Clustering And Querying Images From Unknown Classes Using Metric Learning

Chinmayee Shah
Electrical Engineering, Stanford University
http://web.stanford.edu/~chshah/
chshah@stanford.edu

Abstract

Recently, deep learning has been successfully used to learn meaningful embeddings for image data. Such learned embeddings have various applications including fine grained visual classification, with many classes and only a few images per class, face recognition, and retrieving similar images using a distance-based similarity metric. In this project, we use a triplet network to discrminatively train a network to learn embeddings for images, and evaluate clustering and image retrieval, on a set of unknown classes, that are not used during training. We observe that by postponing the construction of triplets to after the network, that is, by constructing triplets in the feature space rather than the image space, we can take better advantage of large batches and hard examples. Another advantage of such an approach is that it removes the sensitivity on hard mining. We observe that with multiple clusters per class, we can better capture intra-class variance. We evaluate our approach on CUB-200-2011 birds data-set, and report F_1 score and NMI for clustering. Our code is available at `https://github.com/schinmayee/metric-learning`*, and is included in the supplementary material as well.*

1. Introduction

A semantically useful similarity or distance metric can be useful for many real world tasks, such as finding products similar to a given product, finding images such as flowers or birds similar to a queried image, clustering uncategorized Flickr photographs using a network trained on a limited, labeled data-set, or even querying photographs such as those of a particular dog, a place, or a person, from photo albums such as Google or Facebook photographs. Metrics such as Euclidean distance and cosine similarity do not perform well on raw pixels of images. Recent end-to-end approaches [17, 3, 16, 22] approach the problem of learning a similarity/distance metric by discrminatively training a network to learn embeddings so that similar images are close to each other, and images from different classes are far, in this feature space. Such features are shown to outperform manually crafted features such as SIFT, and various binary descriptors [9, 19].

Given these learned features, a classification problem can be reduced to a K-nearest neighbors problem, using distance between the features as a measure similarity. This can particularly useful when all categories are not available during training, but a few labeled images, such as labeled photographs of a particular dog, a specie of bird, a product or a place, might be available. A query for images similar to a given image can retrieve K candidate images that are closest to the query image. Similar images can be grouped using K-means clustering over this feature space.

Conventional image classification methods perform poorly in extreme classification problems, with large number of classes, few images per class, and a large intra-class variance and low inter-class variance. The two major challenges in this setting are the linear dependences of models and algorithms on the number of classes, and the small number of examples per class available during training. Recent works such as [13, 17, 8, 12, 2] discrminatively train neural networks to directly learn a mapping function from an input image to a lower dimensional embedding. They use triplets consisting of an anchor image, a positive image that belongs to the same class as anchor, and a negative image from a different class. The loss is designed to increase the distance between anchor and negative, and reduce the distance between anchor and positive, so that images from similar classes group together in the learned feature space. Conventional triplet based approaches such as the ones in [13, 17] however are very sensitive to methods used for hard mining, and can produce disappointing results in our experience. Moreover, they are hard to train because of small batch sizes imposed by memory limits, resulting in diverging gradients.

Motivated by [16, 2, 12], in this project, we use *batch-hard* triplets — we forward a batch of images with a few randomly sampled classes and a few images per class through the network to compute embeddings, and then take add contributions from all possible triplets, with a non-zero loss, at the output. This makes the learning process more efficient as each batch includes tens of thousands of triplets, more stable due to the large batch size, and also removes the sensitivity on hard mining.

In the spirit of general metric learning where the task is to learn a generic concept of distance, we divide our data-set into disjoint classes — 100 for training and validation, and 100 for testing. We use the same number of classes in training and test set, as in [16] and over the same data-set, CUB-200-2011 [21], but note that our train/test split could still be different, due to differences in the list of classes used for training and testing. We compute F_1 and NMI metrics to evaluate clustering, and accuracy and Recall@K to evaluate image retrieval. This allows us to directly compare our results with state-of-the-art methods already published.

