Assignment 2
CS635 2024

Prepared by Sneha Oram (23M2159)
Aashay Sandansing (23D1594)

October 13, 2024

Problem Statement

To implement Locality Sensitive Hashing (LSH) with image datasets. As a part of training, the images have to be kept in buckets in L hash tables to assign similar images to the same buckets in all hashtables. The program of LSH has to be implemented with 3 datasets with 3 different methodologies. The three image datasets are CIFAR-10 (10 classes), CIFAR-100 (100 classes), and ImageNet (1000 classes). The 3 methodologies to be adopted are - K-means clustering, random hyperplanes-based LSH, and Neural LSH.

Method 1: K-means Clustering

Data pre-processing

- The data (CIFAR-10, CIFAR-100) are downloaded from the torchvision inbuilt dataset and passed into the data loader for faster computation in further processing.
- As a part of preprocessing the images are transformed that is resized (224, 224), and normalized with mean [0.485,0.456,0.406] and standard deviation [0.229, 0.224, 0.225].
- The third dataset, we have considered taking from huggingface for easier computation and inferencing, rather than taking the actual Imagenet dataset (150 GB).
- The ResNet-18 model was used to extract 512x1 image embeddings across all datasets.

Training Design

- K-means clustering was applied to fit the embeddings of the training set.
- For the ImageNet-1K dataset, Mini Batch K-means clustering was used due to its large size.
- The same experiments were also conducted after applying Principal Component Analysis (PCA) to the embeddings, to observe the effect of dimensionality reduction.
- The variance is maintained at 95% in the original embedding dimension and in the lower dimension after applying PCA.

Results and Analysis on 3 datasets

- We have conducted a series of experiments with changing the number of Clusters (k) and reducing to a lower dimension with PCA (L).
- With the increasing number of clusters it can be observed that the precision@10 and precisio@50 are decreasing. However, the mean average precision is showing an opposite trend that is increasing the number of clusters, the MAP is increasing.
- The observed trend can be attributed to the fact that with the increase in the number of clusters, the instances in each of the clusters are reducing which is decreasing the precision@10, and precision@50, but increasing the MAP ratio.
- On applying PCA on the embedding dimension to reduce it from 512 to 248 the performance remains comparable. However, on reducing the dimension too much to 10, 20 the performance is degrading.
- The lower performance in PCA with dimensions 10, and 20 is due to the loss of information on changing dimensions from 512 to 10 or 20.
- With the ImageNet dataset, the observations are a bit different from what is observed in the other 2 datasets. The increase in the number of clusters is helping in the increase of precision@10, and precision@50. This can be explained by the very high number of classes (1000 labels) which is performing positively with more clusters.
- The time taken to retrieve the candidate images is approximately 2-4 seconds, the code takes care of the quick retrieval process.

K-means														
	CIFAR-10					CIFAR-100					ImageNet			
Embedding: 512	K = 10	K = 20	K = 100	K = 200	K = 1000	K = 100	K = 200	K = 500	K = 1000	K = 2000	K = 1000	K = 2000	K = 5000	K = 10000
Mean Precision @10	**75.58**	75.12	74.74	74.97	74.39	43	43.81	**44.17**	43.36	43.38	36.37	36.63	37.18	37.42
Mean Precision @50	**71.16**	70.78	70.7	**71.34**	69.21	35.84	37.01	**37.15**	35.9	**37.29**	30.96	31.69	32.77	33.06
Mean Average Precision	57.34	60.94	67.82	70.72	**74.96**	35.14	39.52	44.81	47.24	**50.38**	26.31	29.66	33.9	35.38
PCA: 248	K = 10	K = 20	K = 100	K = 200	K = 1000	K = 100	K = 200	K = 500	K = 1000	K = 2000	K = 1000	K = 2000	K = 5000	K = 10000
Mean Precision @10	**75.83**	75.01	74.83	75.53	75	43.7	**44.08**	43.59	43.75	43.33	36.22	36.62	37.15	**37.37**
Mean Precision @50	**71.69**	70.78	71	**71.85**	69.73	36.57	37.46	36.24	36.34	**37.19**	30.89	31.85	32.78	**33.01**
Mean Average Precision	58.62	60.83	67.96	71.03	**75.43**	35.72	39.85	44	47.65	**50.65**	26.62	29.76	34.21	**35.56**

Figure 1: Performance of 3 datasets on employing K-means clustering

Method 2: Random hyperplanes based LSH

Data pre-processing

- The data (CIFAR-10, CIFAR-100) are downloaded from the torchvision inbuilt dataset and passed into the data loader for faster computation in further processing.
- As a part of preprocessing the images are transformed that is resized (224,224) and normalized with mean [0.485,0.456,0.406] and standard deviation [0.229, 0.224, 0.225].

- The third dataset we have considered taking from huggingface dataset (25 GB) for easier computation and inferencing, rather than taking the actual Imagenet dataset (150 GB).
- The transformation is done to keep it in standard form before feeding it to the ResNet-18 model for extracting the image embedding of dimension 512.

