Compressing Deep Neural Networks for Recognizing Places

Soham Saha, Girish Varma, C.V.Jawahar
Center for Visual Information Technology, IIIT Hyderabad, India
soham.saha@research.iiit.ac.in, girish.varma@iiit.ac.in, jawahar@iiit.ac.in

Abstract

Visual place recognition on low memory devices such as mobile phones and robotics systems is a challenging problem. The state of the art models for this task uses deep learning architectures having close to 100 million parameters which takes over 400MB of memory. This makes these models infeasible to be deployed in low memory devices and gives rise to the need of compressing them. Hence we study the effectiveness of model compression techniques like trained quantization and pruning for reducing the number of parameters on one of the best performing image retrieval models called NetVLAD. We show that a compressed network can be created by starting with a model pre-trained for the task of visual place recognition and then fine-tuning it via trained pruning and quantization. The compressed model is able to produce the same mAP as the original uncompressed network. We achieve almost 50% parameter pruning with no loss in mAP and 70% pruning with close to 2% mAP reduction, while also performing 8-bit quantization. Furthermore, together with 5-bit quantization, we perform about 50% parameter reduction by pruning and get only about 3% reduction in mAP. The resulting compressed networks have sizes of around 30MB and 65MB which makes them easily usable in memory constrained devices.

1. Introduction

Recent approaches in solving pattern recognition problems are focused on deep learning techniques. Visual place recognition in computer vision is one such task, which is typically cast as an image retrieval problem, where a database of images is queried using a query image and the system produces relevant images. Some of the major challenges for the visual place recognition task is to correctly retrieve the image of the desired place even though it has a different perspective or is under a different ambient lighting. Thus we would want our retrieval system to be scale and light invariant.

Traditionally, this problem is solved by extracting feature vectors using the BOF or SIFT representations and finding the approximate nearest neighbours of the query image in this feature space. Another popular method, the vector of locally aggregated descriptors (VLAD) was originally proposed by Jegov et al. [10] as an improvement over BOF representation of images and the Fisher kernel vector representation. This method essentially provides a vector representation of images and is quite similar to the Fisher kernel representation. It aggregates the descriptors based on a locality criterion in the feature space. Arandjelovic et al. [2] improved on VLAD by introducing vocabulary adaptation and intra-normalization. Over the years, contributions have been made on quality improvement of the aggregated features, optimizing the indexing scheme and other modifications for improving the performance of VLAD in [16], [3].

Starting with the work of Krizhevsky et al. [11], there has been a significant improvement in the performance of various computer vision tasks by using deep neural networks. For the task of image retrieval, Arandjelović et al. [1] proposed a deep learning model that can be trained in an end to end fashion. Their architecture consists of the convolutional layers of popular deep image classification models like Alexnet or VGG16 (Simonyan et al. [15]). This is followed by a NetVLAD layer (see Section 2) which has trainable parameters unlike the original VLAD. Finally, they also have a PCA whitening layer that reduces the dimension of the feature vectors.

The recent trend for improving the performance of computer vision tasks has been to train deep learning models with an increased number of parameters. For example the popular Alexnet has 60 million parameters and requires about 240 MB of memory while VGG16 has 130 million parameters and has takes around 500 MB of memory. However, these models need to be deployed in memory-constrained devices like a mobile phone or a robotics system. Hence, there is a need for reducing the size of the models without deteriorating the performance as well as reducing the test-time evaluation (Denton et al.[4]). In this work, we address the former.

Gong et al. [5] studied some of the standard quantization approaches for storing network weights in compressed format after the training process is completed. Through parameter pruning, we want to remove those parameters which do not have an adverse effect on performance of task at hand. This method of avoiding incorrect pruning was proposed by Geo et al.[6] where they implement on-the-fly connection pruning followed by splicing. Recently, Han et al. [8] proposed a combination of pruning of weights and quantizing them during the training process itself since such huge models are likely to contain many redundant parameters. They further reported the energy savings in the compressed networks in [7].

In our work, we study the effectiveness of trained pruning and quantization methods (see Section 3) proposed by Han et al., for compressing the NetVLAD model (see Section 2) of Arandjelović et al. [1]. We demonstrate our results on the Oxford and Paris buildings datasets (see Section 4). We implement trained pruning and trained quantization iteratively while fine-tuning the original network. This makes the network drop the redundant parameters as well as allows it to fine tune the other parameters so that the network can learn to deliver a similar performance as before by making do with the existing parameters. Such a compact network finds importance in enabling better performance in small memory devices. While performing 8-bit quantization, we can achieve about 50% reduction in parameters (factor of 8X) with same performance and about 70% reduction (factor of 12X) with just about 2% reduction in performance (see Section 5).