The fine grained visual classification work in [8] observes that with extreme number of classes, the intra-class variation can be large, so that sometimes, images from the same class might be farther than images from different classes. It proposes learning manifolds rather than clusters, to account for intra-class variation. While we do not learn key points for clusters, as that makes little sense when classes for train and test data are disjoint, we evaluate clustering with multiple clusters per class. We find that when multiple cluster per class are used to drive model selection, clustering and querying scores for the resulting model are higher. We also evaluate a local positive sampling method, that restricts sampling of positive images within a batch to a neighborhood around an anchor point, rather than all positive pairs, when generating triplets at train time, as mentioned in [8]

2. Related Work

Siamese networks applied to signature verification showed the ability of neural networks to learn a compact embedding [5]. OASIS [6] and local distance learning [10] learn fine-grained image similarity ranking models using hand-crafted features, that are not deep-learning based. Recent approaches such as [17, 3, 16, 22] approach the problem of learning a distance metric by discrminatively training a neural network. Such features are shown to outperform manually crafted features [3] such as SIFT, and various binary descriptors [9, 19].

Deep metric learning can be broadly divided into contrastive loss based methods, triplet networks, and approaches that go beyond triplets such as quadruplets, or even batch-wise loss. Contrastive embedding is trained on paired data and tries to minimize the distance between pairs of examples with same class label, while penalizing examples with different class labels that are closer than a margin m [11]. Triplet embedding is trained on triplets of data with an anchor point, a positive that belongs to the same class, and a negative that belongs to a different class [23, 13]. Triplet networks use a loss over triplets to push the anchor and positive close, and penalize triplets where the distance between anchor and negative is less than the distance between anchor and positive, plus a margin m. Contrastive embedding has been used before for learning visual similarity for products [4]. Triplet networks have been used for face verification, person re-identification, patch matching, for learning similarity between images and for fine-grained visual categorization [17, 18, 22, 8, 3].

A naive triplet-based approach does not perform well due to zero loss from easy examples where the negatives are far from anchor. Various methods to perform hard mining or semi-hard mining are discussed in [17, 8]. [3] tries to reduce dependence on, and cost of hard mining by proposing in-triplet hard examples where they flip anchor and positive if the loss from the resulting new triplet is larger.

More recent works use quadruplets [7] or even a loss over the entire batch [16], to improve the network stability and accuracies. Other approaches [2, 12] propose computing all hard triplets and taking their loss contribution, within a batch of images, rather than a batch of triplets, input to a network. We found that such batch global approaches provide a significant improvement over the conventional triplet network, and reduce dependence on hard mining, giving better results inspite of more expensive batches.

3. Dataset

We present results for CUB-200-2011 birds data-set [21], that has also been used for evaluation by previous works [8, 16]. The data-set has 200 species of birds with a total if 11,788 images across all species. Similar to [8], we divide the data-set into disjoint classes for training and testing. We utilize the bounding box information in the data-set to crop the images, and rescale the images to a fixed size as expected by the network.

4. Approach Overview

We use a neural network to map images to features of size 64. We use a triplet-based loss function over these features, to discriminatively train the network, that is, to push images from the same class closer, and images from different classes further apart. For validation, we compute the embeddings for all images in the validation set, and perform a K-means clustering. For final evaluation, we choose the model that gives highest clustering accuracy on validation data. For retrieving images similar to query images from test set, we compute the distance to all images in test set, and retrieve the images that are closest to the query image.

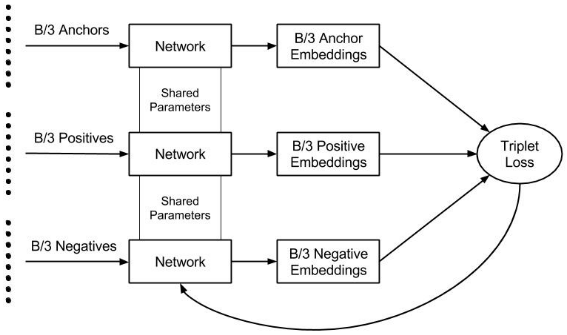

Figure 1. A conventional triplet network feeds in triplets of images at input, and uses the loss over these input triplets to train. A batch with B images only has $\frac{B}{3}$ triplets.

