Assignment 5: Nasrul Huda, Giulia Saresini

Task 1: Clustering for image retrieval (pen & paper)

In 2017, Flickr launched a feature called “Similarity Search” which allows users to find similar images given a query image. A demo and technical description can be found at: http://code.flickr.net/2017/03/07/introducing-similarity-search-at-flickr/.
Based on this article, briefly answer the following questions about how clustering algorithms for large-scale nearest neighbor image search can be implemented in practice.

1. What is the “feature vector” for clustering in this application, and how is it obtained ?

   The “feature vector” is a high-dimensional vector extracted from an intermediate layer of a deep neural network (DNN) trained for tasks like object recognition or scene classification. In the Extraction process, the image is fed into the DNN, which processes it through multiple layers. Instead of using the final output layer (which produces the class probabilities), activations from an intermediate layer are extracted. These activations form the feature vector. For example, a DNN might output a 256-dimensional vector from a penultimate layer, capturing semantic content (e.g., objects, scenes) as well as visual attributes. (e.g., texture, composition). Images with similar content (e.g., “hot air balloons”) will have feature vectors close in Euclidean distance. The vector retains visual information discarded in the final classification layer, enabling nuances similarity comparisons.

2. Consider the direct k-nearest neighbours method to return k images most similar to a query image: extract the feature vector of the query image, and compare it to the feature vectors in the database to find the k feature vectors corresponding to the most similar images. Why is a direct implementation of this method not computationally feasible at a large scale (in the order of billions, 109, of images to search among)? Mention at least two reasons.

   For a database of 10^9 images, direct k-nearest neighbors search is infeasible due to various factors such as high time complexity and storage problems. If we go into detail about these two reasons:

   Time Complexity:
   Comparing the query vector to all 10^9 vectors requires O(n) operations per query. Even with optimizations, this is too slow for real-time applications. For example, a 256-dimensional vector requires 256 operations per distance calculation. For 10^9 images, this totals 256 x 10^9 operations per query.

   Storage Problems:
   Storing full-precision (e.g., 32-bit float) feature vectors for billions of images consumes massive memory. For example, 10^9 images with 256-D vectors require 10^9 x 256 x 4 bytes = 1 TB of memory. Loading this into RAM for fast access is impractical.

3. The idea of product quantization is to split every large feature vector $\vec{x}$ into subvectors $\vec{x}i$, i = 1,...,n, so that $\vec{x} = \vec{x}1 \vec{x}2 ... \vec{x}n$ and then use k-means clustering on the subvectors separately. Then given a query vector $\vec{q}$, its nearest cluster center can be found by clustering each subvector $\vec{q}$ independently. Why does this help speed up nearest neighbor search.

   Product Quantization addresses these challenges through dimensionality splitting and compression. In the key steps of Product Quantization:

   1. A 256-D vector is divided into n subvectors (e.g., n = 8 each 32-D).
   2. Each subvector is clustered independently using k-means (e.g., k = 256 clusters per subspace).
   3. This creates a codebook of centroids for each subspace.
   4. Each subvector is replaced with the index of its nearest centroid (e.g., an 8-byte code for 8 subspaces).
   5. Storage drops from 256 x 4 bytes = 1024 bytes to 8 bytes per vector.
   6. During a query, distances are computed using precomputed lookup tables for subvector-centroid distances.
   7. Example: For a query split into 8 subvectors, compute distances to all 256 centroids in each subspace. Store results in a table.
   8. The total distance for a database vector is approximated by summing the precomputed subvector distances.

   Advantages:

   1. The computations are reduced: Instead of O(d) operations (for d-dimensional vectors), PQ uses O(nk) pre computations and O(n) table lookups.
   2. Memory efficiency : Storing 8-byte codes instead of full vectors reduces memory usage by 99.2%.
   3. Inverted Multi-Index: Splitting vectors into halves and clustering each half (e.g., 1000 clusters per half) creates 10^6 virtual clusters with only 2000 distance computations, enabling efficient candidate retrieval.

   LOPQ Enhancement:

   1. After coarse clustering, residuals (vector minus centroid) are rotated using PCA to align with axes, improving quantization accuracy.
   2. This compensates for statistical dependencies between subspaces, reducing approximation errors.

Task 2: Cross entropy and label smoothing regularization (pen & paper)

In this task, we review the motivation for and interpretation of label smoothing regularization, a technique used in training neural networks for large scale image classification. Consider an image

classification problem with m mutually exclusive classes. The typical loss function used to train classification networks is the cross-entropy loss, defined as:

$$\ell(\vec{q}, \vec{p}) = -\sum_{i=1}^{m} q_i \log p_i,$$

Where $\vec{p} = \begin{bmatrix} p_1 & p_2 & \cdots & p_m \end{bmatrix}^T$ is a vector of class probabilities output by the neural network after a softmax activation function, and $\vec{q} = \begin{bmatrix} q_1 & q_2 & \cdots & q_m \end{bmatrix}^T$ is a vector of probabilities expressing the desired or ground truth output.