Han et al. [8] performed pruning, quantization as well as Huffman encoding to achieve a compression ratio of 35X. However, Huffman encoding is only useful while saving to disk. The network needs to be uncompressed before doing a forward pass during testing. In this work, we are more concerned about the memory required during the forward pass and do not apply the Huffman encoding technique. Hence our compression rate of 12X is lower that the 35X as claimed by Han et al. [8]. Moreover, we are doing it for the task of image retrieval rather than for classification as in the case of Han et al. [8].

2. NetVLAD

Vector of Locally Aggregated Descriptors (VLAD) is a well-known method used for instance level retrieval. Here we first compute a feature descriptor for each image and define k cluster centers from among the descriptors. This is followed by the computation of sum of residuals i.e the difference between the feature descriptor for each image and each of the cluster centers. Thus, if we have a d dimensional feature descriptor, we will have k residues each of d dimension. These vectors are then concatenated resulting in a $k \times d$ length descriptor for each image. This is followed by vocabulary adaptation and intra normalization, the details of which are mentioned in [2].

The VLAD comprises a series of operations which were presented in the form of a Generalized VLAD layer for neural nets called NetVLAD by Arandjelović et al. [1]. The input to the NetVLAD layer (denoted by x_i where i is a spatial location) is the convolutional features of dimension $H \times W \times D$ which is considered as $N = H \times W$ D-dimensional vectors. It has some trainable parameters W, b and cluster centers C having dimensions $K \times D$, K and $K \times D$ respectively. The output feature vector denoted by V (of dimension $K \times D$) is given by the following equation:

$$V(j,k) = \sum_{i=1}^{N} \left(\frac{e^{w_k^T x_i + b_k}}{\sum_{k'} e^{W_{k'}^T + b_{k'}}} \right) (x_i(j) - c_k(j)) \qquad (1)$$

Figure 1 shows NetVLAD layer.

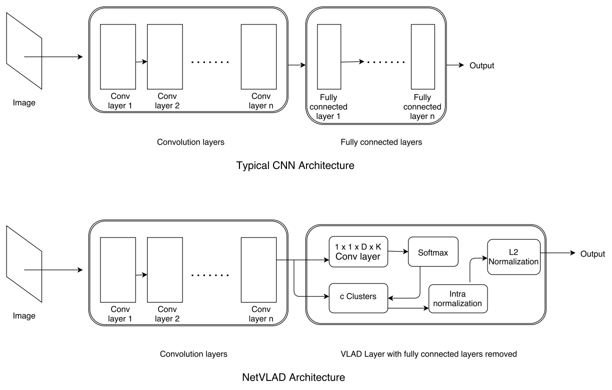

Figure 1. CNN + Fully Connected Layer (top) and NetVLAD architecture (bottom)

2.1. Triplet loss

The standard loss function used for retrieval problems is the triplet loss. This loss takes as inputs the feature vectors corresponding to a query image as well as a positive and negative vector corresponding to that query. In our experiments, for each query, the positive example is a relevant image (typically a image of the same building with a different perspective or lighting conditions) and a negative example is randomly sampled from the rest. The triplet loss is defined in equation 2.

$$L(q,p,n) = \frac{1}{2} h \left(m + \|q-p\|^2 - \|q-n\|^2 \right) \qquad (2)$$

Here, q denotes the query, p is a positive example for q and n is a negative example for q. m denotes the margin which is usually taken to be 1. h denotes the hinge loss where $h(x) = max(0, x)$. Figure 2 illustrates how this loss affects the feature vectors while training.

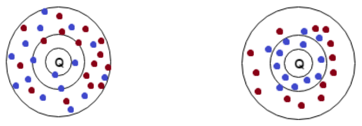

Figure 2. Toy image showcasing the distribution of the samples before and after the fine tuning using triplet loss. The red/blue dots indicate negative/positive examples. The innermost circle resembles the boundary for the Query image. The outermost circle is imaginary resembling the maximum distance a sample image can be from the query. The middle circle is indicative of the margin which we use to separate the positive samples from the negative samples.

