

# Learning Image Similarity with Deep Ranking

OSMAR CORONEL<sup>1</sup>, SUHAAS YERAPATHI<sup>1</sup>,  
PRADNYESH VINEET JOSHI<sup>1</sup>, SUNNY KATIYAR<sup>1</sup>, NAVEEN SATHIYANATHAN<sup>1</sup>

<sup>1</sup>University of Illinois at Urbana-Champaign

oac2@illinois.edu, sny2@illinois.edu, pvjoshi2@illinois.edu,  
sunnyk2@illinois.edu, nms9@illinois.edu

**Abstract.** This paper summarizes the work presented in Wang et al.[11] on learning fine-grained image similarity. The original paper proposes a deep ranking model that employs deep learning techniques to learn similarity metric directly from images, which works better than using hand-crafted features. We re-implement (with a little modification) the methodology proposed in Wang et al.[11]: (1) we compare different network architectures to extract image features, (2) we implement an efficient triplet sampling algorithm to learn the model with (non-parallelized) stochastic gradient descent algorithm.

**Keywords:** Residual Neural Network. Image Ranking. Image Retrieval

## 1 Introduction

Search-by-example, i.e. finding images similar to a query image is an integral function of modern search engines. Search-by-example relies on an effective image similarity metric to find similar images.

Most of the existing similarity models, such as Hadsell et al.[6] and Taylor et al.[9], consider category-level image similarity. In this approach, two images are considered similar as long as they belong to the same category. This category-level image similarity is not sufficient for the search-by-example image search application.

Boureau et al.[1], Chechik et al.[2] and Taylor et al.[9] take a different approach to the problem by first extracting the image features such as Gabor Filters, SIF and HOG, and then learning image similarity models on top of these features. The performance of these models is limited by the representation power of the image features. Jointly learning the image features and similarity models with supervised similarity information has been found to more effectively capture the fine-grained image similarity.

Deep learning models perform well on classification tasks, but these classification models cannot be directly used to capture fine-grained image similarity. The deep ranking model proposed in Wang et al.[11] characterizes the fine-grained image similarity relationship with a set of triplets. A triplet contains a query image, a posi-

tive image, and a negative image, where the positive image is more similar to the query image than the negative image. Deep ranking models use the relative similarity ordering in the triplets to achieve better performance than the category-level image similarity models.

Since training neural networks needs a large amount of data, Wang et al.[11] propose an online triplet sampling algorithm to generate meaningful and discriminative triplets. Asynchronous stochastic gradient algorithm is used to optimize a triplet-based ranking function. A multiscale network structure containing a convolutional neural network with two low resolution paths is used for similar image ranking.

To evaluate the model performance, the similarity relationship of the images is labeled using triplets. The performance of the image similarity model is then the fraction of the triplet orderings that agrees with the ranking of the model.

In conclusion, the main contributions of Wang et al.[11] are as follows: (1) A novel deep ranking model that can learn fine-grained image similarity model directly from images is proposed. We also propose a new bootstrapping way to generate the training data. (2) A multi-scale network structure has been developed. (3) A computationally efficient online triplet sampling algorithm is proposed, which is essential for learning deep ranking models with online learning algorithms. (4) An evaluation dataset with similarity ranking information for images from the same category has been released.

## 2 Related Work

Previous work on image similarity learning, such as Wang et al.[10] and Guillaumin et al.[5], study the category-level image similarity, where two images are considered similar as long as they belong to the same category. Existing deep learning models for image similarity also focus on learning category-level image similarity [9]. Category-level similarity denotes the semantic similarity, not the visual similarity. Deselaers et al.[3] show that while semantic and visual similarity are generally consistent across different categories, there is still variation within a category if the semantic scope of the category is large. So it is important to learn a model that captures fine-grained similarity for images within the same category.

