has to go through three paths: a ConvNet layer and 2 sub-sampling layers.

The ConvNet has the same architecture as a ResNet neural network[2]. We decided to use a ResNet here since a ResNet can extract the same local features that a convolutional neural net does. To add to this, the residual blocks present in the network are easier to optimize, and gain significant accuracy from increased depth. For this specific training we used a ResNet-101 architecture with the final classification layer replaced by a linear layer.

The two sub-sampling layers take down-sampled images and use shallower network architectures. These two parts have less in-variance and are responsible for capturing the visual appearance.

Finally, we normalize the embeddings from the three parts, and combine them with a linear embedding layer. The dimension of our final embedding is 4096.



An important difference to note here is that the paper considers three multi-scale neural networks (one for q, one for p and one for n) and trains them in parallel. This will be very memory expensive and probably slow, so as this resource pointed out [3] it is possible to train a single network where the images of a triplet are passed in sequentially and acheive nearly identical results.

4.3 Ranking layer

The loss of our ranking layer can be defined as follows:

$$l(q, p, n) = max\{0, g + D(f(q), f(p)) - D(f(q), f(n))\}$$

A triplet $(q, p, n))$ characterizes the relative similarity of images. The output of our loss function l is 0 or non-zero based on if the query image q is closer to the positive image p or negative image n.
The variable g is a gap parameter used to regularize the distance between query-positive images

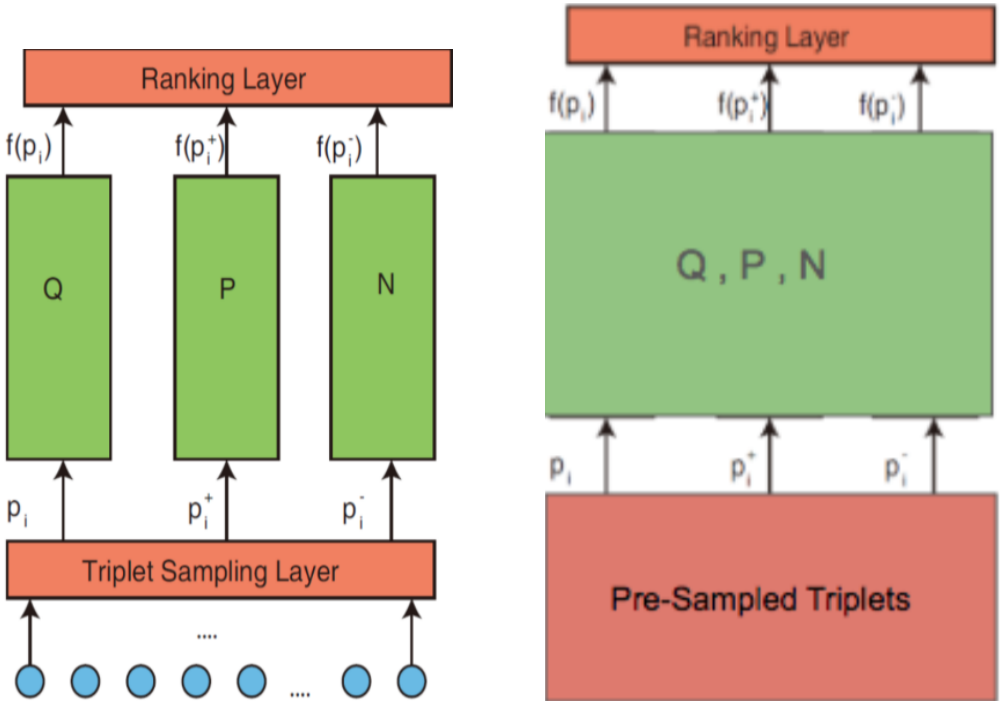
and distance between query-negative images.
Our final objective function can be written as:

$$\min_{x} \sum_{i} \xi_i + \lambda \|W\|_2^2$$
$$\text{s.t } max\{0, g + D(f(q), f(p)) - D(f(q), f(n))\} \leq \xi_i$$
$$\forall (q, p, n) \in T$$

where T is the set of pre-sampled triplets. This all is implemented in PyTorch as nn.TripletMarginLoss.