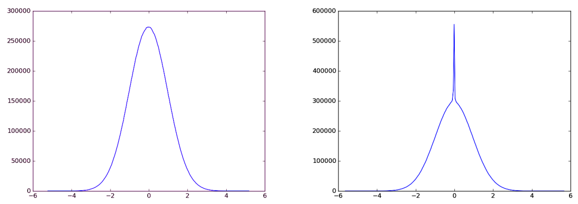

Figure 3. Distribution of weights before and after regularization. X-axis denotes the values of weights while Y-axis denotes the frequency of weights.

3. Compression

We perform *trained pruning* and *trained quantization* iteratively on some networks for the task of visual place recognition and report the performance on well known datasets. We explain these compression techniques in the subsequent sections.

3.1. Trained Network Pruning

Network Pruning has been widely used for compressing neural nets as shown by LeCun et al. [12]. They help to reduce the number of parameters thereby reducing complexity and avoiding redundancy.

We build on top of these approaches. Our aim is to drop all connections which have a value less than some threshold. We start with a model which has been trained to a desired precision on the visual place recognition dataset using the architecture mentioned in Section 2. Now, we must ensure that the network learns to do its task even after the parameters have been dropped, and in order to maximize it we must make the network learn in such a way that more parameters are closer to zero. In other words, the weight matrix should be sparse. In order to achieve that, we add a ℓ_1 regularizer to the weights which ensures that sparse matrices are learnt. This works better than using ridge regression as regularization i.e using a ℓ_2 regularizer. These have been demonstrated in Tibshirani et al. [17] and Hoerl et al.[9]. Thus, we add the ℓ_1 regularizer to every layer in our network.

The weight regularization can be framed as follows:

$$J(\theta) = Loss + \lambda \sum_{l=1}^{n_l - 1} \sum_{i=1}^{s_l} \sum_{j=1}^{s_{l+1}} |W_{ji}^l| \qquad (3)$$

where λ is a tuning parameter and can be considered as a hyperparameter, l is the layer number in the neural network architecture, n_l is the number of layers and s_l is the number of hidden units in the lth layer. j, i are indices of the weight matrix W in the lth layer. The effect of adding the regularizer can be viewed in Figure 4

After training the network by backpropagating the gradients from the loss function for some time, we prune the network connections below a threshold and retrain the network using the new modified loss function for a few epochs. This is followed by trained quantization which is described in the next section. This iterative process of pruning weights, quantizing and retraining is repeated for a certain number of epochs until the loss stabilizes. We ensure pruning by altering the gradients for the already pruned parameters to zero after every pruning operation. If any other parameters enter the threshold as a result of the gradient updates, then they are also pruned in the next iteration. Retraining the network ensures that the other parameters are able to compensate for the drop of the existing parameters. The parameters which are not pruned are able to adapt accordingly after realizing that the pruned parameters are non-existent.

An important hyperparameter which we need to fix is the threshold. Let us denote it by θ. This is significant because it determines the range in which weights will be dropped. Since our aim is to make the weight matrix as sparse as possible without compromising on the original mAP, we experiment with several values of θ ranging from 0.1 to 0.001 as long as it sparsifies the weight matrix without loss in mAP. We prune the weights according to the following rule.

$$W_i = \begin{cases} W_i & W_i < -\theta \\ 0 & -\theta \leq W_i \leq \theta \\ W_i & \theta < W_i \end{cases} \qquad (4)$$

where W is the weight matrix and θ is the threshold value.

3.2. Trained Quantization

This step is carried out after every pruning step. The way in which we compress the network is by reducing the number of bits required to represent each weight. The trained pruning step is accompanied by weight sharing where we reduce the number of effective weights being stored, by having multiple connections share the same weight.

The steps for trained quantization are listed below:

1. Perform Network pruning on the already trained network as mentioned in Section 3.1

2. Use k-means clustering for non-uniform quantization on the weights. For 8-bit non uniform quantization we shall get 256 cluster centers.

3. Now, for each time we update the weights, we accumulate the gradients having the same cluster indices, and add them up. Then we add these gradient cluster centers to the original codebook of cluster centers. This codebook is what we need to store in memory.

4. At the end, we replace the original weights with these 256 distinct updated cluster centers.

   Figure 4 gives an overview of the process.