Some of the works that are similar to Wang et al.[11] include relative attributes mentioned in Parikh et al.[7], which learns image attribute ranking among the images with the same attributes. OASIS [2] and local distance learning [4] learn fine-grained image similarity ranking models on top of the hand-crafted features, but are not deep learning based. Wu et al.[12] uses a deep learning based ranking model on top of "handcrafted" features, instead of directly using pixels. The Deep Ranking model proposed in Wang et al.[11] integrates the deep learning techniques with the fine-grained ranking model to learn fine-grained image similarity ranking model directly from images.

The triplet sampling algorithms in Chechik et al.[2] and Parikh et al.[7] assume that we can load the whole dataset into memory, which is impractical for a large dataset. Wang et al.[11] designs a computationally efficient online triplet sampling algorithm that does not require loading the whole dataset into memory, which makes it possible to learn deep ranking models with very large amount of training data.

## 3 Data Description

### 3.1 Training and Validation Data

The ImageNet ILSVRC-2012 dataset [8] is used for training and validation. The ImageNet data contains roughly 1000 images in each of 1000 categories. In total, there are about 1.2 million training images, and 50,000 validation images.

### 3.2 Triplet Evaluation Data

We have also used a triplet dataset released as part of Wang et al.[11] to evaluate our image similarity model.

To create this triplet evaluation data, the authors started from 1000 popular text queries and sampled triplets (Q, A, B) from the top 50 search results for each query from the Google image search engine. The images in the triplets were then rated using human raters. The raters had four choices: (1) both image A and B are similar to query image Q; (2) both image A and B are dissimilar to query image Q; (3) image A is more similar to Q than B; (4) image B is more similar to Q than A. Each triplet was rated by three raters. Only the triplets with unanimous scores from the three rates enter the final dataset. For our application, the triplets with rating (1) and rating (2) were discarded, because those triplets do not reflect any image similarity ordering. About 5,033 of such triplets are used in evaluation. The original images in the data set were up-sampled from 64x64 to 224x224 to make the dimensions consistent with the network input size.

## 4 Overview

The objective of this paper is to learn to retrieve similar images from the training set. Suppose we have a set of images  $P$  and a pairwise relevance score  $r_{i,j} = r(p_i, p_j)$  that scores the similarity between images  $p_i \in P$  and  $p_j \in P$ . Therefore, the objective is to learn a function  $f$  on the training set such that when given a distance function  $D$ , the relevance scoring function  $r$  and a set of three images  $p_i, p_i^+$  and  $p_i^-$  that correspond to the query image, an example positive image and an example negative image respectively, the function learns the following relationship:

$$D(f(p_i), f(p_i^+)) < D(f(p_i), f(p_i^-)) \forall p_i, p_i^+, p_i^- \text{ such that } r(p_i, p_i^+) > r(p_i, p_i^-)$$

### 4.1 Loss Function and Regularization

Let us consider the triplet sample  $t_i = (p_i, p_i^+, p_i^-)$ . We assume that the triplet comes from an unobserved triplet population  $T$  of all query images, positive images and negative images respectively and  $t_i \sim_{iid} T$ . For the given triplet  $t_i$  we define the hinge loss function:

$$l(p_i, p_i^+, p_i^-) = \max[0, g + D(f(p_i), f(p_i^+)) - D(f(p_i), f(p_i^-))]$$

where  $g$  is the gap parameter that regularizes the gap between the distance of two image pairs  $(p_i, p_i^+)$  and  $(p_i, p_i^-)$ . The hinge loss approximates the 0-1 binary ranking error loss. Unlike the binary error loss, the hinge loss is convex which makes it easier for the optimizer to learn parameters. The objective function is:

$$\sum_i \varepsilon_i + \lambda \|W\|_2^2; s.t. \max[0, g + D(f(p_i), f(p_i^+)) - D(f(p_i), f(p_i^-))] \leq \varepsilon_i \forall p_i, p_i^+, p_i^- \text{ such that } r(p_i, p_i^+) > r(p_i, p_i^-)$$

where  $\lambda$  is the regularization parameter that is used to improve generalization by manipulating the margin learnt by the network and  $W$  is the parameter set of learnt embedding function  $f$ . This problem is converted into an unconstrained optimization problem by setting  $\varepsilon_i = \max[0, g + D(f(p_i), f(p_i^+)) - D(f(p_i), f(p_i^-))]$ .