Training Design

- The dot product between the randomly generated hyperplanes and the image embeddings is computed.
- The sign of the final dot product vector is taken to get the k-bit hash code of the images.
- The dot product of each image in the train set (gallery set) is computed with the hyperplanes, and the k-bit hash code is obtained for all the images.
- The next task is to create a dictionary with key values as the k-bit hash code, so there are 2^k keys or buckets and their values are the images with that k-bit hash code. Here the indices of the images are stored in the dictionary values rather than the actual images.
- This process is repeated L times, to create L hash tables each with 2^k keys or buckets.
- Next, for inferencing, each image from the test set is taken, its dot product with the same hyperplanes (generated during creating hashtables with the gallery set) is computed, and its k-bit hash code is generated.
- After generating a k-bit hashcode for the query image, this hashcode is searched in the dictionary keys. On getting the matched key or the hashcode the values (that is the incides) are extracted. This is repeated for all L hash tables and a union of all retrieved indices is collected and saved.
- The next task is to compute the cosine similarity between the query embedding (test image embedding) and the embedding of the images from the retrieved indices and then arrange them in decreasing order. This would be required to compute the precision@10, precisiom@50, and mean average precision.
- This process is done for all the images in the test set and the average overall precision@10, precsion@50, and mean average precision is computed.

Results and Analysis on 3 datasets

- We have conducted a series of experiments with changing the number of hyperplanes (k) and hashtables (L).
- With the increasing number of hyperplanes it can be observed that the precision@10 and precisio@50 are decreasing. However, the mean average precision shows an opposite trend that is increasing the number of hyperplanes, the MAP is increasing.
- The observed trend can be attributed to the fact that with the increase in the number of hyperplanes, the instances in each of the buckets are reducing which is decreasing the precsion@10, precision@50, but increasing the MAP ratio.

- With the ImageNet dataset, the number of hyperplanes is selected to be 10, 15, 30, and 100. This is to accommodate the higher number of classes with more bits. In this dataset, all three evaluation metrics are showing an increasing trend with the increase in the J-bits.
- The time taken to retrieve the candidate images is approximately 2-4 seconds, the code takes care of the quick retrieval process.

Random Hyperplanes Locality Sensitive Hashing														
	CIFAR-10					CIFAR-100					ImageNet			
#Hashtables = 3	**H = 3**	**H = 4**	**H = 7**	**H = 8**	**H = 10**	**H = 7**	**H = 8**	**H = 9**	**H = 10**	**H = 11**	**H = 10**	**H = 15**	**H = 30**	**H = 100**
Mean Precision @10	76.25	**76.15**	74.64	73.84	72	**43.86**	42.32	40.89	39.53	37.94	41.97	39.21	40.36	**42.33**
Mean Precision @50	**71.08**	70.82	68.73	67.42	64.56	**33.65**	31.66	29.74	27.75	25.73	35.23	37.96	39.63	**35.66**
Mean Average Precision	39.85	40.4	42.93	43.99	**45.66**	20.06	21.13	22.22	23.3	**24.1**	23.33	23.17	23.99	**25.61**
#Hashtables = 6	**H = 3**	**H = 4**	**H = 7**	**H = 8**	**H = 10**	**H = 7**	**H = 8**	**H = 9**	**H = 10**	**H = 11**	**H = 10**	**H = 15**	**H = 30**	**H = 100**
Mean Precision @10	**76.61**	76.51	75.88	75.77	74.68	**45.61**	45.25	44.28	43.54	42.37	44.23	44.01	45.31	**41.98**
Mean Precision @50	**71.5**	71.39	70.39	69.98	68.44	**35.95**	35.34	34.12	32.87	31.22	34.98	35.36	31.36	**33.36**
Mean Average Precision	39.25	39.83	42.46	43.15	**44.89**	19.24	19.85	20.81	21.73	**22.52**	24.96	24.36	22.33	**25.79**
#Hashtables = 12	**H = 3**	**H = 4**	**H = 7**	**H = 8**	**H = 10**	**H = 7**	**H = 8**	**H = 9**	**H = 10**	**H = 11**	**H = 10**	**H = 15**	**H = 30**	**H = 100**
Mean Precision @10	**76.64**	76.63	76.5	76.48	76.24	**46.7**	46.64	46.43	46.03	45.58	47.3	49.36	46.49	**50.01**
Mean Precision @50	**71.55**	71.54	71.34	71.26	70.81	**37.42**	37.37	37.04	36.44	35.75	36.78	37.04	39.33	**40.36**
Mean Average Precision	39.02	39.16	41	41.13	**42.74**	18	18.25	18.55	19.5	**20.26**	19.47	20.79	21.79	**22**

Figure 2: Performance of 3 datasets on employing random hyperplane-based LSH

Method 3: Neural LSH

Data pre-processing

- The data (CIFAR-10, CIFAR-100) are downloaded from the torchvision inbuilt dataset and passed into the data loader for faster computation in further processing.
- As a part of preprocessing the images are transformed that are resized (224,224) and normalized with mean [0.485,0.456,0.406] and standard deviation [0.229, 0.224, 0.225].
- The transformation is done to keep it in standard form before feeding it to the ResNet-50 model for extracting the image embedding of dimension 2048.
- The third dataset we have considered taking from huggingface dataset (25 GB) for easier computation and inferencing, rather than taking the actual Imagenet dataset (150 GB).

