Content Based Image Retrieval using Convolutional Neural Networks

Rohit Singhatwadia

14JE000323

Computer Science & Engineering

Indian Institute of Technology (ISM) Dhanbad

Abstract

We propose a novel technique for Content Based Image Retrieval (CBIR) using Convolutional Neural Networks (CNN). We make use of the powerful VGGNet pre-trained on the ImageNet dataset and extract the activations of the last convolutional layer for our feature vector representation. We apply the Principal Component Analysis (PCA) Transform to reduce the dimensionality of the feature vector which leads to a large reduction in retrieval time. Intra-class clustering is employed, the benefits of which are two-fold - retrieval time is reduced and the images retrieved are semantically more meaningful. We show our results on five datasets - Pascal VOC 2012, CIFAR-10, Corel-10K, GHIM-10K and Corel-1K and achieve a significant increase in classification accuracy and a reduction in retrieval time.

Content Based Image Retrieval Using Convolutional Neural Networks

Content-based image retrieval (CBIR), also known as query by image content (QBIC) and content-based visual information retrieval (CBVIR) is the application of computer vision techniques to the image retrieval problem, that is, the problem of searching for digital images in large databases. Content-based image retrieval is opposed to traditional concept-based approaches. "Content-based" means that the search analyzes the contents of the image rather than the metadata such as keywords, tags, or descriptions associated with the image. The term "content" in this context might refer to colors, shapes, textures, or any other information that can be derived from the image itself. CBIR is desirable because searches that rely purely on metadata are dependent on annotation quality and completeness. Having humans manually annotate images by entering keywords or metadata in a large database can be time consuming and may not capture the keywords desired to describe the image. The evaluation of the effectiveness of keyword image search is subjective and has not been well-defined. In the same regard, CBIR systems have similar challenges in defining success.

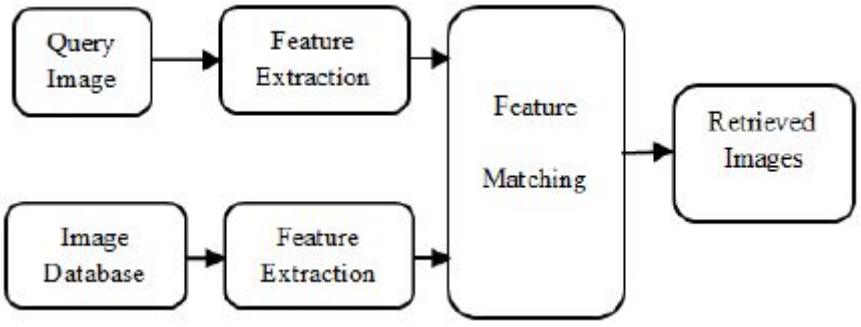


Figure 1. CBIR

Literature Review

In the past considerable amount of work has been done in the area of clustering of images. Thomas Deselaers, Daniel Keysers and Hermann Ney worked on clustering of visually similar images to improve the the results of image search engines, for this task they

used features invariant against translation and rotation to represent the content of the images and k-means and LGB cluster algorithms to present the images in a convenient manner to the user. Guoping Qiu worked on image and feature co-clustering for this task the author makes use of a computational energy function suitable for co-clustering images and their features and then optimizes this function using hopfield model based stochastic algorithm. Brenden J Frey and Delbert Dueck designed a method called affinity propagation. In this method similarity measure between a pair of data points is taken as input and then messages are exchanged between data points to form high quality clusters. Giridharan Iyengar and Andrew B Lippman worked on clustering images for efficient retrieval using relative entropy. They make an assumption that visual features are represented by probability densities and based on this assumption they design a clustering algorithm for probability densities. VSVS Murthy, E Vamshidhar, JNVR Swarup Kumar and P Shankara Rao extract color feature from images and combine hierarchical clustering algorithm with K-means to cluster images to implement content based image retrieval.