Learning the embedding function  $f$  requires an appropriate architecture that can extract image features without using hand-crafted features or predefined filters. In this paper we will explore different deep learning architectures for learning the image similarity model from images. We will employ triplet sampling and a suitable optimization function to minimize the loss function described above.

## 4.2 Evaluation Metrics

Two evaluation metrics have been used – similarity precision score and test accuracy at top-K for K=30. Similarity precision is defined as the percentage of triplets being correctly ranked. Given a triplet  $t_i = (p_i, p_i^+, p_i^-)$ , where  $p_i^+$  should be more similar to  $p_i$  than  $p_i^-$ . Given  $p_i$  as query, if  $p_i^+$  is ranked higher than  $p_i^-$ , then we say that triplet  $t_i$  is correctly ranked.

Test accuracy at top-K for K=30 for a test image is calculated as follows. For a given test image, we calculate the Euclidean distance between the test image and all train images. The test accuracy at top-30 is defined as the percentage of train images that are from the same class as the test image, within the nearest 30 train images.

## 5 Network Architecture

Multiple triplet sample based architectures were explored for learning the embedding function  $f$ . A triple sample  $t_i$  comprises of  $(p_i, p_i^+, p_i^-)$  where  $p_i$  is the query image,  $p_i^+$  is a positive image sample and  $p_i^-$  is a negative image sample. These samples are fed independently into three networks of identical structure and parameters. The neural network computes the embedding function  $f$  of an input image  $p \in P$ . For a chosen number of embeddings  $d$ , the output of the function is of the form  $f(p_i) \in R^d$ .

Hinge loss is evaluated on the top embedding layer for each triplet  $t_i$ . The ranking layer shown in the image does not have any parameter. It evaluates the triplet loss that estimates the model's violation of the ranking

order. Gradient of the loss is calculated and backpropagated to lower layers to adjust parameters to minimize the loss.

We use a residual neural network architecture to capture different invariances that are also captured by a convolutional neural network, but also does not suffer from issues related to propagation of information from the lower layers to higher layers. The head of the residual neural network is a fully connected network that takes the flattened output of the residual network and maps it linearly to embedding dimensions  $d$  which is a hyperparameter. Residual network architecture is expected to learn faster than an equivalent convolutional neural network.

### 5.1 Data Standardization and Augmentation

The input image comprises of three channels - red, green and blue. The residual network applies weights and non-linearities on all 3 input channels of the image. The input image is standardized using the global mean and standard deviation to ensure that the learning process leads to weights that are in proportional scale. Standardization also helps in contrast robustness of the learned weights.

### 5.2 Embedding Layer

The output of the residual neural network is flattened and fed to a fully network. The fully connected network linearly maps the output to a  $d$  dimensional vector, which is treated as the embedding layer.

### 5.3 Architecture Discussion

Although the residual network retains properties of a convolutional neural network such as translation and contrast invariances and overcomes the issue of stagnation in very deep convolutional networks, the invariances encoded in the architecture may be harmful for fine-grained image similarity tasks. Despite the shortcomings experiments using the data set for classification showed that residual networks outperform convolutional networks and fully connected networks in the given data set.