1. Each p_i is formed from the activation values $\vec{z} = \begin{bmatrix} z_1 & z_2 & \cdots & z_m \end{bmatrix}^T$, also called the logits, from the output layer before the softmax activation function:

$$p_i = \frac{\exp(z_i)}{\sum_{j=1}^{m} \exp(z_j)}.$$

   Show that $\frac{\partial \ell(\vec{q},\vec{p})}{\partial z_i} = p_i - q_i$. What are the minimum and maximum values of this partial derivative ? Hint : recall that p_i and q_i are probabilities.

   Given the cross-entropy loss:

$$\ell(\vec{q}, \vec{p}) = -\sum_{i=1}^{m} q_i \log p_i,$$

$$p_i = \frac{\exp(z_i)}{\sum_{j=1}^{m} \exp(z_j)}.$$

   where

   Rewrite $\log p_i$ using softmax:

$$\log p_i = z_i - \log\left(\sum_{j=1}^{m} \exp(z_j)\right).$$

   Substitute into the loss:

$$\ell = -\sum_{i=1}^{m} q_i \left(z_i - \log\left(\sum_{j=1}^{m} \exp(z_j)\right)\right)$$

   Simplify using $\sum_{i=1}^{m} q_i = 1$:

   Compute the partial derivative $\frac{\partial \ell}{\partial z_i}$:

$$\frac{\partial \ell}{\partial z_i} = -q_i + \frac{\exp(z_i)}{\sum_{j=1}^{m} \exp(z_j)} = p_i - q_i.$$

Minimum and Maximum values:

Since $p_i, q_i \in [0, 1]$, the derivative $\frac{\partial \ell}{\partial z_i}$: ranges from :

Minimum: -1
Maximum: 1

2. An image with true class h is typically assigned a vector $\vec{q}$ such that

$$q_i = \begin{cases} 1 & \text{iff } i = h \\ 0 & \text{otherwise} \end{cases},$$

i.e., exactly the element corresponding to the true class equals 1 while others equal 0. Show that in such a case, $\ell(\vec{q}, \vec{p}) = -\log p_h$. That is, when we minimize cross-entropy, we are (equivalently) maximizing the log likelihood $\log p_h$ of the true class.

Cross Entropy Loss for One-hot Labels

For a one-hot vector $\vec{q}$ where $q_h = 1$ and $q_i = 0$ for $i \neq h$:

$$\ell(\vec{q}, \vec{p}) = -\sum_{i=1}^{m} q_i \log p_i = -\log p_h.$$

Interpretation: Minimizing cross-entropy is equivalent to maximizing $\log p_h$, the log probability of the true class.

3. The theoretical maximum of $\log p_h$ is log 1 = 0 (Why?). This maximum is approached when the logit z_h of the ground truth class is much larger than all other logits $z_i,\ i \neq h.$. Szegedy et al. argue that this causes two problems:

    1. Over-fitting. If the model learns to assign full probability to the ground-truth label for each training example, it is not guaranteed to generalize.
    2. Encourages the differences between the largest logit and all other logits to become large. Combined with the bounded gradient ∂ℓ(q⃗,p)
    ∂zi , this reduces the ability of the
    model to adapt to new data. Intuitively, the model becomes too confident about its

predictions.

To address these problems, they propose label smoothing regularization (LSR), where instead of the one-hot ground truth vector $\vec{q}$, a new vector $\vec{q'} = [q'_1 \quad q'_2 \quad \cdots \quad q'_m]^T$ is used that is a weighted mixture of $\vec{q}$ and a uniform distribution $\vec{u} = [\frac{1}{m} \quad \frac{1}{m} \quad \cdots \quad \frac{1}{m}]^T$.

. Specifically, $\vec{q'} = (1-\epsilon)\vec{q} + \epsilon\vec{u}$, where $\epsilon \in (0,1)$ is a weight term typically chosen small, e.g., 0.1. When $\ell(\vec{q'}, \vec{p})$ is used as the loss function, the network is no longer encouraged to make the largest logit much larger than all others. It was empirically shown that LSR consistently improved top-1 and top-5 errors on ImageNet classification by about 0.2%. Show that:

$$\ell(\vec{q'}, \vec{p}) = (1-\epsilon)\ell(\vec{q}, \vec{p}) + \epsilon\ell(\vec{u}, \vec{p}).$$

This allows an alternative interpretation. The second term $\ell(\vec{u},\vec{p})$ penalizes the network for deviations from the prior u, while the first term encourages the network to match the one-hot vector $\vec{q}$.

