Quentin de La Chaise

Naël Briand

TP2 Image and Video Processing :

Image Classification

1.2 Bag-of-Words Classification Pipeline Overview

The Bag-of-words method converts raw image data into identifiable patterns. Through the extraction of HOG features and clustering with K-means, the system simplifies complex visual information into structured formats. All the image, initially a collection of pixels, is represented by a histogram that maps its connection to a predefined set of visual words. Then we assign labels to unfamiliar images by matching them with recognized categories thanks to a nearest neighbor classifier and the use of these histograms.

Question : So, without employing any approximation techniques the primary component contributing to space complexity is the storage of BoW histograms for the training images. Each training image is represented as a BoW histogram, which consists of K words corresponding to the visual words obtained from clustering.

Let T be the number of training images and K the number of visual words. The space required to store the BoW histograms for all training images can be expressed as:

$$O(K*T)$$

So, the space complexity is proportional to the number of training images and the number of visual words.

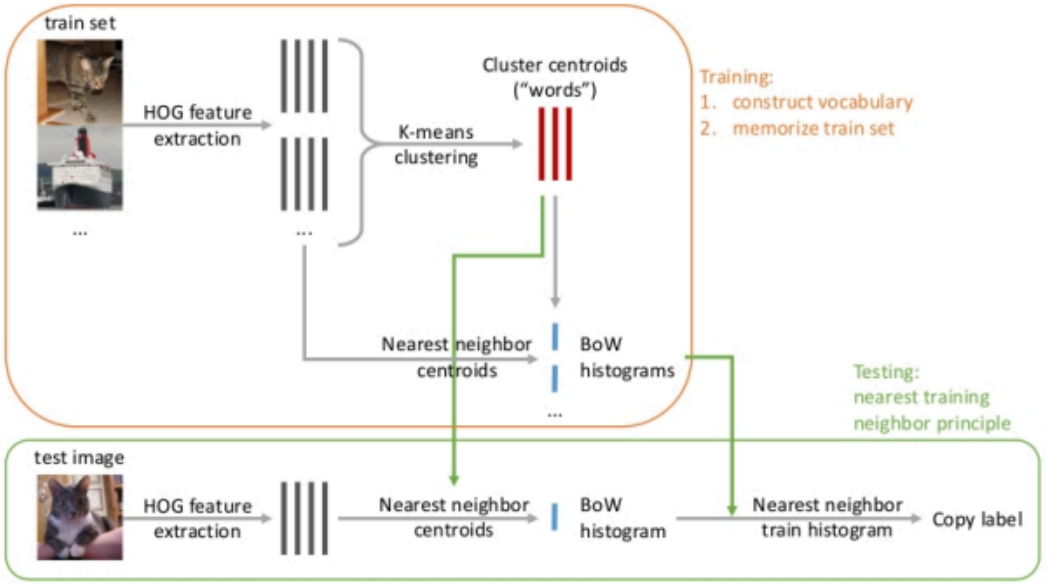
Afterwards time complexity depends on two main steps: calculating distances between the test image and all training images, and finding the nearest neighbor.

So first for each test image, we compute the distance between its BoW histogram and the histograms of all T training images. Assuming that calculating the distance takes O(K) time for each histogram comparison, the total time for calculating distances to all training images becomes:

$$O(K*T)$$

After computing the distances, we need to identify the minimum distance to determine the nearest neighbor. This requires scanning the T distance values, which takes O(T) time. However, this operation is usually considered part of the distance computation since it operates in linear time relative to the number of training images.

So we have finally: O(K*T)



1.3 Feature Description with Histograms of Oriented Gradients (HOG)

HOG captures edge direction patterns in images by computing gradient magnitude and orientation then structure tensors provide information about the direction and consistency of image gradients within a region.