## 6 Training

### 6.1 Optimization

A deep neural network usually has multiple local minima. Convergence to global minima is not guaranteed

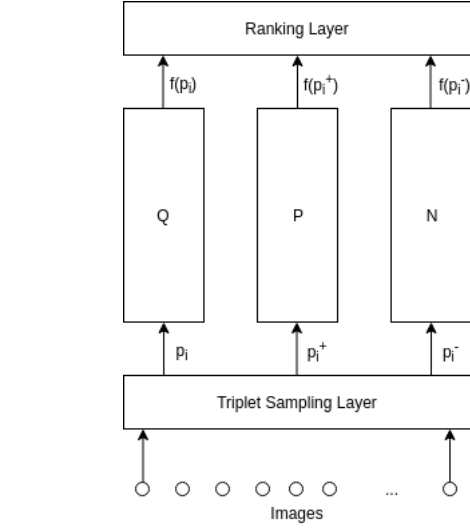


Figure 1: Network architecture for learning image similarity

for a deep neural network. Several theoretical and practical fixes have been used to avoid the problem of local minima for different optimization algorithms. Stochastic variant of Nesterov momentum is one such method that converges quicker than stochastic gradient descent.

Backpropagation algorithm is applied to compute the gradient of the loss function with respect to the parameters. The residual neural network can be represented in the form:

$$f(x) = g_n(g_{n-1}(\dots g_1(x)\dots))$$

where  $g_l$  is the mapping from layer  $l - 1$  to layer  $l$ . The parameters used for this mapping is  $w_l$ . The gradient of the embedding with respect to the parameters in layer  $l$  is given by  $\frac{\partial f(\cdot)}{\partial w_l} = \frac{\partial f(\cdot)}{\partial g_l} \frac{\partial g_l}{\partial w_l}$ . In the standard backward automatic differentiation setting we calculate  $\frac{\partial f(\cdot)}{\partial g_l} = \frac{\partial f(\cdot)}{\partial g_{l+1}} \frac{\partial g_{l+1}(\cdot)}{\partial g_l}$ .

## 6.2 Triplet Sampling

Deep learning algorithms work well with large number of training examples. Based on the unrestricted set of triplet of the form  $(p_i, p_j, p_k)$  the total number of samples is approximately  $(1 \times 10^5)^3 = 1 \times 10^{15}$ . It is computationally infeasible to create and utilize such as large data set for training. Under the proposed triplet set  $t_i = (p_i, p_i^+, p_i^-)$  we get a total of  $200 \times \frac{500 \times 499}{2} \times 199 \times 500 = 2.48 \times 10^{12}$  samples, which is still a large number.

Sampling triplets for training is critical for learning the ranking function  $f$  using a neural network. Sam-

pling a triplet uniformly from the data set is suboptimal because a uniformly sampled triplet does not effectively capture top-ranked similar images to a given query image. Therefore, an online version of sampling is applied for sampling a triplet for learning the parameters.

To sample a triplet  $t_i = (p_i, p_i^+, p_i^-)$  we first sample a query image  $p_i \in P$  randomly from the data set. The class  $c_i$  of the query image  $p_i$  is noted. Let  $C_i$  be the set of all images  $p_i \in P$  such that the class of  $p_i$  is  $c_i$  and  $C'_i$  be the set of all images  $p_i \in P$  such that the class of  $p_i$  is not  $c_i$ . The positive image is sampled randomly such that  $p_i^+ \in C_i; p_i^+ \neq p_i$ . The negative image is sampled randomly such that  $p_i^- \in C'_i$ . An alternative to this sampling is importance sampling based on relevance score of the query image, positive image and negative image. However, this method was not used because of the computational cost of sampling which adds to the computational cost of training the network parameters.

## 6.3 Mini-batch Training

The deep residual neural network architectures used for learning the function  $f$  create large tensors that are stored in GPU memory. The entire batch of images cannot be loaded in-memory for the purpose of training because of the limited GPU memory. Different mini-batch sizes were required for training residual neural networks with different number of blocks because the number and sizes of tensors varies by the structure of the network. Therefore, the batch version of gradient is replaced by the mini-batch version  $L = \frac{1}{M} \sum_i^M l_i$  where  $M$  is the mini-batch size. Similarly, the gradient is computed as  $G(w_j) = \frac{1}{M} g_i(w_j)$ , where  $g_i(w_j)$  is the stochastic version of gradient of the loss function with respect to weight  $w_j$  computed using sample  $i$ .