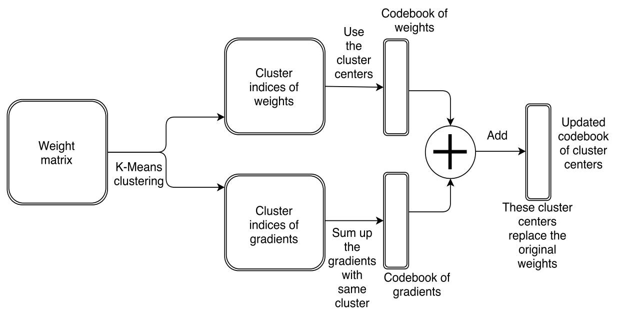
Weight matrix
K-Means clustering
Cluster indices of weights
Cluster indices of gradients
Use the cluster centers
Codebook of weights
Sum up the gradients with same cluster
Codebook of gradients
Add
Updated codebook of cluster centers
These cluster centers replace the original weights

Figure 4. Steps involved in Trained Quantization

4. Dataset and Experiments

We use transfer learning for our experiments i.e we use a pretrained network which has been trained on the visual place recognition task and we fine tune it on *one of Oxford and Paris Buildings* dataset and *test on the other*. This is helpful because it uses the knowledge of a previously learnt task and uses it for a similar task, but on different data. This fine tuning is required because it lets the network learn how to perform the visual place recognition task with less number of parameters.

We test our results in two ways and report the same. First, we fine tune on Oxford Buildings dataset and validate the performance of our method by using the Paris Buildings dataset. Then, we repeat this process by using Paris Buildings for fine tuning and Oxford Buildings for testing. This approach is justified due to the following reason. There are 55 queries in each of these datasets. Had we made a split of one dataset say into fine tuning and testing query sets, and reported our performance on that, it would not have reflected a realistic scenario. This is because once the model has been deployed on a mobile phone or any other memory constrained device for that matter, there is no further scope of fine tuning. Thus, recording the performance on a completely new dataset such as that which has not been seen by the network beforehand makes much more sense.

We study the effect of the compression techniques mentioned in Section 3 on the following pretrained NetVLAD models Arandjelović et al.[1].

1. Alexnet + NetVLAD : Consists of the convolutional layers of Alexnet (pretrained on Imagenet), followed by the NetVLAD layer and a whitening layer, pretrained on Pittsburgh 30k dataset.

2. VGG16 + NetVLAD : Consists of the convolutional layers of VGG16 (pretrained on Imagenet), followed by the NetVLAD layer and a whitening layer, pretrained on Pittsburgh 30k dataset.

We report our results on the following datasets

1. Oxford 5K : Images of Oxford buildings [13].

2. Paris 6K : Images of Paris buildings [14].

Both the datasets, contain search results corresponding to 55 query images. There are 'good' and 'ok' images in each dataset which are considered as positive images for a query in our triplet loss function. The 'junk' images and other non-positive images for each query are considered as negative images. As mentioned above we use one of these for fine tuning and the other for testing.

We observe the change in MAP for several values of threshold which results in different number of parameters being discarded (Figure 5). From such a plot, it is easier for us to determine what optimum value we should select for the θ.

Table 1 enlists the results of trained pruning and quantization on aforementioned networks which we used.

5. Results and Discussion

We show that our method is able to reduce the memory usage of the network with negligible loss in precision. We are able to prune close to 50% of the parameters with no loss in mAP and almost 70% parameters with only 2% drop in mAP while fine tuning with 8-bit quantization. Overall, this corresponds to 8X and 12X compression rates respectively. Also, fine tuning with 5-bit quantization allows us to attain about 50% parameter reduction with about 3% mAP drop and this corresponds to about 12X compression rate. These results are reported in Table 1.

5.1. Alexnet+NetVLAD

We evaluate the performance of our compressed network by fine-tuning with Oxford Buildings dataset and testing our performance using Paris buildings dataset and vice-versa. We report the mAP and the corresponding plots. In this architecture, there are 5 convolutional layers followed by a NetVLAD layer. Additionally, a whitening operation after

Method	Threshold for pruning	Percentage of Parameters Pruned	Drop in MAP (Oxford Buildings)	Drop in MAP (Paris Buildings)	Memory usage (MB)
Alexnet + NetVLAD + whitening (base model)	0	0	0%	0%	248.6
8 bits quantization	0.001	25.77	0%	0%	41.4
	0.005	48.44	0%	0%	32.4
	0.01	69.92	2.1%	1.8%	20.0
	0.05	85.77	14.2%	13.3%	10.3
5 bits quantization	0.005	52.39	2.9%	3.4%	19.5
	0.01	74.95	7.3%	6.7%	10.6
VGG16 + NetVLAD + whitening (base model)	0	0	0%	0%	529.5
8 bits quantization	0.001	25.52	0%	0%	89.6
	0.005	51.77	0%	0%	65.1
	0.01	68.23	2%	2.1%	40.5
	0.05	84.68	11.8%	14.1%	21.7
5 bits quantization	0.005	55.77	2.2%	3.6%	42.1
	0.01	75.66	6.8%	5.6%	21.2