- Step 1: Compute Gradients

Gradients in the x and y directions for an image I(x, y) are obtained using Sobel filters:

Gx = d I(x, y)/ d x and Gy = d I(x, y)/d y

- Step 2: Magnitude M(x, y)

M(x, y) = sqrt(Gx(x, y)^2 + Gy(x, y)^2)

- Step 3: Orientation theta(x, y)

theta(x, y) = tan^(-1)[Gy(x, y) / Gx(x, y)]

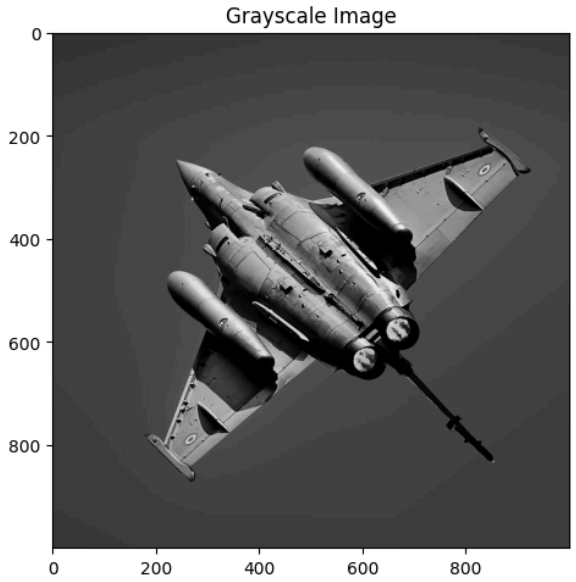
- Step 4: Histogram of Gradients

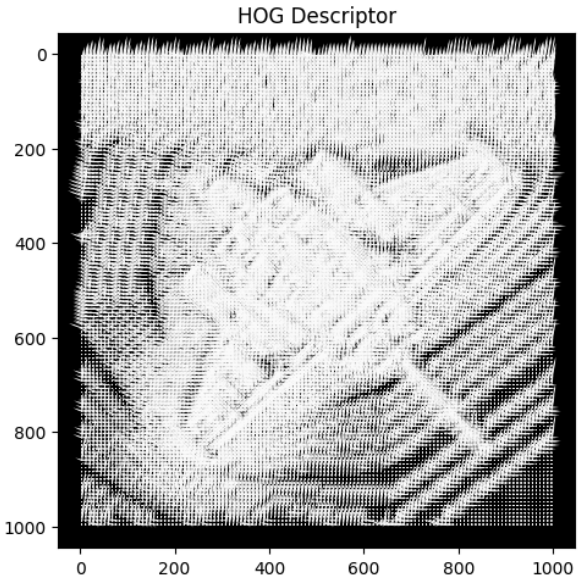
With the gradient orientation theta(x, y) and the magnitude M(x, y) grouped into bins we create an histogram for each cell which is combined and normalized with all others.This results in a method that aims to capture edge direction distributions.

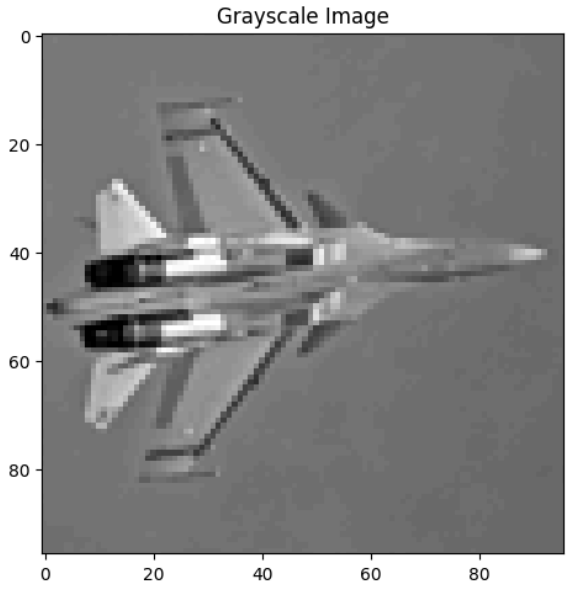
In the HOG descriptor we have Cell Size, Block Size, Block Stride and Num Bins and we choose them in order to have finer parameters like smaller cell sizes, larger block sizes, and more bins in order to capture more detail. But here we have to stay reasonable in this accuracy because the objective is just to classify so no need of high level detail.