Convolutional Neural Networks

In machine learning, a convolutional neural network (CNN, or ConvNet) is a class of deep, feed-forward artificial neural networks that has successfully been applied to analyzing visual imagery. CNNs use a variation of multilayer perceptrons designed to require minimal preprocessing. They are also known as shift invariant or space invariant artificial neural networks (SIANN), based on their shared-weights architecture and translation invariance characteristics. Convolutional networks were inspired by biological processes in which the connectivity pattern between neuronsis inspired by the organization of the animal visual cortex. Individual cortical neurons respond to stimuli only in a restricted region of the visual

field known as the receptive field. The receptive fields of different neurons partially overlap such that they cover the entire visual field.

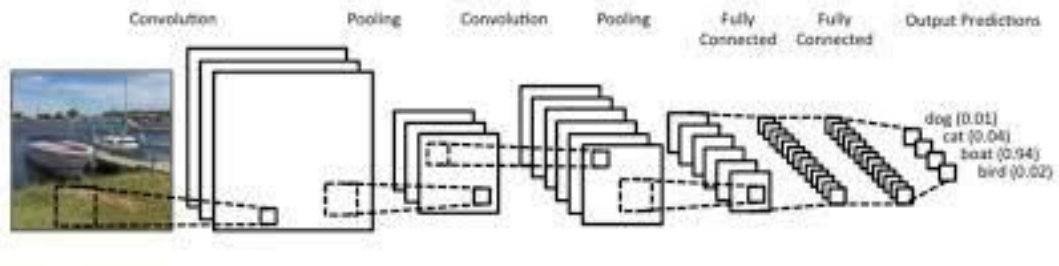


Figure 2. ConvNet Architecture

A CNN consists of an input and an output layer, as well as multiple hidden layers. The hidden layers of a CNN typically consist of convolutional layers, pooling layers, fully connected layers and normalization layers.

Description of the process as a convolution in neural networks is by convention. Mathematically it is a cross-correlation rather than a convolution. This only has significance for the indices in the matrix, and thus which weights are placed at which index.

Convolutional

Convolutional layers apply a convolution operation to the input, passing the result to the next layer. The convolution emulates the response of an individual neuron to visual stimuli.

Although fully connected feedforward neural networks can be used to learn features as well as classify data, it is not practical to apply this architecture to images. A very high number of neurons would be necessary, even in a shallow (opposite of deep) architecture, due to the very large input sizes associated with images, where each pixel is a relevant variable. For instance, a fully connected layer for a (small) image of size 100 x 100 has 10000 weights

for *each* neuron in the second layer. The convolution operation brings a solution to this problem as it reduces the number of free parameters, allowing the network to be deeper with fewer parameters. For instance, regardless of image size, tiling regions of size 5 x 5, each with the same shared weights, requires only 25 learnable parameters. In this way, it resolves the vanishing or exploding gradients problem in training traditional multi-layer neural networks with many layers by using backpropagation.

Pooling

Convolutional networks may include local or global pooling layers, which combine the outputs of neuron clusters at one layer into a single neuron in the next layer. For example, *max pooling* uses the maximum value from each of a cluster of neurons at the prior layer. Another example is *average pooling*, which uses the average value from each of a cluster of neurons at the prior layer.

Fully connected

Fully connected layers connect every neuron in one layer to every neuron in another layer. It is in principle the same as the traditional multi-layer perceptron neural network (MLP).

Weights

CNNs share weights in convolutional layers, which means that the same filter (weights bank) is used for each receptive field in the layer; this reduces memory footprint and improves performance.

Principal Component Analysis

Principal component analysis (PCA) is a statistical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables called principal components. If there are observations with p variables, then the number of distinct principal components is min(n-1,p). This transformation is defined in such a way that the first principal component has the largest possible variance (that is, accounts for as much of the variability in the data as possible), and each succeeding component in turn has the highest variance possible under the constraint that it is orthogonal to the preceding components. The resulting vectors are an uncorrelated orthogonal basis set. PCA is sensitive to the relative scaling of the original variables.