## 7 Data Augmentation

Embedding function  $f$  learnt by a neural network can be affected by factors such as translation, rotation, contrast and blur levels of the images used for training. Therefore the effectiveness of similarity learnt by the network depends on the embedding of training images  $f(p_i^+)$  and  $f(p_i^-)$  for a given image  $p_i$ , which depend on the mentioned factors. In order to match the distribution of training and testing images we employ several data augmentation methods: random crop, horizontal flip, color jitter, etc. Different network architectures are sensitive

to different transformations. It was observed that random horizontal crop and flip (in given order) worked well for tiny imagenet data set.

## 8 Hyperparameters

Choice of hyperparameters is constrained by the data set and the total GPU memory available for computation. Tiny imagenet contains images of dimension  $64 \times 64 \times 3$  each. Due to the number and size of tensors created during the forward and backward passes the batch size cannot be set to an arbitrarily large number. The batch size was chosen for different residual network structures by trial and error to avoid GPU memory overflow.

It was noticed that Adam optimizer could not be used due to numerical overflow. Therefore, stochastic gradient descent (SGD) was used for learning the network parameters by optimizing the loss. Different learning rates were used for different residual network structures because SGD applies the same learning rate for all weights and does not adapt the learning rate across iterations. A sample of hyperparameters used for different network structures is shown in table 1.

## 9 Description of Code

The code flow works as follows: the main.py file runs and creates a model, which is then passed into a Data object defined in the utils.py file. The data object then creates datasets based around the images in the training and validation sets within tinyimagenet. These datasets are custom made - they fulfill the purpose of the triplet sampling layer. When the next sample is called from the dataset, it returns a triplet. First, a single image is sampled based on the next sample index. Then, based on the index, it gets another image of the same class. Finally, it makes a list of all classes that are not the same index, and randomly samples an index and a random image from that class. This is equivalent to the negative image.

After this is done, the training loop occurs, also in utils.py. It is a simple training loop, but for each item retrieved from the dataset, the query, positive, and negative image is forwarded through the model. The loss is calculated using a triplet margin loss criterion, and then backpropagation occurs using the loss.

Testing works by pushing all query images through the model to get the train and test embeddings of each image, then we measure the distance of each image to

get the top-k values. The average correct nearest neighbors across the embeddings is the top-k accuracy.

A random sample of the query images are used to get the top 10 images to show, as well as the bottom 10 images.

## 10 Computational cost of training

The final model that we built had used a pre-trained ResNet50 model, with the final fully connected layer being replaced with a fully connected layer with more output features, 2048 in this case. With a batch size of 10, it took roughly 24 hours for the model to train on Blue Waters and reach a train loss of 0.06975 with a top-30 accuracy on the test dataset of 62.0%. We tried using a larger batch size for faster convergence and for saving some computation time, however, a batch size of more than 10 would give us the GPU memory error.

## 11 Results

The first model that we built had used a pre-trained ResNet18 model. ResNet18 has a shallower network as compared to other models (ResNet50, ResNet101, etc) that have deeper networks. This model didn't give promising results and was badly overfitted. The train loss after 49 epochs reached 0.000429. However, the top-30 accuracy on the test set was only 4.62%.

Another model that we built had used a pre-trained ResNet101 model. The deep network of ResNet101 would lead to GPU memory error whenever we tried using a large batch size. A smaller batch size would mean slower convergence and higher computation time. After training this model for 60 hours on Blue Waters, the train loss reached 0.0478 with the top-30 accuracy of 37.56% on the test set. We also tried using a pre-trained ResNet34 model. It gave a train loss of 0.643 after 40 hours of training on Blue Waters and gave the top-30 accuracy of 2.32% on the test set.

Our final model that used the pre-trained ResNet50 model with a 2048-feature output layer achieved the desired results. It reached the train loss of 0.06975 after 24 hours of training on Blue Waters and gave the top-30 accuracy of 62% on the test set. Figure 2 in the appendix represents how the train loss decreased with each epoch in the final model.