5. Triplet Based Approach

There are several approaches to modeling loss for metric learning. We focus on the more recent triplet-based approach, as used in many recent works [8, 12, 13, 3, 17]. A triplet consists of 3 images, an *anchor* image (a), a *positive* image (p) that belongs to the same class as a, and a *negative* (n) that belongs to a different class. The basic idea is to map the images a, p and n to features f_a, f_p and f_n, so that f_a and f_p are closer than f_a and f_n. We normalize the features f_a, f_p and f_n, similar to [8, 22], before computing the distances. Let d_{ap} denote the distance between normalized anchor and positive features, and d_{an} denote the distance between normalized anchor and negative features, computed using L_2 distance. That is, $d_{ap} = ||\frac{f_a}{||f_a||_2} - \frac{f_p}{||f_p||_2}||_2$, and $d_{an} = ||\frac{f_a}{||f_a||_2} - \frac{f_n}{||f_n||_2}||_2$.

5.1. Conventional Triplet Approach

A conventional triplet based approach as presented in [13, 17, 8], samples triplets at input. Figure 1 illustrates this. This is also the first approach we tried. A batch of triplets of size $\frac{B}{3}$ consists of B images, as each triplet has 3 images. These images are forwarded through the same network, to compute embeddings, and then generate the triplet loss. This triplet loss is finally back-propagated through the network for training.

There are various ways to compute triplet loss. We experimented with the following loss functions for the conventional triplet approach (loss is averaged, in the standard manner, over all triplet samples):

- **Hinge Loss:** The most commonly used loss function uses hinge loss, with a hyper-parameter, margin, m [17, 23, 13]. The loss function is expressed as:

$$L(a,p,n) = \max(0, d_{ap} - d_{an} + m)$$

There are also variations that use squared distances instead, to compute the hinge loss, as follows:

$$L(a,p,n) = \max(0, d_{ap}^2 - d_{an}^2 + m)$$

In our experiments, the first version outperformed the second one with squared terms.

- **Ratio Loss:** A recent work that uses triplet loss for determining similar image patches [3] proposes a ratio loss, to interpret closeness of features. The idea is similar to a logistic function — the loss is always nonzero, so that all triplets contribute to some learning for the network. We experimented with a variation, the following loss function:

$$L(a,p,n) = (\frac{e^{d_{ap}}}{e^{d_{ap}} + e^{d_{an}}})^2 - (\frac{e^{d_{an}}}{e^{d_{ap}} + e^{d_{an}}})^2$$

The total loss for the network, over all $\frac{B}{3}$ triplets input to the network, T, is then computed as $\frac{3}{B} \times \Sigma_{(a,p,n)\in T} L(a,p,n)$ This loss is back-propagated to update network parameters.

A naive approach with randomly sampled triplets from training set however, performs poorly. We implemented online hard-mining to improve the clustering accuracy — we do this by sampling hard negative examples while training, within each batch, and then regenerating all triplets every few epochs. The regenerated triplets use a mix of these hard examples, and new random examples, so that we get new negatives. For hard negatives, our first approach was to sample anchor-negative pairs with the least distance, within each batch, add those to a pool of hard negatives, and use these when regenerating triplets. We also implemented semi-hard mining, similar to the idea in [17], where we sample anchor-negative pairs with $d_{an} < d_{ap}$, instead of anchor-negative pairs with the least separation, in an attempt to make the loss more stable, and converge to better optimal solutions. Since the final objective is to achieve good K-means clustering and query results, we experimented with a third strategy for hard-mining, where we use examples that are misclassified with K-means clustering, as hard negatives.

Finally, we also implemented in-triplet hard mining, by flipping anchor and positive if the resulting loss is larger, as suggested in [3]. However, we found in-triplet hard mining to not be very useful as either the examples are too easy with random mining, or the constructed triplet generates a good loss with hard mining.