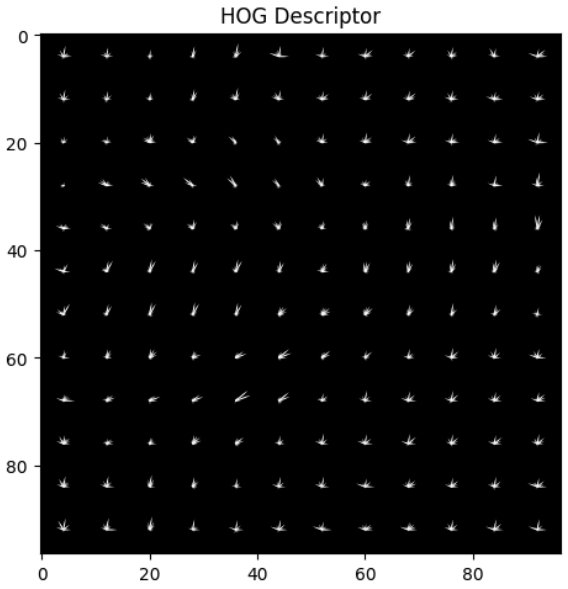
Pseudo Code for HOG Descriptor Computation

Function: compute_hog_descriptor(image)
Step 1: Compute Sobel gradients
  - (Gx, Gy)← Sobel gradients in X and Y directions
Magnitude and Orientation:
  - magnitude, orientation as we saw previously
Step 2:Define HOG parameters
  - cell size ← (8, 8)
  - block size ← (16, 16)
  - block stride ← (8, 8)
  - num bins ← 9
Step 3: Compute and reshape HOG descriptor.
Step 4: Calculate average gradients for each cell/block.
Step 5: Visualize gradients using a quiver plot.
Return: U, V, X, Y (gradient vectors for visualization).