Table 2 shows the computation time, train loss, and top-30 test accuracy of different models that we built.

Table 3 shows the similarity precision score obtained by the final trained model for different data sets.

S.No	Base model	Learning rate	Batch size	Step size	Gamma	Embedding size
1	ResNet18	0.001	16	5	0.1	2048
2	ResNet101	0.0002	8	5	0.4	4096
3	ResNet50	0.0001	10	3	0.5	2048
4	ResNet34	0.001	12	10	0.1	2048

**Table 1:** Hyperparameters for best performing models on training set

Base model	Computation time (hours)	Train loss	Top-30 test accuracy
ResNet18	40	0.0004	4.62%
ResNet34	40	0.6432	2.32%
ResNet101	60	0.0478	37.56%
ResNet50	40	0.0417	60.13%
ResNet50*	24	0.0697	62.00%

**Table 2:** Results from different pre-trained models  
Note\*: Final tuned ResNet50 model

Data set	Similarity Precision Score
ImageNet Train Set	90.84%
ImageNet Test Set	90.49%
Triplet Evaluation Data	77.97%

**Table 3:** Similarity precision score for different data sets

Figure 3 in the appendix represents the top and figure 4 in the appendix represents the bottom images obtained on 5 different query images using the final trained model. The number above the query image is the class and embedding, and the numbers above the next ten images are the class and the euclidean distance from the query.

## 12 Scope for Improvement

The model learns an embedding function  $f$  that preserves similarity between images. It performs well on images that are similar in distribution to the images in the data set (Tiny Imagenet). It also generalize well on unseen data set that was manually labelled. However, it does not incorporate fine-grained features and invariances. This may be because the triplet sampling is done randomly within and outside the class of the query image while training. Fine-grained features can be incorporated by ranking the relevance of positive and

negative images with respect to the query image and sampling with probability proportional to the relevance. This modified form of importance sampling works well for image ranking task, but was not explored because of the additional computational time for sampling.

It was also observed that residual neural network with more parameters and larger batch size (for stability of gradient estimate) can capture fine-grained features and invariances more accurately than shallower residual neural networks. Training such large networks with large batch sizes requires more GPU memory. Therefore, deeper models were not explored for image ranking.

## 13 Public Repositories

There were a couple of public repositories that did similar image ranking tasks. In particular, these three were used as guiding points for our project:

- <https://github.com/SathwikTejaswi/deep-ranking>
- <https://github.com/Zhenye-Na/image-similarity-using-deep-ranking>
- <https://github.com/akarshzingade/image-similarity-deep-ranking>

