ExchNet: A Unified Hashing Network for Large-Scale Fine-Grained Image Retrieval (Supplementary Materials)

Anonymous ECCV submission

Paper ID 6502

In the supplementary materials, we supply implementation details, retrieval performance of different settings and pseudo codes of the learning algorithm for optimizing ExchNet.

1. Implementation details of approximate nearest neighbor methods and other hashing methods;
2. Retrieval performance and query time of different settings on *Food101*;
3. Pseudo codes of the learning algorithm for optimizing ExchNet.

1 Implementation details of approximate nearest neighbor methods and other hashing methods

In this section, we elaborate the experimental details of approximate nearest neighbor methods and other hashing methods.

For comparisons with other ANN methods (cf. Section 4.3 of the paper), we firstly utilize the triplet loss to learn feature embeddings. The employed backbone network is ResNet50 and semi-hard sampling is utilized for obtaining better retrieval accuracy. We train the network with the SGD optimizer with weight decay 5×10^{-4}. Embedding sizes are 512 and 1024. Initial learning rate is set to 1×10^{-5} and divided by 3 per 30 epochs. Based on the learned real-valued features and hash codes, we use the re-ranking technique to speed up fine-grained image retrieval. In general, we first utilize the learned binary codes to retrieve top N candidates for a database with n samples ($N \ll n$). Then, the re-ranking algorithm is used to re-rank these candidates. At last, top K ($K < N$) candidates are generated. For Ball Tree and KDTree, we first utilize principal component analysis to reduce the dimensionality of real-valued features to 32, and we set the number of leaf size as 40. For PQ, we adopt the inverted file system with asymmetric distance computation (IVFADC) and set the number of centroids as 300 and the code size as 8.

For comparisons with non-deep based hashing methods, we utilize 2,048-dim deep features extracted by ResNet50 pre-trained on ImageNet with 224×224 input resolutions. For SDH, we select 1,000 samples as kernels. For deep hashing methods, we replace the backbone of every method to ResNet50 and the hyper-parameters are set as recommended in their papers for fair comparisons.

2 Retrieval performance and query time of different settings on *Food101*

For comparisons with other ANN methods, we retrieve top N nearest neighbors by multiple hash tables lookup, where N is the number of candidates selected by the binary hash codes. The length of each table L is set to 8/16. In Table 1 of the supplementary materials, detailed retrieval accuracy and query time of different experimental settings are provided. As to results reported in Table 2 of the paper, N and L are 3000 and 16, respectively.

Table 1. Retrieval performances and times with different settings on *Food101*.

#Dim	Table Length	Precision@10				
		N=1000	N=2000	N=3000	N=4000	N=5000
512	$L = 8$	70.37%	75.80%	77.23%	78.00%	78.39%
	$L = 16$	73.11%	76.50%	77.69%	78.29%	78.63%
1,024	$L = 8$	70.67%	76.18%	77.62%	78.43%	78.76%
	$L = 16$	73.44%	76.86%	78.06%	78.64%	78.97%
		Query time				
512	$L = 8$	10.26	18.49	30.59	35.00	55.13
	$L = 16$	12.65	26.24	40.54	61.20	69.71
1,024	$L = 8$	15.30	32.64	44.54	66.94	86.43
	$L = 16$	17.34	44.38	56.57	75.80	150.86

3 Pseudo codes of the learning algorithm for optimizing ExchNet

In this section, we present the pseudo codes of the learning algorithm (cf. Section 3.4 of the paper) for optimizing our ExchNet. As shown in Algorithm 1, in each iteration, we first optimize the parameters Θ of a DNN when binary codes $\boldsymbol{V}$ and anchored local features $\boldsymbol{\mathcal{C}}$ fixed. Then, we update $\boldsymbol{V}$ when Θ and $\boldsymbol{\mathcal{C}}$ fixed. Finally, anchored local features $\boldsymbol{\mathcal{C}}$ are updated.

Algorithm 1: Learning algorithm for our proposed method.

Input: $\boldsymbol{X}$: image set; $\boldsymbol{y}$: image labels.
Output: Θ: parameters of DNN; $\boldsymbol{V}$: binary codes.

Initialization: initialize Θ, mini-batch size N.
for $i = 1 \mapsto T_{max}$ **do**
  `/* Update parameter` Θ. `*/`
  for $j = 1 \mapsto \#epochs$ **do**
    for $n = 1 \mapsto N$ **do**
      Sample a mini-batch samples from $\boldsymbol{X}$.
      Calculate $\hat{\boldsymbol{u}}_i$ by forward propagation.
      Calculate the gradient according to Equation (14).
      Update parameter Θ by back-propagation.
    end
  end
  `/* Update parameter` $\boldsymbol{V}$. `*/`
  for $k = 1 \mapsto q$ **do**
    Update $\boldsymbol{V}_{*k}$ according to update rule in Equation (17).
  end
  `/* Update parameter` $\boldsymbol{\mathcal{C}}$. `*/`
  for $\forall \mathcal{C}_i \in \boldsymbol{\mathcal{C}}$ **do**
    Update $\boldsymbol{C}_i$ according to update rule in Equation (18).
  end
end