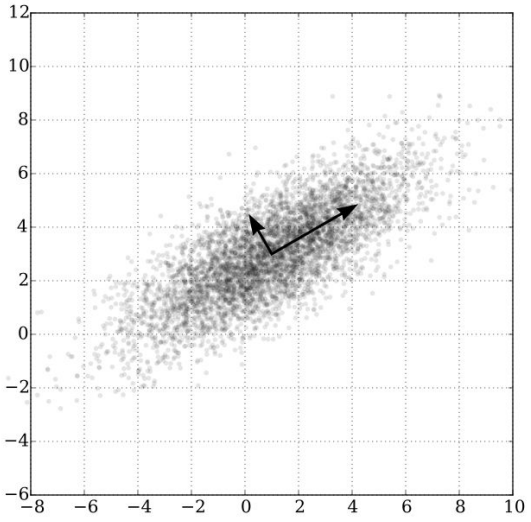


Figure 4. PCA of a multivariate Gaussian Distribution

PCA can be thought of as fitting an n-dimensional ellipsoid to the data, where each axis of the ellipsoid represents a principal component. If some axis of the ellipsoid is small, then the variance along that axis is also small, and by omitting that axis and its corresponding

principal component from our representation of the dataset, we lose only a commensurately small amount of information.

To find the axes of the ellipsoid, we must first subtract the mean of each variable from the dataset to center the data around the origin. Then, we compute the covariance matrix of the data, and calculate the eigenvalues and corresponding eigenvectors of this covariance matrix. Then we must normalize each of the orthogonal eigenvectors to become unit vectors. Once this is done, each of the mutually orthogonal, unit eigenvectors can be interpreted as an axis of the ellipsoid fitted to the data. This choice of basis will transform our covariance matrix into a diagonalised form with the diagonal elements representing the variance of each axis. The proportion of the variance that each eigenvector represents can be calculated by dividing the eigenvalue corresponding to that eigenvector by the sum of all eigenvalues. This procedure is sensitive to the scaling of the data, and there is no consensus as to how to best scale the data to obtain optimal results.

k-Means Clustering

k-means clustering is a method of vector quantization, originally from signal processing, that is popular for cluster analysis in data mining. k-means clustering aims to partition n observations into k clusters in which each observation belongs to the cluster with the nearest mean, serving as a prototype of the cluster. This results in a partitioning of the data space into Voronoi cells.

The problem is computationally difficult (NP-hard); however, there are efficient heuristic algorithms that are commonly employed and converge quickly to a local optimum. These are usually similar to the expectation-maximization algorithm for mixtures of Gaussian distributions via an iterative refinement approach employed by both k-means and Gaussian

Mixture Modeling. Additionally, they both use cluster centers to model the data; however, k-means clustering tends to find clusters of comparable spatial extent, while the expectation-maximization mechanism allows clusters to have different shapes.

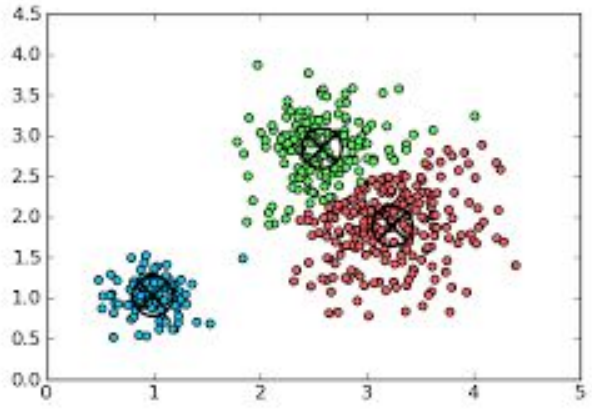


Figure 5. k-means clustering

The algorithm has a loose relationship to the k-nearest neighbor classifier, a popular machine learning technique for classification that is often confused with k-means because of the k in the name. One can apply the 1-nearest neighbor classifier on the cluster centers obtained by k-means to classify new data into the existing clusters. This is known as nearest centroid classifier or Rocchio algorithm.

Transfer Learning

In practice, very few people train an entire Convolutional Network from scratch (with random initialization), because it is relatively rare to have a dataset of sufficient size. Instead, it is common to pretrain a ConvNet on a very large dataset (e.g. ImageNet, which contains 1.2 million images with 1000 categories), and then use the ConvNet either as an initialization or a fixed feature extractor for the task of interest. The three major Transfer Learning scenarios look as follows:

ConvNet As Fixed Feature Extractor

Take a ConvNet pretrained on ImageNet, remove the last fully-connected layer (this layer's outputs are the 1000 class scores for a different task like ImageNet), then treat the rest of the ConvNet as a fixed feature extractor for the new dataset. In an AlexNet, this would compute a 4096-D vector for every image that contains the activations of the hidden layer immediately before the classifier. We call these features CNN codes. It is important for performance that these codes are ReLUd (i.e. thresholded at zero) if they were also thresholded during the training of the ConvNet on ImageNet (as is usually the case). Once you extract the 4096-D codes for all images, train a linear classifier (e.g. Linear SVM or Softmax classifier) for the new dataset.