Table 1. Compression results on Alexnet and VGG16 with NetVLAD and PCA whitening pre-trained on Pittsburgh30k dataset. The time taken per query is about 0.31 seconds in each case. Arandjelović et al. reports the MAP for Oxford Buildings and Paris Buildings as 69.8% and 76.5% respectively with Alexnet+NetVLAD and 71.6% and 79.7% respectively with VGG16 + NetVLAD which we use as a baseline for our experiments.

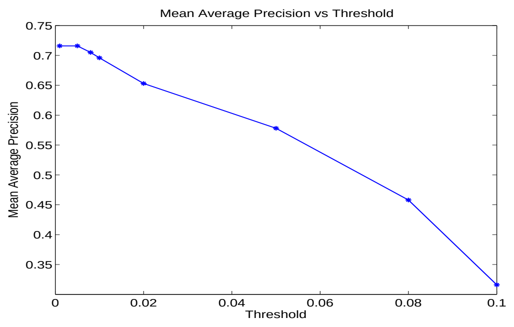

Figure 5. This plot shows the Drop in MAP (final performance) with the variation in Threshold used for pruning in VGG + NetVLAD on the Oxford Buildings dataset.

the NetVLAD layer is used. We show that the original network size is 248.6 MB and it can be compressed to 32.4 MB without loss in precision. Furthermore, we show that it can be compressed to about 20 MB with only around 2% loss in precision. Both of these are achieved while performing 8-bit quantization. These compression ratios enable the network to be used in mobile applications and real-time systems.

5.2. VGG16 + NetVLAD

We also look at compressing VGG16 + NetVLAD network since the bigger size of this network requires for it to be compressed even more in order to be used in a mobile application. This network with PCA whitening has a size of about 528 MB. However, we compress the network to about 65 MB without any loss in precision. The network can further be compressed to about 40 MB with about 2% loss in precision. The overall compression achieved is 12X with only 2% loss in performance.

It should be observed that when we are pruning the network an important hyperparameter is the value of θ used for pruning. This determines what percentage of weights will be discarded in the network and eventually has an impact on the final performance of the network. We observe that the mAP reduces drastically with increasing threshold value in Figure 5. This result also holds true intuitively since a higher threshold value means that more number of parameters are dropped according to Equation 2.

The plots for the experimental results are shown in Figure 5.

6. Conclusion

Deep learning models have improved the accuracy of various pattern recognition and machine learning tasks. However, the improvements were achieved by using models with increasingly larger number of parameters, making them infeasible to be run in memory constrained devices. Hence, the problem of compressing these models assumes significance. In this work, we studied the effect of model compression techniques like quantization and pruning for the task of visual place recognition. We succeed in reducing the model size from 248.6 MB to 32 MB and from 529.5 MB to 65.1 MB without any drop in MAP. This enables our model to be be run locally on memory-constrained devices rather than sending the image to a server, thereby avoiding additional latency.

From our experiments, we show that a greater percentage of parameters can be pruned with 5-bit quantization primarily by retraining the network iteratively. Also, this iterative retraining of the network with pruning and quantization is significant as it allows the un-pruned network parameters to