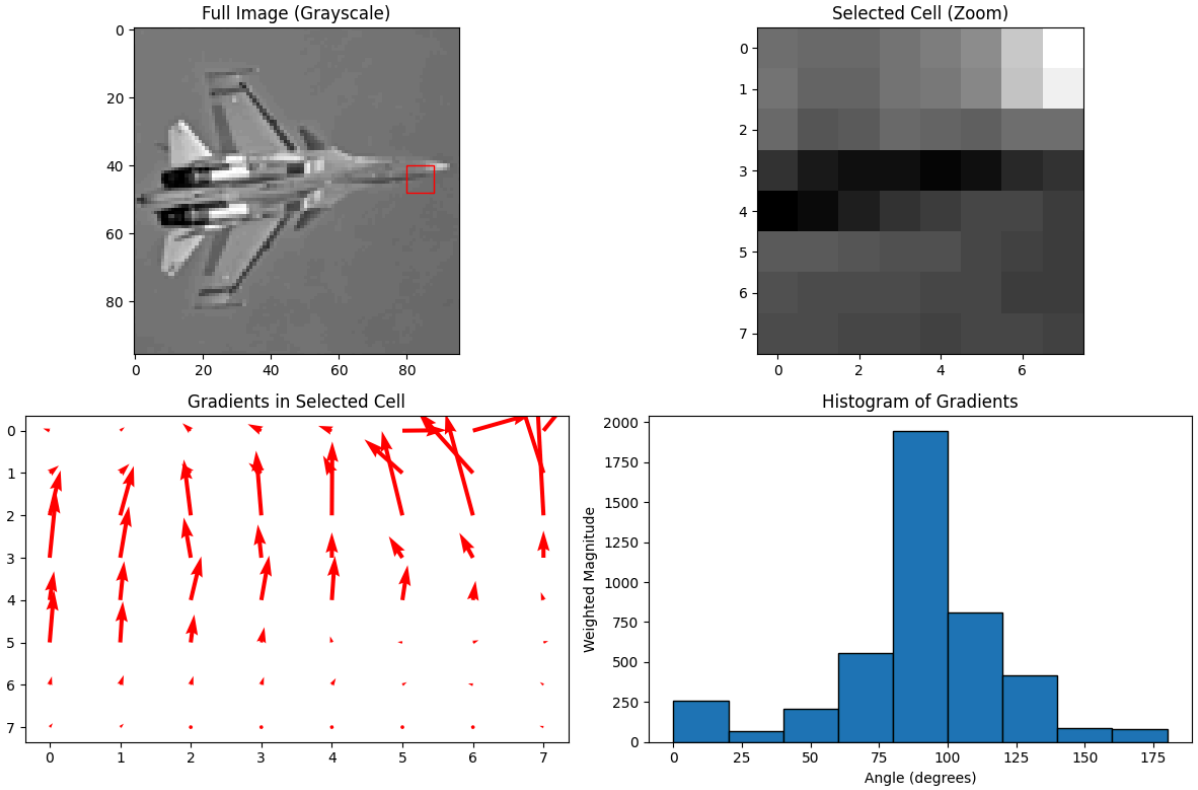








Analyze of the results



This Figure illustrates the HOG representation of an image.

- Grayscale Image: This grayscale image depicts an aircraft

- HOG Descriptor: This displays HOG features, with white strokes representing the strength and orientation of gradients (edges or contrast changes) across the image.

- Gradients : This shows gradient vectors (arrows) within a single cell, indicating the direction and magnitude of intensity changes (edges).

- Histogram of the Gradients :This shows a clear distribution of edge directions within the cell

Now we will use this HOG tool to classify image.
Let's start with the HOG representation of each Image

The process involves organizing images into folders corresponding to specific categories:

- Labeling with folder: Each image is stored in a parent folder named after its associated category like airplane or bird or other.

- Extract Descriptors: Each image undergoes processing to calculate HOG descriptors, which are feature vectors capturing local shape and texture based on gradient distributions.

- Parallel Processing: The process utilizes parallel processing to efficiently handle large datasets, aiming to speed up computations by processing multiple images simultaneously.

Pseudo code for HOG Descriptor for Image Processing in Folders

Function: compute_hog_for_image(image_path, parent_folder)

Step 1: Read the image in grayscale mode.
  - image ← read_image(image_path, grayscale=True)
Step 2: Compute the HOG descriptor.
  - hog_descriptor, num_blocks_x, num_blocks_y ← compute_hog_descriptor(image)
Step 3: Return the HOG descriptor and parent folder.
  - return hog_descriptor, parent_folder

Function: process_images_in_folder(folder_path, parent_folder)

Step 1: Initialize an empty results list.
Step 2: For each image name in the folder:
  - Construct the image path.
  - If the image is a valid file (png, jpg, jpeg):
    - Compute the HOG data for the image.
    - If HOG data is valid, append it to results.
Step 3: Return results.

Function: process_parent_directory(parent_dir, n_jobs=-1)

Step 1: List folders in the parent directory.
Step 2: Initialize an empty results list.
Step 3: For each folder:
  - Construct the folder path.
  - Process images in the folder and store results.

Step 4: Return all results

1.4 Visual Vocabulary Construction with K-means Clustering

Visual vocabulary is a method used for image classification which represents images as collections of "visual words”

In this method we still have to:

1. K-means Clustering: Clustering these features to form a dictionary of "visual words."
2. Image Representation: Each image is represented by a histogram of the frequency of these visual words, capturing the distribution of patterns and textures.

This method abstracts raw image data into a compact representation, useful for image retrieval, classification, and object recognition.

K-means clustering, partitions data into K clusters by assigning each data point to the nearest cluster centroid, minimizing variance within clusters.

In image processing, K-means groups HOG descriptors extracted from images into clusters, with each cluster representing specific image patterns. This approach aids in building a Bag-of-Words (BoW) model for image classification.

The K-means algorithm follows the steps below :

Pseudo code: K-means Clustering with Ground Truth

I : Initialize random cluster centroids.
Function: initialize_centroids(data, K)

Step 1: Count Samples: Determine the number of data samples available.
Step 2: Select Indices: Randomly choose indices from the dataset.
Step 3: Assign Centroids: Select data points corresponding to the chosen indices to serve as initial centroids.
Step 4: Return: Output the initialized centroids.

II: : Assign each data point to the cluster whose centroid is nearest to the point with respect to the Euclidean distance.