4.4 Visual Overview

This is the difference between architecture in [1] and our final modified architecture:



(a) paper architecture (b) modified architecture

Figure 1: network architectures

5 Training and Optimization

After obtaining our dataset of triplets T with $|T| \geq 100,000$ we had to train this over a modified ResNet-101 and sub-sampling layer. To make our training easier we obtained pre-trained weights for ResNet-101. We then trained over our newly-added forward layer of the ResNet and sub-sampling layers. We also experimented with training more and more layers in the ResNet.

As for an optimizer we decided to use mini-batch gradient descent with nestrov's accelerated gradient since this helps it converge faster than traditional alternatives. We used $\alpha = 0.001$ and $momemtum = 0.9$. We also experimented with an optimizer like RMSprop with $\alpha = 0.001$.
Back-propagation was used to propagate the gradients from the ranking loss to the rest of the layers.

6 Evaluation

For evaluating the embeddings produced by our model we used the following methods:

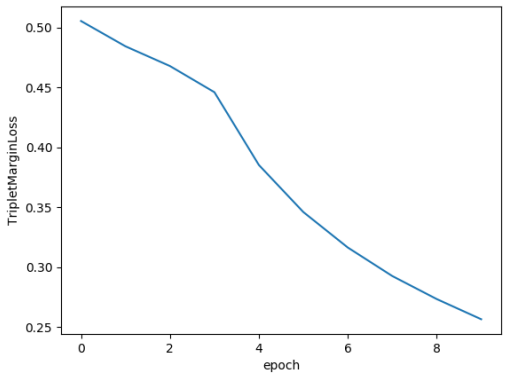
1. *similarity precision*: Similarity precision is defined as the number of triplets ranked correctly i.e: a triplet (q, p, n) is ranked correctly if $D(f(q), f(n)) > D(f(q), f(p))$ or $l(q, p, n) = 0$. We count the total number of such triplets in the train and test set.

2. *score-at-top-k*: Instead of considering the traditional definition of score-at-top-k used in IR literature, We decided to consider a more lenient approach. We trained a kNN and for every test-image q_i checked its k nearest neighbours. If at-least one of its neighbours belonged to the same class then we considered this ranking as correct.
kNN-accuracy $= \sum_{q_i}(1 \text{ if } q_i \in knn(q_i) \text{ else } 0)$.

We obtained the following results for different experiments:

Num of pos/neg images per query	Model & Optimizer	Training time	% of correctly ranked triplets in train	% of correctly ranked triplets in test	kNN accuracy(k=30)
1	forward-layer of ResNet-101 + subsampling with SGD+momentum	12 hrs	59%	61%	43%
1	forward-layer of ResNet-101 + subsampling with SGD+momentum	36 hrs	58%	62%	54%
2	conv5_x & forward-layer of ResNet-101 + subsampling with RMSprop	36 hrs	69.4%	66.7%	61.7%
2	conv5_x & forward-layer of ResNet-101 + subsampling with SGD+momentum	36 hrs	75%	70.95%	73.6%

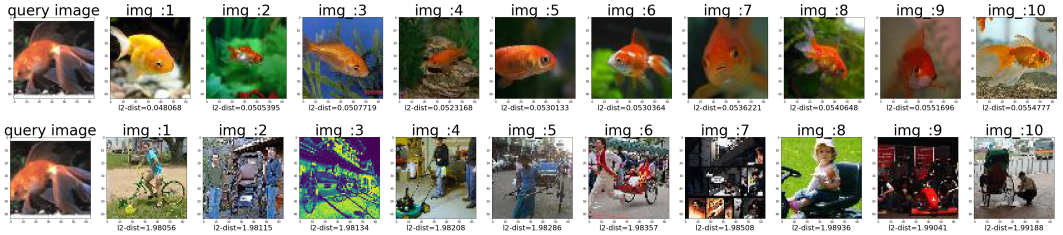
For our architecture where we trained conv5_x and last forward layer of the ResNet-101 and achieve the highest metrics, we obtained the following loss-graph:



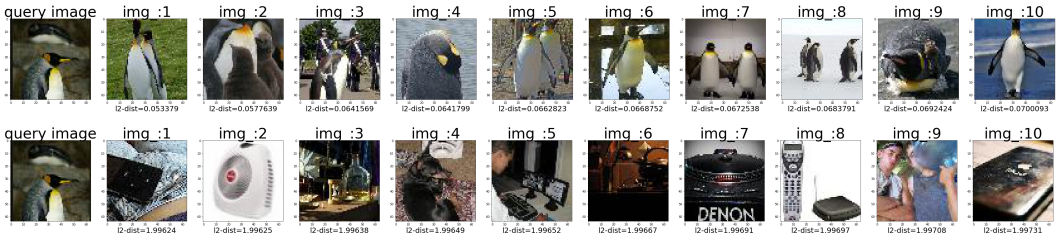
7 Ranked Examples

We now present for 5 query images from the validation dataset, with top 10 images and bottom 10 images for each:

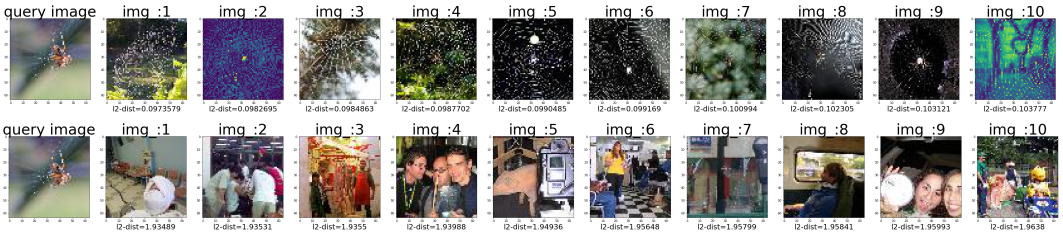
1.



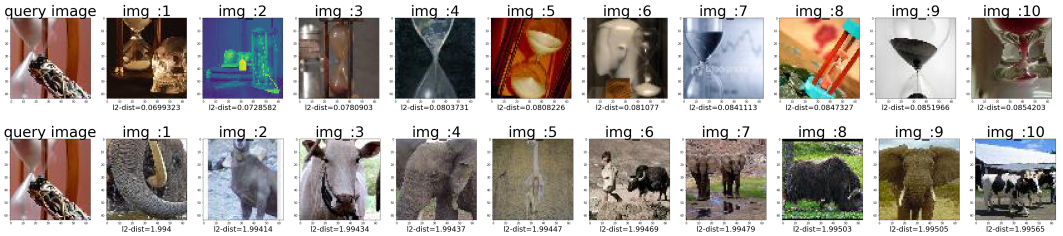
2.



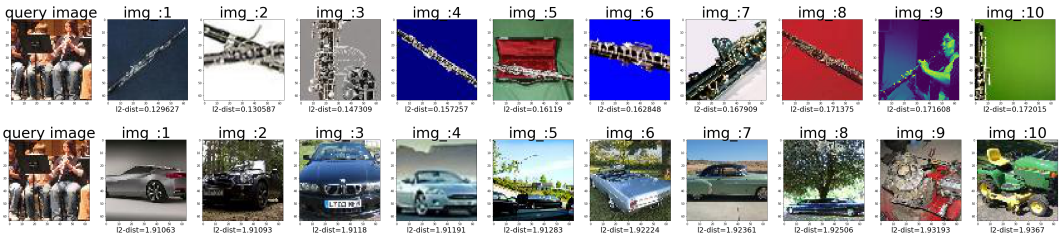
3.



4.



5.



8 Future Improvements

Here are some ways to improve our current model:

1. Use negative in-class examples, this helps account for fine-grained accuracy whereas here we only considered category level similarity.
2. Incorporate some-kind of ranking information, like the GoldenFeatures mentioned in [1]

References

[1] J. Wang, Y. Song, T. Leung, C. Rosenberg, J. Wang, J. Philbin, B. Chen, and Y. Wu, "Learning fine-grained image similarity with deep ranking," *CoRR*, vol. abs/1404.4661, 2014.

[2] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015.

[3] A. Zingade, "Image similarity using deep ranking," Dec 2017.