Fine-tuning The ConvNet

The second strategy is to not only replace and retrain the classifier on top of the ConvNet on the new dataset, but to also fine-tune the weights of the pretrained network by continuing the backpropagation. It is possible to fine-tune all the layers of the ConvNet, or it's possible to keep some of the earlier layers fixed (due to overfitting concerns) and only fine-tune some higher-level portion of the network. This is motivated by the observation that the earlier features of a ConvNet contain more generic features (e.g. edge detectors or color blob detectors) that should be useful to many tasks, but later layers of the ConvNet becomes progressively more specific to the details of the classes contained in the original dataset. In case of ImageNet for example, which contains many dog breeds, a significant portion of the representational power of the ConvNet may be devoted to features that are specific to differentiating between dog breeds.

Pretrained Models

Since modern ConvNets take 2-3 weeks to train across multiple GPUs on ImageNet, it is common to see people release their final ConvNet checkpoints for the benefit of others who can use the networks for fine-tuning. For example, the Caffe library has a Model Zoo where people share their network weights.

We have employed the first technique of using the ConvNet as a fixed feature extractor in our CBIR system. We use the VGG16 architecture as given in Figure 3 with the network architecture weights pre-trained on the ImageNet dataset.

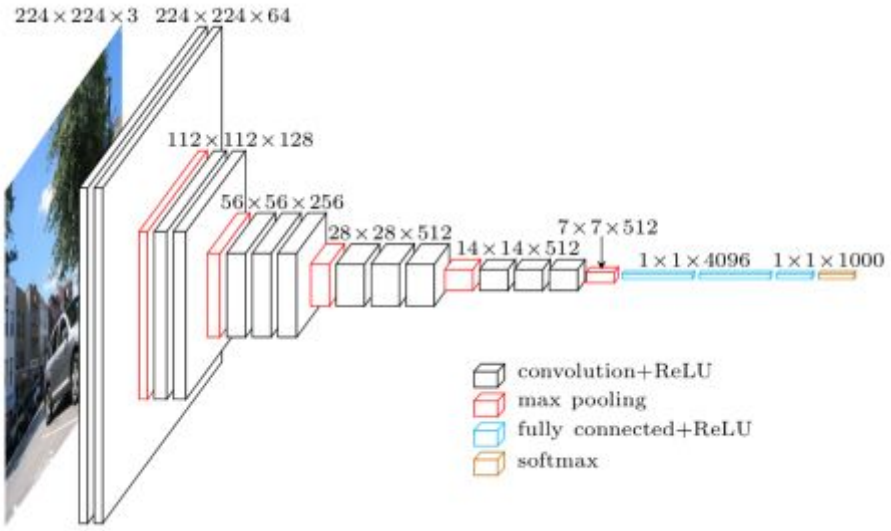
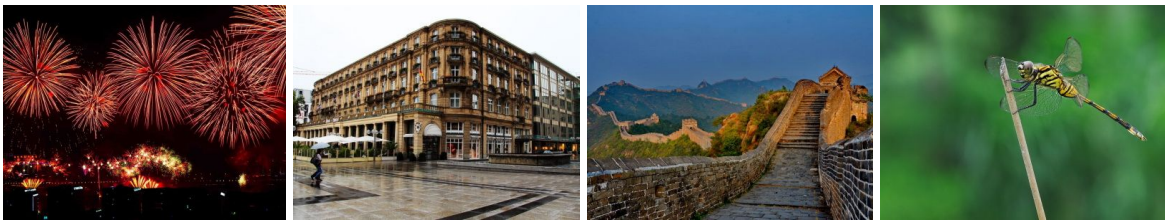


Figure 3. VGG16 Architecture

Datasets

Pascal VOC 2012

GHIM-10K



Corel-10K

CIFAR-10

Corel-1K

Proposed Method

Preprocessing/Training Algorithm

1) Split dataset into training set and testing set.