In our experiments, we found that hinge loss outperformed ratio loss. We found hard mining to be useful, over random samples, but accuracies with all hard negative sampling approaches to be comparable. This might also be because our network itself was very unstable with the conventional triplet based approach — the small batch

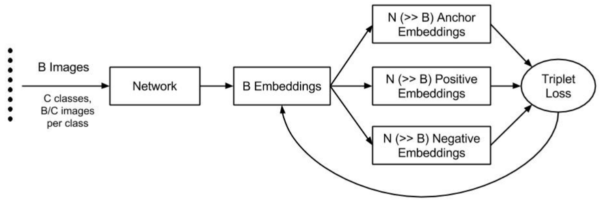

Figure 2. Batch hard approach selects all hard triplets from all combinations of triplets within a batch.

sizes imposed by GPU memory limit causes diverging gradients and the loss to oscillate a lot, preventing the network from learning anything useful and converging well (recall each batch with B images only has $\frac{B}{3}$ triplets). We experimented with a lot of different hyper-parameters, loss functions, and sampling strategies, but found this to be a significant problem, due to the small batch size. It is possible to address this problem by forward propagating multiple batches to compute loss and then back propagating the net loss. However, the result is still very inefficient. Very recent works [16, 12, 2] have also described this problem, and proposed efficient solutions to ameliorate this problem. Our next approach, *batch hard*, is motivated by these.

5.2. Batch-Hard Approach

Motivated by [16, 12, 2], we tried a different approach to constructing triplets. Figure 2 illustrates the idea. In this approach, instead of feeding in triplets of images at the input, we construct a batch of size B, with C randomly selected classes, and $\frac{B}{C}$ randomly sampled images per class. These images are forwarded through the network to compute B embeddings. We then loop over all possible anchor, positive, negative triplets in each batch, select all *hard* triplets that have a non-zero loss (that is, filter out triplets which would not contribute to loss and only reduce the gradient magnitude during back-propagation), compute the total loss, and back-propagate this loss. For an input batch I of size B, let $S = \{(a,p,n)|a,p,n \in I, C_p = C_a, C_n \neq C_a, L(a,p,n) \neq 0\}$. Here, C_i denotes class of i. Then the total loss equals $\frac{1}{|S|} \times \Sigma_S L(a,p,n)$. This method generates $O(\frac{B^3}{C})$ triplets in every batch, much larger than $\frac{B}{3}$ ($O((\frac{B}{C})^2 \times C)$ positive-anchor pairs, $O(B)$ triplets per positive-anchor pair). With this approach, we found that it was easier to train the network, and the accuracies that we got were much higher than ones with conventional triplet networks.

In addition to the hinge loss and ratio loss described in the conventional triplet section, we also tried a *soft* loss for the batch-hard approach. Inspired by [12, 16], we designed this loss function as follows:

$$L(a,p,n) = d_{ap} + \log(e^{m-d_{an}} + e^{m-d_{pn}})$$

Here, m is a hyper-parameter, and d_{pn} is the distance between positive and negative, $d_{pn} = ||\frac{f_p}{||f_p||_2} - \frac{f_n}{||f_n||_2}||_2$. The idea is to also use contributions from the positive-negative pair within a triplet. We found the results comparable to hinge loss, but not much better. However, we think it is possible to get better results with more hyper-parameter tuning. Similar to conventional network, we got best results for simple hinge loss with distances, compared to the one with squared distances.

5.3. Local Positive Sampling

Inspired by [8, 18], which propose local sampling, to account for intra-class variance, we tried local positive sampling for constructing triplets. In this approach, instead of looking at all anchor, positive, negative combinations within a batch, we consider only those where the anchor-positive distance is within 60 percentile of all anchor-positive pairs for that anchor. The motivation is to exclude very hard examples, and to try to map images to extended manifolds in the feature space, rather than spheres, to account for intra-class variance. Local positive sampling works better when we use K-means clustering with multiple clusters per class. However, we did not see significant improvements with local positive sampling, and hypothesize that this might be due to robustness of the *batch hard* approach. Note that the works [8, 18] that proposed learning manifolds used the conventional triplet architecture.

6. Evaluating Clustering and Querying

Once we have a trained network, we cluster images from test data, using K-means. Such clustering is useful for grouping similar images. We also generate random queries, and retrieve images closest to the query image. This section describes how we implement and evaluate clustering and image retrieval.