Training Design

- The linear layer is trained with epoch = 20 and learning rate - 0.001 to bring the embedding dimension from 2048 to a lower H dimension. So the linear layer is a fully connected layer with input dimension 2048 and output dimension H, with activation as tanh. During inferencing the activation is changed to a sign function, to get the J-bit hash code.
- The experiment is repeated L times.
- After training, J bits are selected randomly out of H output dimensions to create hash buckets of the image embeddings.

- The neural network is trained to reduce a combination of losses $\Delta_1, \Delta_2, \Delta_3$. The Δ_1 is the bit balance loss that checks that the buckets are balanced as the expectation is to have $\frac{N}{2^H}$ images. This is ensured by penalizing the loss function when the sum of the J-bits is not equal to 0. The Δ_2 is the no sitting on the fence loss where the optimizer is penalized for seating at 0. The Δ_3 is weak supervision that penalizes if the query label and retrieved image labels are not the same.
- The 3 losses are combined with 3 hyperparameters α, β, and γ with $\alpha + \beta + \gamma = 1$.
- The first two loss functions are easily implemented, but for the third loss of weak supervision, negative samples are extracted from the dataset. This is done by computing outer product of the embedding vectors resulting is matrix. This matrix is multiplied with a mask to get only the upper triangular values which gives the negative sample values.
- The output dimension H of the neural network is kept as 800 for all three datasets.
- The number of Hashtables considered for all experiments in neural LSH is 8.
- The coefficient of each of the losses α, β, and γ are kept constant at 0.3, 0.3, and 0.4 respectively throughout the training and inferencing.

$$\begin{aligned}\min_{\psi} \tfrac{\alpha}{|V|} \textstyle\sum_{u \in V} &\left|\mathbf{1}^\top \tanh(C_\psi(\mathbf{x}_u))\right| \\ &+ \tfrac{\beta}{|V|} \textstyle\sum_{u \in V} \Big\| \left|\tanh(C_\psi(\mathbf{x}_u))\right| - \mathbf{1} \Big\|_1 \\ + \tfrac{\gamma}{|\overline{E}|} \textstyle\sum_{(u,v) \in \overline{E}} &\left|\tanh(C_\psi(\mathbf{x}_u)) \cdot \tanh(C_\psi(\mathbf{x}_v))\right|\end{aligned}$$

Figure 3: Loss function for the compression network

Results and Analysis on 3 datasets

- We have conducted a series of experiments with changing the number of J-bits, randomly selecting from the $H = 800$ dimensions.
- With the increasing number of J-bits (randomly selected from H) it can be observed that the precision@10 and precisio@50 are decreasing. However, the mean average precision shows an opposite trend that is increasing the number of J-bits, and the MAP is increasing.
- The observed trend can be attributed to the fact that with the increase in the number of J-bits, the instances in each of the buckets are reducing which is decreasing the precsion@10, precision@50, but increasing the MAP ratio.
- With the ImageNet dataset, the J is selected to be 20, 80, and 100. This is to accommodate the higher number of classes with more bits of J. In this dataset, all three evaluation metrics are showing an increasing trend with the increase in the J-bits.
- The time taken to retrieve the candidate images is approximately 2-4 seconds, the code takes care of the quick retrieval process.

Neural Locality Sensitive Hashing											
	CIFAR-10				CIFAR-100				ImageNet-1K		
#HashTables = 8	J = 4	J = 8	J = 16	J = 32	J = 4	J = 8	J = 16	J = 32	J = 20	J = 80	J = 100
Mean Precision @10	79.65	**80.264**	74.49	78.32	**49.67**	47.12	46.96	38.24	**48.22**	48.12	49.36
Mean Precision @50	78.19	**78.78**	73.03	71.79	**40.95**	40.12	35.78	23.73	**40.96**	41.36	42.01
Mean Average Precision	42.96	48.63	52.09	**53.12**	19.79	25.39	27.74	**33.61**	18	26.33	29.3

Figure 4: Performance of 3 datasets on employing neural LSH

Overall Observation

After experimenting with 3 datasets with different methodologies, it can be concluded that the performance of the Neural LSH is better compared to Random hyperplane LSH which in turn is better compared to the K-means clustering. The better performance of random LSH compared to the K-means clustering can be attributed to the data being uniformly distributed across all classes. There are an equal number of instances for each label.