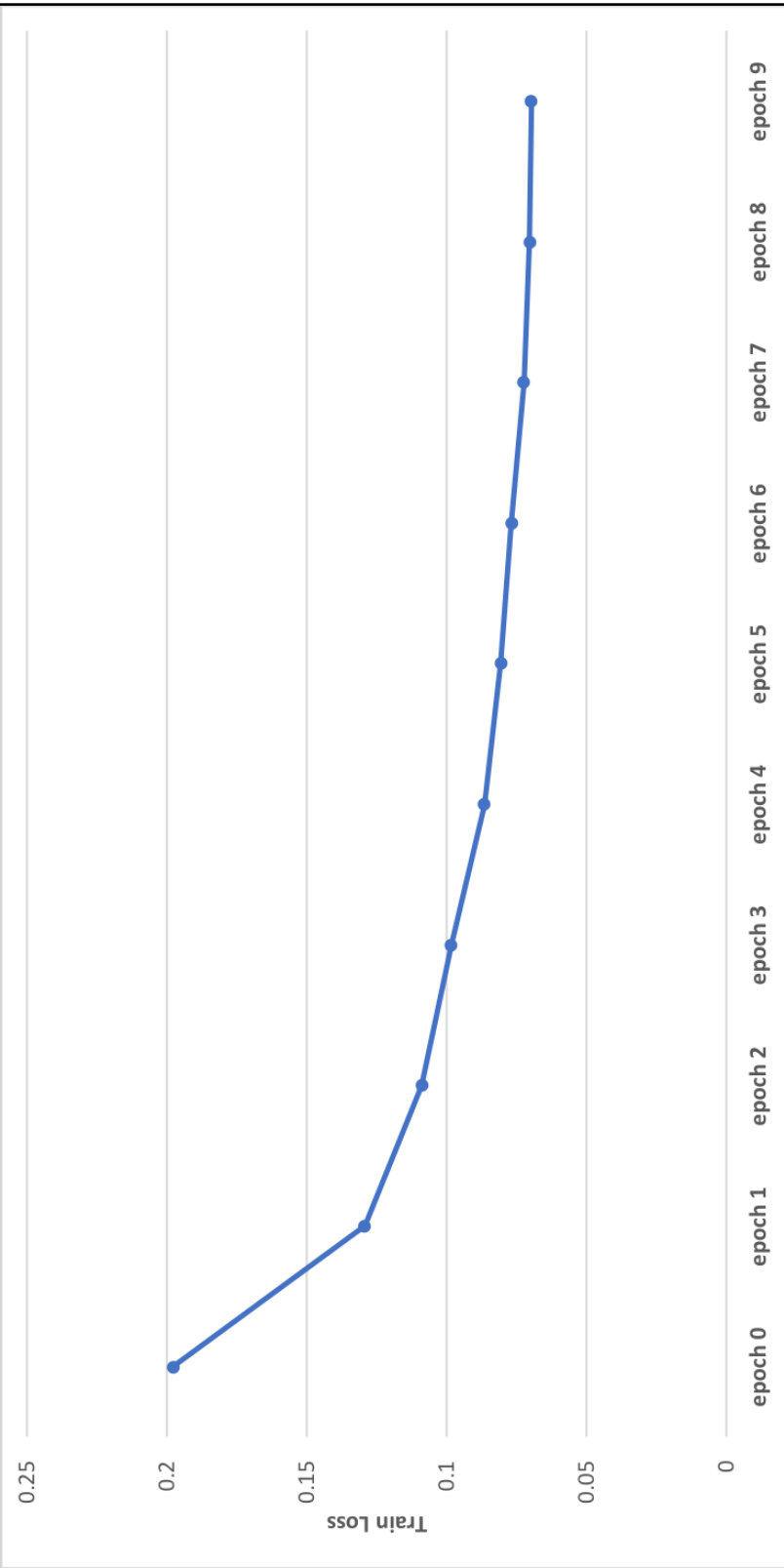
These projects were used to guide our creation of the triplet sampling for our project. They gave various ways of implementing the layer, 2 of them had varying ways of writing the triplets to a text file then sampling it, and one of the repositories had a way of sampling a triplet by creating a custom loader. For our project, we created a custom loader to sample triplets, similar to <https://github.com/akarshzingade/image-similarity-deep-ranking>. In terms of our contributions to the code, we changed a lot of how the dataloader works compared to the public repository. Instead of sampling images and then outputting them together when `__getitem__` was called, we sampled and output triplets directly when `__getitem__` was called. We also did not use the images folder for sampling our images and classes, but instead

we indexed the images and classes, and then sampled from the indexes.

The training code was all from previous homework assignments, and the training loop was very standard as a result. The images were created without the use of public code. The accuracy calculation was done using Torch tensors, and was also pretty standard.