6.1. K-Means Clustering Over Learned Features

We use the trained network to compute embeddings for all images in test data. We then initialize K-Means with number of centroids equal to the number of classes in test data, similar to [16]. Once we have the images clustered, we label each cluster with the class of the most frequently occurring class in that cluster, as explained in [15]. The accuracy or purity of clustering is then defined as the ratio of the sum of number of points that belong to a class C and are also in a cluster labeled C, to the total number of points [15].

We also compute normalized mutual information, NMI [16, 15] Consider set of clusters $\Omega = \{\omega_1, \dots, \omega_k\}$, and ground truth classes $C = \{c_1, \dots, c_k\}$, where w_i indicates the examples with cluster assignment i, and c_i indicates examples with class i.

$$\text{NMI}(\Omega, C) = \frac{2I(\Omega, C)}{H(\Omega) + H(C)}$$

Here, I indicates mutual information, and H is entropy. Finally, we also compute the F_1 score, the harmonic mean of precision and recall, with precision and recall computed using number of true negatives and positives, and false negatives and positives, as described in [15], and also [16].

6.2. Intra-Class Variation With Multiple Clusters

It has been shown that for fine-grained classes, there is often very high intra-class variation [8]. To account for the variation, we also compute K-means clusters with multiple clusters per class. For model selection, we experimented with the model with the highest clustering accuracy when using 3 clusters per class. On test data with N classes, we experimented with K-means clustering with $3N$ clusters. This not only yields better accuracies and F_1 score for clustering, but also better accuracies for queries.

6.3. Querying Using K Nearest Neighbors

We randomly select a few images from each class in test data, to query. For each query image, we compute the distance in feature space, to all the remaining images in test data, and return images that are closest to the queried image. We report Recall@K, as presented in [16], and explained in [14], for various choices of K, the number of result images returned. Recall@K assigns a score of 1 for if a result with K images contains at least one correct result, that is, an image that is the same class as the queried image. The score is then averaged over all queries. We also evaluate accuracy, the fraction of results that are the same class as queried image, averaged over all queries.

7. Implementation Details

We implemented the discussed methods using Python 2.7 and PyTorch [1]. We tested our implementation on custom networks with a few convolutional, batch normalization, pooling and fully connected layers, trained from scratch, and on several existing pretrained networks. We got best results training a Inception V3 [20] that was pretrained on ImageNet. For Inception V3, we resize images to 299×299, as expected by the network. We replaced the final fully connected layer for classification in Inception V3 with a fully connected layer mapping to an output of size 64, to compute features of size 64. We also performed L_2 normalization on features, similar to that in [8, 22]. We used Adam optimizer, and scaled down the learning rate for all but the final layer, to improve the learning speed, without disturbing the lower-most layers. The complete code with scripts is available at `https://github.com/schinmayee/metric-learning`, and is also included in the supplementary material, with a `README.md` with details. The supplementary material also includes logs used to generate results presented in this report.

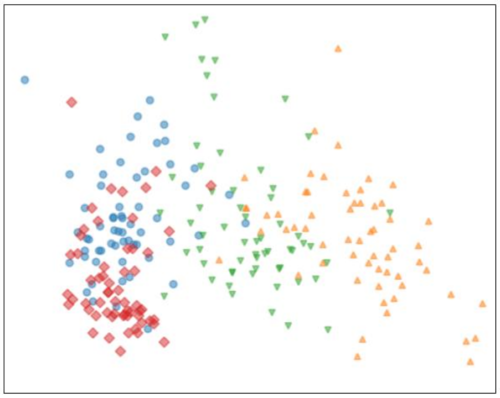

Figure 3. PCA visualization for 4 random classes from test set, with the 2 most significant components — markers with different colors and shapes denote different classes.

8. Experiments And Results

Figure 3 shows a 2D plot of 2 most significant components of computed embeddings of size 64, for 4 randomly selected classes in the test data. As expected, points that belong to the same class tend to group together. However, there is sometimes significant variation within a class, as mentioned in [8], and seen from the spread of points. The following sections evaluate clustering and image retrieval, on the same lines as [16]. We used a batch size of 64 images, with 4 classes per batch, and 16 images per class, for the *batch hard* approach. We trained our network for a total of 1000 iterations (batches) or less, which was just a few hours with a single GPU, and got impressive results.