Function: assign_clusters(data, centroids)

Step 1: Calculate Distances: Compute the Euclidean distances between data points and the centroids.
Step 2: Assign Labels: Label each data point with the index of its nearest centroid.
Step 3: Return: Output the assigned labels for the data points.

III - Update each cluster centroid to the mean of all data points that are currently assigned to the cluster.
Function: update_centroids(data, labels, K)

Step 1: Recalculate Centroids: For each cluster, compute the mean of the data points assigned to that cluster.
Step 2: Return: Output the updated centroids based on the new assignments.

Function: kmeans(data, K, max_iters, tol)

Step 1: Initialize Centroids: Use the `initialize_centroids` function to generate initial centroid values.
Step 2: Iterate: Repeat the following for a maximum of `max_iters`:
- Assign clusters by calling `assign_clusters` with current centroids.
- Update the centroids using `update_centroids` based on the assigned labels.
- If the change in centroids is less than the specified tolerance `tol`, terminate the loop.

Step 3: Return: Provide the final centroids and the corresponding labels for each data point.

Function:kmeans_with_ground_truth(data, results, K)

Step 1: Run K-means: Call the `kmeans` function to compute centroids and assign labels to the data.
Step 2: Pair Labels: Combine each assigned label with its corresponding original data type.
Step 3: Return: Output the clusters alongside their original data types.

Function: apply_mapping_to_clusters(cluster_with_original_type)`
Step 1: Map Clusters: Create a mapping of cluster indices to their respective parent folder names.
Step 2. Update List: Pair each assigned cluster with its corresponding original type for easier interpretation.
Step 3. Return: Provide the updated list containing these mappings.

Function: calculate_clustering_accuracy(updated_cluster_with_original_type)

Step 1: Extract Data: Retrieve the assigned clusters and original types from the updated list.
Step 2: Identify Unique Values: Determine the unique assigned clusters and original types present.
Step 3: Build Confusion Matrix: Construct a confusion matrix to compare the original and assigned clusters.
Step 4: Calculate Accuracy: Use linear sum assignment on the confusion matrix to compute the clustering accuracy.
Step 5: Return: Provide the calculated accuracy value.

The methodology implemented in this study involved some modifications to standard practices. The number of K-means clusters \(K \) was set to 100, and the algorithm was executed for 10 iterations—common parameters for such experiments. To enhance the reliability of the results, I evaluated the model performance across 10 separate runs, starting with two classes.

Variations were introduced during K-means clustering, particularly in the convergence behavior and the averaging of results from multiple runs. This approach helps mitigate the randomness inherent in centroid initialization, providing a more accurate representation of overall accuracy and general model performance across different trials.

Running K-means multiple times is essential due to its reliance on random centroid initialization, which can produce varying outcomes. This repeated execution helps average out such randomness, ensuring that the reported accuracy reflects the model's true performance rather than being skewed by an outlier result.

1.7 Experiment
- k = 10, L1 Norm, Accuracy = 0.26 The low accuracy indicates poor separation among the 10 clusters, resulting in significant overlap.

- k = 10, L2 Norm, Accuracy = 0.25 Similar to the L1 norm, the L2 norm also shows low accuracy, suggesting that increasing the number of clusters does not enhance classification performance.

- k = 2, L1 Norm, Accuracy = 0.88 A significant improvement in accuracy indicates better separation into two clusters, suggesting the dataset favors fewer clusters for effective discrimination.

- k = 2, L2 Norm, Accuracy = 0.89 The accuracy is slightly higher, reinforcing that in lower dimensions, the choice of norm has minimal impact on performance.

The distribution of image classifications into clusters based on different k-values and norms (L1 and L2). The results reflect how various image classes are categorized using K-means clustering.

Finite Dimensional Hypothesis

In finite-dimensional vector spaces, L1 and L2 norms are often interchangeable without drastically affecting results. While L2 is preferred in clustering for its emphasis on larger deviations, the similar outcomes for both norms in this case arise from the bounded dimensionality of HOG descriptors. The small number of clusters (k = 2) reduces the significance of the norm choice.

Conversely, higher k values lead to decreased accuracy due to increased cluster overlap, as evident in the poor results for both k = 10 scenarios.