Label Smoothing Regularization (LSR)

Given $\vec{q'} = (1-\epsilon)\vec{q} + \epsilon\vec{u}$, where $\vec{u}$ is uniform $(u_i = \frac{1}{m})$:

Substitute $\vec{q'}$ into the loss:

$$\ell(\vec{q'}, \vec{p}) = -\sum_{i=1}^{m} [(1-\epsilon)q_i + \epsilon u_i] \log p_i.$$

Split into two terms:

$$\ell(\vec{q'}, \vec{p}) = -(1-\epsilon)\sum_{i=1}^{m} q_i \log p_i - \epsilon \sum_{i=1}^{m} u_i \log p_i.$$

Recognize the cross-entropy terms:

$$\ell(\vec{q'}, \vec{p}) = (1-\epsilon)\ell(\vec{q}, \vec{p}) + \epsilon\ell(\vec{u}, \vec{p}).$$

Task 3: Convolutional layers (pen & paper)
See note at start of assignment sheet concerning this task !

Recall from the lecture that it is typical to think of the inputs and outputs of convolutional layers as 3-dimensional volumes. It is in fact so common, that this fact is implicitly assumed in many sources and must be used by the reader to interpret the sizes of the layer inputs/outputs, and their number of

parameters, if required. Spatial dimensions are often given, but the third dimension must be inferred from other information.

The table below shows a segment of two convolutional layers in a CNN, showing the typical information available to a reader of a design document. Based on the table and the interpretation in the lecture of inputs/outputs as 3D volumes, answer the questions below. You can ignore batch size. Justify all your answers.

Layer index	Number of filters	Filter size	Bias?	Activation	Spatial dimension of output
$l-1$	32	3 x 3	yes	ReLU	112 x 112
l	64	3 x 3	yes	ReLU	112 x 112

1. Given the information in the table only, can you infer the full size of the 3D volume that is the input to layer l - 1? If yes, what is it? If not, why not? Note: layer l - 1 is not necessarily the first layer in the network.

   The input to layer l - 1 is a 3D volume with dimensions: Height x Width x Depth
   From the table:
   1. The spatial dimensions of the output of layer l - 1 are 112 x 112
   2. The number of filters in layer l - 1 is 32, which corresponds to the depth of its output.

   However, the input depth to layer l - 1 is not explicitly given, Since layer l - 1 is not necessarily the first layer, we cannot infer the input depth from the table alone.

   Answer: The full size of the input to layer l - 1 cannot be determined from the table because the input depth is unknown.

2. Can you infer the size of the 3D volume that is the input to layer l (equivalently, the output of layer l - 1)? If yes, what is it? If not, why not?

   The input to layer l is the output of layer l - 1. From the table:
   1. The spatial dimensions of the output of layer l - 1 are 112 x 112.
   2. The number of filters in layer l - 1 is 32, which corresponds to the depth of its output.

   Thus, the input to layer l is a 3D volume of size: 112 x 112 x 32
   Answer: The input to layer l has dimensions 112 x 112 x 32

3. Does layer l apply any kind of padding on its input? Why can you say this is or is not the case?

   The spatial dimensions of the output of layer l are 112 x 112, the same as its input, This implies that padding is applied in layer l.

   Reason: Without padding, a 3 x 3 convolution would reduce the spatial dimensions by 2 (e.g., from 112 x 112 to 110 x 110). Since the output size remains 112 x 112, padding must be used to preserve the spatial dimensions.

Answer: Layer l applies padding to maintain the spatial dimensions of its input.

4. How would the spatial dimensions of the output layer l change if bias is removed? Why?

   Removing the bias term from layer l does not affect the spatial dimensions of the output. The bias term only adds a learnable offset to each output channel and does not influence the height or width of the output.

   Answer: The spatial dimensions of the output remain 112 x 112 even if bias is removed.

5. How many parameters does layer l have? Besides the final answer, also give an equation and explain where the values come from.

   The number of parameters in a convolutional layer is determined by:
   1. Weights: Number of filters x filter size x input depth.
   2. Biases: Number of filters (one bias per filter)

   From the table:
   1. Number of filters in layer l: 64
   2. Filter size: 3 x 3
   3. Input depth: 32 (from layer l - 1)
   4. Bias: Enabled.

   Calculation:
   Number of parameters = (Number of filters x Filter size x Input depth)

   Substitute values:
   Number of parameters = (64 x 3 x 3 x 32) + 64 = 18,496.

   Answer: Layer l has 18,496 parameters.