8.1. Clustering

	F_1 Score	NMI
Hinge Loss, 1 Cluster	**0.17**	**0.35**
Hinge Loss, 3 Clusters	**0.24**	**0.39**
Hinge Loss, 3 Clusters, Local	0.21	0.35
Square Hinge Loss, 1 Cluster	0.12	0.25
Square Hinge Loss, 3 Clusters	0.17	0.28
Square Hinge Loss, 3 Clusters, Local	0.16	0.28
Soft Loss, 1 Cluster	0.15	0.31
Soft Loss, 3 Clusters	0.21	0.35
Soft Loss, 3 Clusters, Local	0.21	0.35

Table 1. Results on test data, for different loss functions, number of clusters per class, and sampling, for *batch hard* approach.

Table 1 gives the F_1 score and NMI for 3 loss functions — hinge loss, hinge loss with squared distances, and soft loss. Results for one cluster per class and three clusters per

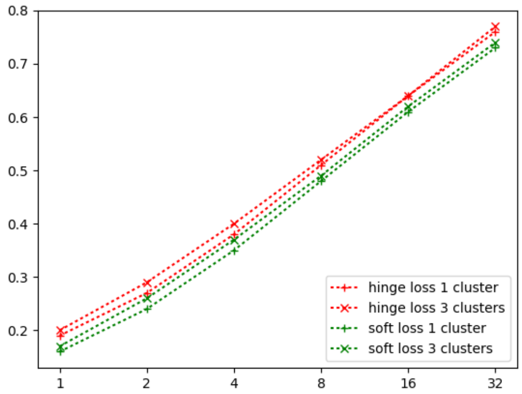

Figure 4. Recall@K score on test split.

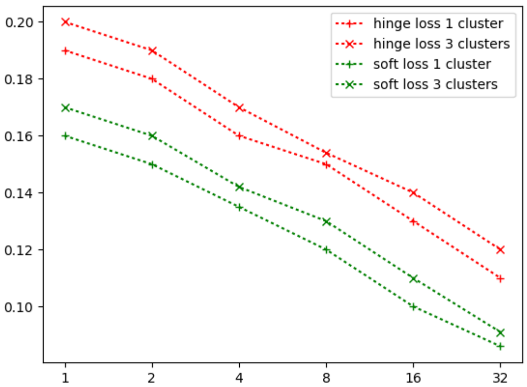

Figure 5. Accuracy for queried results vs. K, the number of returned images, on test split.

class are provided. For the three clusters per class case, we use three clusters per class for validation as well as final evaluation on test data. For the case with three clusters per class, we also provide the results for local positive sampling, where triplets include only those positive-anchor pairs which are within 60 percentile of all possible positive-anchor pairs. We get the highest F_1 score for hinge loss — 0.17 when using 1 cluster per class, and 0.24, when using 3 clusters per class. These are much higher than the maximum score of 0.1 that we were able to get with conventional triplet approach. [16] has reported a score of 0.12 for contrastive loss, 0.16 for triplet loss, and around 0.2 for their proposed lifted structured loss, which are to our knowledge, state-of-the-art results published on the same data-set, and with a similar train/test split.

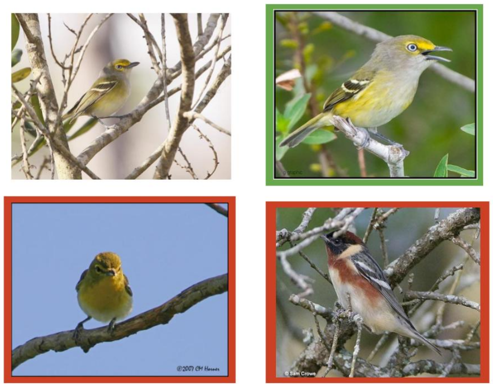

Figure 6. An image query gives one correct result, highlighted in green. Incorrect results are highlighted in red.