2) Perform one pass of the training set images via VGG16 network and store the activations of the last convolution layer. These are the original features.

3) Perform Principal Component Analysis to reduce the dimensions of the features.

4) Perform intra-class clustering via the K-Means algorithm, with K number of clusters.

5) For classification, train a Multi-Layer Perceptron for the features obtained in (3)

Testing Algorithm

1) Predict the class of the test image using the classification model.

2) Find the closest cluster centroid for the predicted class.

3) Report the top five images from the closest cluster as the results.

Experimental Results

We performed our experiments on a Ubuntu 16.04 system with a Nvidia GeForce 840M GPU. We show our results for four train-test splits: 60-40, 70-30, 80-20, 90-10 and show the classification accuracy for each split. We chose the number of intra-class clusters as $k = 3$.

Accuracy

Original	54.8	87.3	68.1	88.75
60-40	**Pascal**	**GHIM-10K**	**C10K**	**C1K**
50	51.53	80.35	59.6	85.75
75	53.16	83.47	62	85.25
100	54.06	84.8	64.68	84.5
125	53.65	86.08	64.02	84.5
150	53.97	85.8	64.02	86
200	53.35	86.32	66.35	84
300	53.47	86.62	66.53	84.25
400	54.48	87.65	67.13	85

Original	53.79	87.5	70.27	88
70-30	**Pascal**	**GHIM-10K**	**C10K**	**C1K**
50	52.91	81.3	60.67	87
75	54.12	83.4	63.17	86.67
100	54.9	84.23	65.6	85.33
125	55.52	85.23	65.33	86
150	53.65	85.8	66.4	86.33
200	55.17	86.1	66.97	85
300	53.56	86.23	68.33	83
400	52.49	86.73	68.1	83

Original	56.55	89.4	72.3	89.5
80-20	**Pascal**	**GHIM-10K**	**C10K**	**C1K**
50	54.01	81.15	63.7	87
75	53.86	83.5	65.85	86
100	53.6	86.35	67.45	85
125	55.41	87.35	67.9	86
150	55.24	87.45	69.05	87
200	54.39	88.15	69.1	84
300	55.09	88.75	71.05	84.5
400	54.92	86.6	71.1	83

Original	55.2	90.2	74.8	93
90-10	**Pascal**	**GHIM-10K**	**C10K**	**C1K**
50	54.1	84.9	63.4	89
75	54.29	85.7	66.5	87
100	53.88	85.8	68.5	91

125	54.46	87.6	70.1	88
150	54.23	88	69.4	91
200	53.88	88.3	70.2	87
300	54.29	88.5	71.5	89
400	54.23	88.2	70.7	86

Original	93.55
	CIFAR-10
50	93.67
75	93.74
100	93.64
125	93.7
150	93.7
200	93.7
300	93.65
400	93.59

Retrieval Time (Time in seconds)

Original	0.171	1.251	0.29	0.169	0.075	0.169	0.002	0.168	0.011
60-40	**Pascal**			**GHIM-10K**		**C10K**		**C1K**	
	Prediction	Clustering	Without Clustering	Prediction	Clustering	Prediction	Clustering	Prediction	Clustering
50	0.168	0.022	0.043	0.164	0.0018	0.164	0.0005	0.169	0.0006
75	0.167	0.023	0.044	0.166	0.0018	0.163	0.0005	0.164	0.0005
100	0.167	0.009	0.088	0.165	0.0019	0.163	0.0005	0.166	0.0005
125	0.168	0.02	0.045	0.166	0.002	0.168	0.0006	0.165	0.0005
150	0.167	0.025	0.044	0.166	0.0019	0.166	0.0005	0.165	0.0005
200	0.164	0.011	0.045	0.164	0.0019	0.16	0.0009	0.163	0.0008
300	0.167	0.027	0.049	0.166	0.0028	0.164	0.0009	0.165	0.001
400	0.167	0.03	0.048	0.165	0.0022	0.163	0.001	0.164	0.0008