## References

























































- [1] Y.-L. Boureau, F. Bach, Y. LeCun, and J. Ponce. Learning mid-level features for recognition. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 2559–2566. Citeseer, 2010.
- [2] G. Chechik, V. Sharma, U. Shalit, and S. Bengio. Large scale online learning of image similarity through ranking. *Journal of Machine Learning Research*, 11(Mar):1109–1135, 2010.
- [3] T. Deselaers and V. Ferrari. Visual and semantic similarity in imagenet. In *CVPR 2011*, pages 1777–1784. IEEE, 2011.
- [4] A. Frome, Y. Singer, and J. Malik. Image retrieval and classification using local distance functions. In *Advances in neural information processing systems*, pages 417–424, 2007.
- [5] M. Guillaumin, T. Mensink, J. Verbeek, and C. Schmid. Tagprop: Discriminative metric learning in nearest neighbor models for image auto-annotation. In *2009 IEEE 12th international conference on computer vision*, pages 309–316. IEEE, 2009.
- [6] R. Hadsell, S. Chopra, and Y. LeCun. Dimensionality reduction by learning an invariant mapping. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR’06)*, volume 2, pages 1735–1742. IEEE, 2006.
- [7] D. Parikh and K. Grauman. Relative attributes. In *2011 International Conference on Computer Vision*, pages 503–510. IEEE, 2011.
- [8] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015.
- [9] G. W. Taylor, I. Spiro, C. Bregler, and R. Fergus. Learning invariance through imitation. In *CVPR 2011*, pages 2729–2736. IEEE, 2011.
- [10] G. Wang, D. Hoiem, and D. Forsyth. Learning image similarity from flickr groups using stochastic intersection kernel machines. In *2009 IEEE 12th International Conference on Computer Vision*, pages 428–435. IEEE, 2009.
- [11] 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. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1386–1393, 2014.
- [12] P. Wu, S. C. Hoi, H. Xia, P. Zhao, D. Wang, and C. Miao. Online multimodal deep similarity learning with application to image retrieval. In *Proceedings of the 21st ACM international conference on Multimedia*, pages 153–162. ACM, 2013.



**Figure 2:** Train loss vs epoch



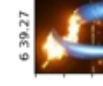
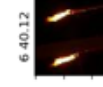
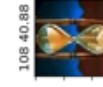
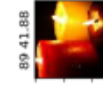
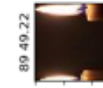
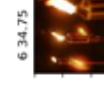
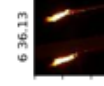
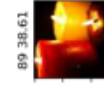
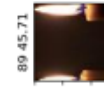
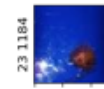
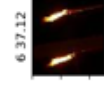
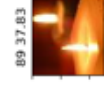
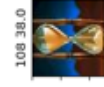
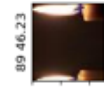
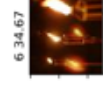
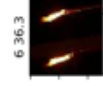
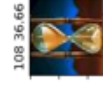
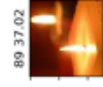
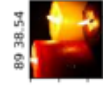
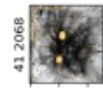
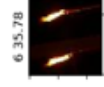
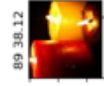
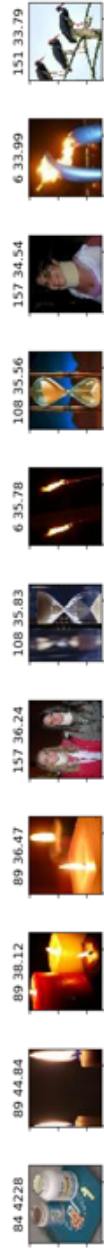
# Query

84 4228		84 8.07		84 8.22		84 8.41		84 8.46		84 8.57		84 8.68		84 8.68		84 8.68		84 8.86		84 8.88		84 8.89	
41 2068		41 9.59		41 9.71		143 10.07		41 10.07		41 10.12		41 10.3		41 10.47		41 10.5		41 10.65		41 10.78			
10 523		10 8.29		10 9.11		10 9.14		10 9.29		10 9.31		10 9.32		10 9.36		10 9.59		10 9.72		10 9.83			
23 1184		23 7.11		23 7.93		23 8.21		23 8.29		23 8.42		23 8.44		23 8.49		23 8.5		23 8.56		23 8.58			
182 9142		151 7.22		151 8.19		182 8.31		151 8.37		151 8.4		151 8.82		151 8.87		151 8.99		151 9.05		151 9.1			

# Top Images

Figure 3: Top Images

## Query



## Bottom Images