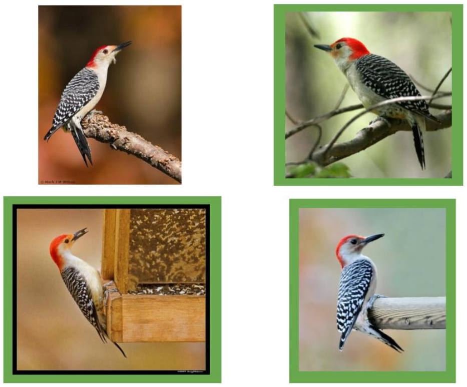

Figure 7. An image query returns results that are all correct! Incorrect results are highlighted in red.

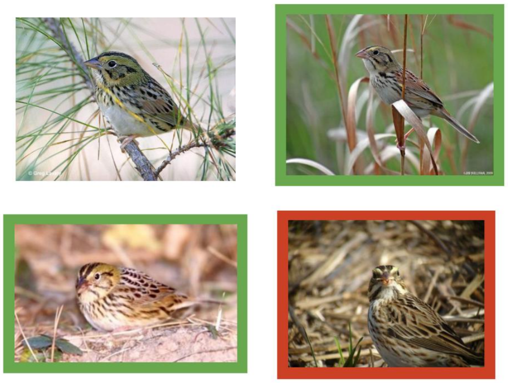

Figure 8. An image query returns 2 correct results. Incorrect results are highlighted in red.

8.2. Query Results

Similar to [16], Figure 4 gives Recall@K. K here is the number of images retrieved. These numbers are within a range of 0.2 from numbers for all different methods reported in [16]. Additionally, we computed the accuracy of query results — the fraction of correct results retrieved (images from the same class as query image). The accuracy of returned results drops with increase in K, indicating that as we expand the sphere around a query image, we tend to get more examples from other classes, which are close enough to the query image. This result agrees with the hypothesis in [8] that intra-class variance is often higher than inter-class variance. Finally, note that when we select the best model based on accuracy with K-means clustering with 3 clusters per class (on train/validation data), we get better Recall@K and accuracy scores, again indicating that it is useful to capture intra-class variance with multiple clusters per class.

Figures 6, 7 and 8 present some query results. The image in the top left corner in each case is the queried image. Correct results, that is, returned images that are from the same class as queried image, are highlighted in green, and incorrect images, that are images from a different class, are highlighted in red.

9. Conclusions and Discussion

Batch hard triplet approach significantly outperforms conventional triplet approaches, even when conventional approaches use hard mining. Batch hard approaches have more stable networks, and converge much faster than conventional approaches.

However, the accuracies for queried results even with batch hard approach, on test split (unknown classes) is only 10% to 20%, which still has a lot of room for improvement. One of the challenges with fine-grained classes and clustering and querying images from unknown classes is the high intra-class variance and low inter-class variance. To account for the variance, we experimented with local positive sampling and clustering with more than 1 cluster per class. While we did not see huge improvements with local positive sampling for batch hard approach, we did notice an improvement in accuracy when we use clustering accuracy for 3 clusters per class to drive the model selection, vs. 1 cluster per class.

One important issue with training triplet networks, that we noticed, is that clustering and querying accuracy can improve with extended training, even if the loss does not change much, as the network continues to learn from hard triplets, pushing hard negatives apart. This was also reported in [12], which recommends to continue training triplet networks even if the loss flattens.

We think it should be possible to design new sampling methods or loss functions, or add a classification loss using cluster centroids obtained using K-means clustering, to the total loss term, to drive the network to learn embeddings that map to extended manifolds rather than spheres, and capture intra-class variance, similar to [8], even for batch hard networks. It does not make sense to learn centroids or key points, as proposed in [8], since classes from test data are not known at training time. Computing cluster centroids however is expensive, and that limits how frequently we can update these centroids for the classification loss term.