Original	0.174	1.545	0.355	0.169	0.08	0.169	0.002	0.165	0.014

70-30	**Pascal**			**GHIM-10K**		**C10K**		**C1K**	
	Prediction	Clustering	Without Clustering	Prediction	Clustering	Prediction	Clustering	Prediction	Clustering
50	0.165	0.027	0.049	0.165	0.002	0.168	0.0006	0.165	0.0006
75	0.168	0.028	0.05	0.162	0.002	0.165	0.0005	0.164	0.0005
100	0.166	0.028	0.094	0.164	0.0023	0.166	0.0006	0.165	0.0006
125	0.164	0.03	0.048	0.165	0.0024	0.167	0.0006	0.164	0.0006
150	0.165	0.031	0.05	0.166	0.0025	0.17	0.001	0.166	0.0009
200	0.166	0.035	0.05	0.168	0.0024	0.17	0.0009	0.165	0.001
300	0.164	0.038	0.059	0.163	0.0029	0.165	0.001	0.166	0.001
400	0.164	0.02	0.054	0.162	0.0031	0.165	0.0009	0.164	0.001

Original	0.173	0.5	0.06	0.169	0.082	0.166	0.002	0.166	0.019
80-20	**Pascal**			**GHIM-10K**		**C10K**		**C1K**	
	Prediction	Clustering	Without Clustering	Prediction	Clustering	Prediction	Clustering	Prediction	Clustering
50	0.165	0.002	0.009	0.164	0.0022	0.166	0.0006	0.163	0.0005
75	0.166	0.003	0.008	0.164	0.0019	0.166	0.0006	0.164	0.0006
100	0.166	0.004	0.009	0.162	0.0031	0.167	0.0006	0.165	0.0007
125	0.163	0.004	0.009	0.166	0.0019	0.166	0.0009	0.166	0.0009
150	0.165	0.004	0.009	0.167	0.0022	0.167	0.001	0.166	0.001
200	0.167	0.004	0.009	0.166	0.0025	0.166	0.001	0.165	0.001
300	0.163	0.005	0.009	0.167	0.0028	0.167	0.0011	0.166	0.001
400	0.167	0.005	0.009	0.163	0.0027	0.167	0.0011	0.165	0.0011

Original	0.165	3.354	0.436	0.169	0.088	0.166	0.002	0.166	0.017
90-10	**Pascal**			**GHIM-10K**		**C10K**		**C1K**	
	Prediction	Clustering	Without Clusteri	Prediction	Clustering	Prediction	Clustering	Prediction	Clustering

			ng						
50	0.166	0.037	0.062	0.163	0.0019	0.166	0.0006	0.167	0.0006
75	0.168	0.012	0.064	0.164	0.0011	0.166	0.0005	0.166	0.0006
100	0.168	0.014	0.063	0.164	0.004	0.166	0.0005	0.165	0.0006
125	0.17	0.015	0.065	0.165	0.0034	0.168	0.0009	0.166	0.001
150	0.171	0.042	0.064	0.164	0.003	0.166	0.0009	0.166	0.001
200	0.169	0.036	0.062	0.171	0.0033	0.166	0.001	0.164	0.0012
300	0.168	0.023	0.069	0.166	0.0015	0.166	0.001	0.163	0.0012
400	0.165	0.056	0.069	0.165	0.005	0.167	0.001	0.165	0.0013

Original	0.167	0.03	0.044
	CIFAR-10		
	Prediction	Clustering	Without Clustering
50	0.166	0.009	0.043
75	0.165	0.01	0.04
100	0.171	0.01	0.04
125	0.166	0.012	0.04
150	0.164	0.013	0.041
200	0.164	0.014	0.04
300	0.166	0.016	0.045
400	0.166	0.019	0.045

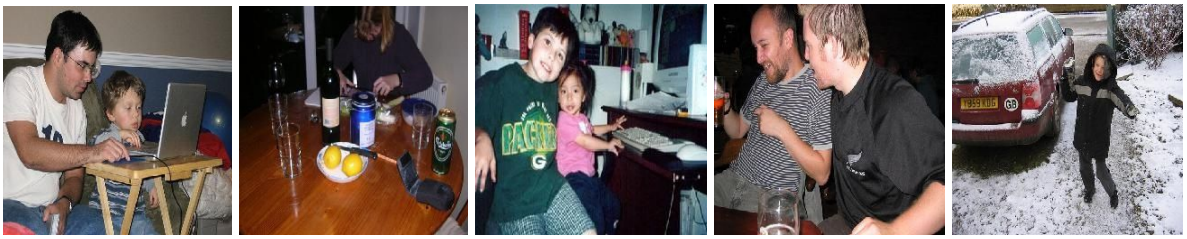
As evident from the table, we observe a significant reduction in the retrieval time compared to the prevalent method.