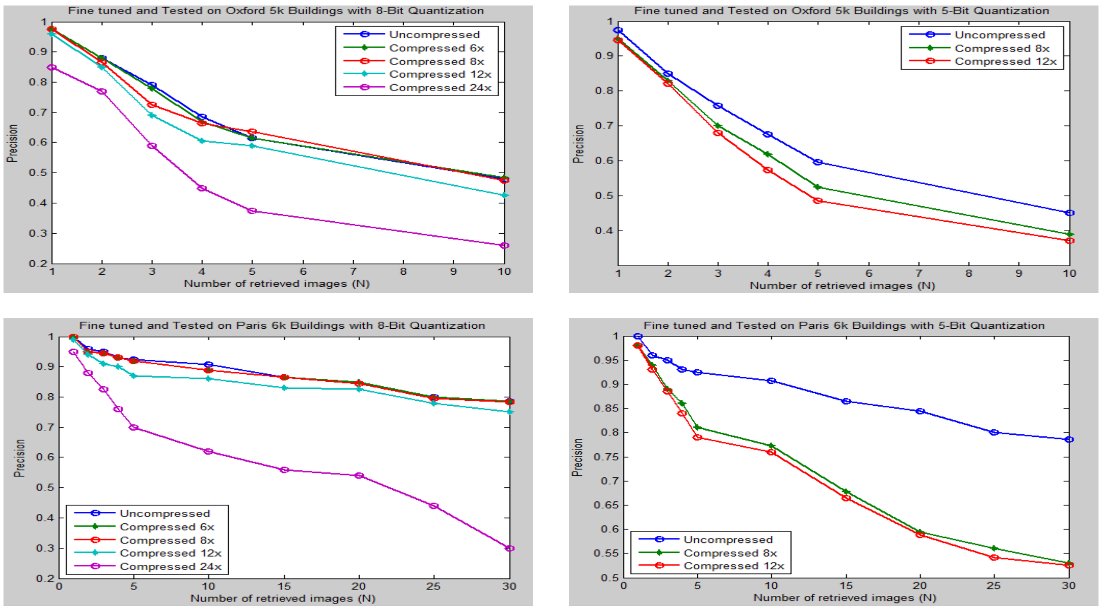

Figure 6. Results for VGG16 + NetVLAD + whitening pre-trained on Pittsburgh30k dataset. Plots contain the corresponding titles.

adapt themselves to the parameter modifications.

References

[1] R. Arandjelović, P. Gronat, A. Torii, T. Pajdla, and J. Sivic. NetVLAD: CNN architecture for weakly supervised place recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2016. 1, 2, 4

[2] R. Arandjelovic and A. Zisserman. All about vlad. In *Proceedings of the 2013 IEEE Conference on Computer Vision and Pattern Recognition*, CVPR '13, pages 1578–1585, Washington, DC, USA, 2013. IEEE Computer Society. 1, 2

[3] J. Delhumeau, P.-H. Gosselin, H. Jégou, and P. Pérez. Revisiting the vlad image representation. In *Proceedings of the 21st ACM international conference on Multimedia*, pages 653–656. ACM, 2013. 1

[4] E. L. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus. Exploiting linear structure within convolutional networks for efficient evaluation. In *Advances in Neural Information Processing Systems*, pages 1269–1277, 2014. 1

[5] Y. Gong, L. Liu, M. Yang, and L. Bourdev. Compressing deep convolutional networks using vector quantization. *arXiv preprint arXiv:1412.6115*, 2014. 1

[6] Y. Guo, A. Yao, and Y. Chen. Dynamic network surgery for efficient dnns. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 1379–1387. Curran Associates, Inc., 2016. 2

[7] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally. Eie: Efficient inference engine on compressed deep neural network. In *International Symposium on Computer Architecture (ISCA 2016)*, 2016. 2

[8] S. Han, H. Mao, and W. J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. In *International Conference on Learning Representations (ICLR'16 best paper award)*, 2015. 2

[9] A. E. Hoerl and R. W. Kennard. Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 12(1):55–67, 1970. 3

[10] H. Jegou, F. Perronnin, M. Douze, J. Sanchez, P. Perez, and C. Schmid. Aggregating local image descriptors into compact codes. *IEEE Trans. Pattern Anal. Mach. Intell.*, 34(9):1704–1716, Sept. 2012. 1

[11] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, page 2012. 1

[12] Y. LeCun, J. S. Denker, and S. A. Solla. Optimal brain damage. *Advances in neural information processing systems 2, NIPS 1989*, 2:598–605, 1990. 3

[13] J. Philbin, O. Chum, M. Isard, J. Sivic, and A. Zisserman. Object retrieval with large vocabularies and fast spatial matching. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2007. 4

[14] J. Philbin, O. Chum, M. Isard, J. Sivic, and A. Zisserman. Lost in quantization: Improving particular object retrieval in large scale image databases. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2008. 4

[15] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. 1

[16] E. Spyromitros-Xioufis, S. Papadopoulos, I. Kompatsiaris, G. Tsoumakas, and I. Vlahavas. A comprehensive study over vlad and product quantization in large-scale image retrieval. *IEEE Transactions on Multimedia*, 2014. 1

[17] R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 267–288, 1996. 3