Another interesting area of future work would be exploring other batch global loss functions, rather than pairs, triplets or quadruplets within a batch. Batch hard triplets already outperform conventional triplets, and if we are paying the cost of going over all $O(\frac{B^3}{C})$ triplets within a batch, we think it should be possible to design alternative loss functions over the entire batch that perform better. It may even be possible to design more efficient batch global loss formulations, using matrix operations. This is the idea behind lifted structured loss [16], and we think exploring alternatives is an interesting future direction. Even though forward and backward propagation for each batch becomes more expensive in these cases, networks converge with far fewer iterations, making these approaches more appealing than conventional contrastive or triplet approaches.

References

[1] `http://pytorch.org/`.

[2] B. Amos, B. Ludwiczuk, and M. Satyanarayanan. Openface: A general-purpose face recognition library with mobile applications. Technical report, Technical report, CMU-CS-16-118, CMU School of Computer Science, 2016.

[3] V. Balntas, E. Riba, D. Ponsa, and K. Mikolajczyk. Learning local feature descriptors with triplets and shallow convolutional neural networks. In *BMVC*, volume 1, page 3, 2016.

[4] S. Bell and K. Bala. Learning visual similarity for product design with convolutional neural networks. *ACM Transactions on Graphics (TOG)*, 34(4):98, 2015.

[5] J. Bromley, I. Guyon, Y. LeCun, E. Säckinger, and R. Shah. Signature verification using a" siamese" time delay neural network. In *Advances in Neural Information Processing Systems*, pages 737–744, 1994.

[6] 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.

[7] W. Chen, X. Chen, J. Zhang, and K. Huang. Beyond triplet loss: a deep quadruplet network for person re-identification. *arXiv preprint arXiv:1704.01719*, 2017.

[8] Y. Cui, F. Zhou, Y. Lin, and S. Belongie. Fine-grained categorization and dataset bootstrapping using deep metric learning with humans in the loop. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1153–1162, 2016.

[9] A. Dosovitskiy, P. Fischer, J. Springenberg, M. Riedmiller, and T. Brox. Discriminative unsupervised feature learning with exemplar convolutional neural networks, arxiv preprint. *arXiv preprint arXiv:1506.02753*, 2015.

[10] 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.

[11] R. Hadsell, S. Chopra, and Y. LeCun. Dimensionality reduction by learning an invariant mapping. In *Computer vision and pattern recognition, 2006 IEEE computer society conference on*, volume 2, pages 1735–1742. IEEE, 2006.

[12] A. Hermans, L. Beyer, and B. Leibe. In defense of the triplet loss for person re-identification. *arXiv preprint arXiv:1703.07737*, 2017.

[13] E. Hoffer and N. Ailon. Deep metric learning using triplet network. In *International Workshop on Similarity-Based Pattern Recognition*, pages 84–92. Springer, 2015.

[14] H. Jegou, M. Douze, and C. Schmid. Product quantization for nearest neighbor search. *IEEE transactions on pattern analysis and machine intelligence*, 33(1):117–128, 2011.

[15] C. D. Manning, P. Raghavan, H. Schütze, et al. *Introduction to information retrieval*, volume 1. Cambridge university press Cambridge, 2008.

[16] H. Oh Song, Y. Xiang, S. Jegelka, and S. Savarese. Deep metric learning via lifted structured feature embedding. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4004–4012, 2016.

[17] F. Schroff, D. Kalenichenko, and J. Philbin. Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 815–823, 2015.

[18] H. Shi, Y. Yang, X. Zhu, S. Liao, Z. Lei, W. Zheng, and S. Z. Li. Embedding deep metric for person re-identification: A study against large variations. In *European Conference on Computer Vision*, pages 732–748. Springer, 2016.

[19] E. Simo-Serra, E. Trulls, L. Ferraz, I. Kokkinos, P. Fua, and F. Moreno-Noguer. Discriminative learning of deep convolutional feature point descriptors. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 118–126, 2015.

[20] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2818–2826, 2016.

[21] C. Wah, S. Branson, P. Welinder, P. Perona, and S. Belongie. The caltech-ucsd birds-200-2011 dataset. 2011.

[22] 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.

[23] K. Q. Weinberger and L. K. Saul. Distance metric learning for large margin nearest neighbor classification. *Journal of Machine Learning Research*, 10(Feb):207–244, 2009.