Now, we demonstrate that our approach is also successful in retrieving semantically meaningful retrieval results. Images that are much closer to the test image semantically are retrieved in our approach.

Test Image

Images Retrieved Without Clustering

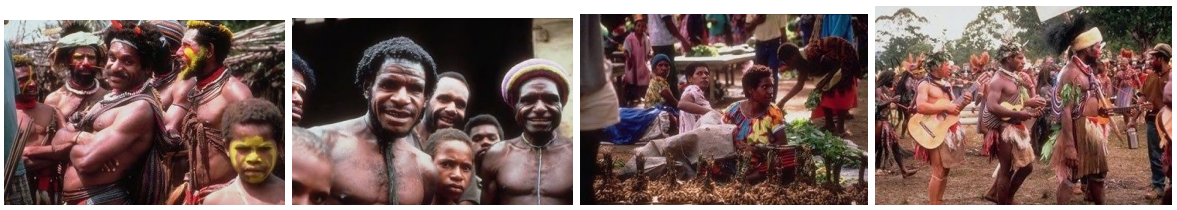
Images Retrieved With Clustering (Our Approach)



Test Image

Images Retrieved Without Clustering

Images Retrieved With Clustering (Our Approach)



Conclusion

We conclude that our model performs significantly faster than other models in literature due to dimensionality reduction and intra-class clustering. We observe that the benefits of intra-class clustering are twofold - it reduces the search space by a large extent and it aids in the retrieval of images that are semantically closer to the test image. We have demonstrated our technique on five datasets - Pascal VOC 2012, GHIM-10K, CIFAR-10, Corel-10K, Corel-1K and observe that our model performs significantly better on all datasets.

References

Lin, K., Yang, H. F., Hsiao, J. H., & Chen, C. S. (2015, June). Deep learning of binary hash codes for fast image retrieval. In *Computer Vision and Pattern Recognition Workshops (CVPRW), 2015 IEEE Conference on* (pp. 27-35). IEEE.

Fu, R., Li, B., Gao, Y., & Wang, P. (2016, October). Content-based image retrieval based on CNN and SVM. In *Computer and Communications (ICCC), 2016 2nd IEEE International Conference on* (pp. 638-642). IEEE.

Yu, W., Yang, K., Yao, H., Sun, X., & Xu, P. (2017). Exploiting the complementary strengths of multi-layer CNN features for image retrieval. *Neurocomputing*, *237*, 235-241.

Lin, K., Lu, J., Chen, C. S., & Zhou, J. (2016). Learning compact binary descriptors with unsupervised deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (pp. 1183-1192).

Gao, L., Song, J., Zou, F., Zhang, D., & Shao, J. (2015, October). Scalable multimedia retrieval by deep learning hashing with relative similarity learning. In *Proceedings of the 23rd ACM international conference on Multimedia* (pp. 903-906). ACM.

Alzu'bi, A., Amira, A., & Ramzan, N. (2017). Content-based image retrieval with compact deep convolutional features. *Neurocomputing*, *249*, 95-105.

Guo, Y., Liu, Y., Oerlemans, A., Lao, S., Wu, S., & Lew, M. S. (2016). Deep learning for visual understanding: A review. *Neurocomputing*, *187*, 27-48.

Gupta, U., & Chaudhury, S. (2015, December). Deep transfer learning with ontology for image classification. In *Computer Vision, Pattern Recognition, Image Processing and Graphics (NCVPRIPG), 2015 Fifth National Conference on* (pp. 1-4). IEEE.

Wang, B., Zhang, X., & Li, N. (2006, August). Relevance feedback technique for content-based image retrieval using neural network learning. In *Machine Learning and Cybernetics, 2006 International Conference on* (pp. 3692-3696). IEEE.