implicitShapeModelTrainingTask

May 6, 2025

1 Exercise 2: Implicit Shape Model

Task: Develop a first prototype of your own ***Implicit Shape Model*** object detection system.

1.0.1 Preparation

1. Take several images (at least 10) for an object that is of interest to you in front of a homogeneous background or in front of a blue screen.
2. Take as many images of a typical background of you object type.
3. Take as many images of an object in front of typical background.

1.0.2 Software development

1. Code a jupyter notebook (you may take this as a starting point) that trains an implicit shape model for your object class. Take the lecture slides and Leibe, B., Leonardis, A., & Schiele, B. (2004, May). Combined object categorization and segmentation with an implicit shape model. In Workshop on statistical learning in computer vision, ECCV (Vol. 2, No. 5, p. 7) as a role model.
2. Code a jupyter notebook that uses the trained model to detect objects in front of background.

2 Feature Extraction with Non-Maximum Suppression

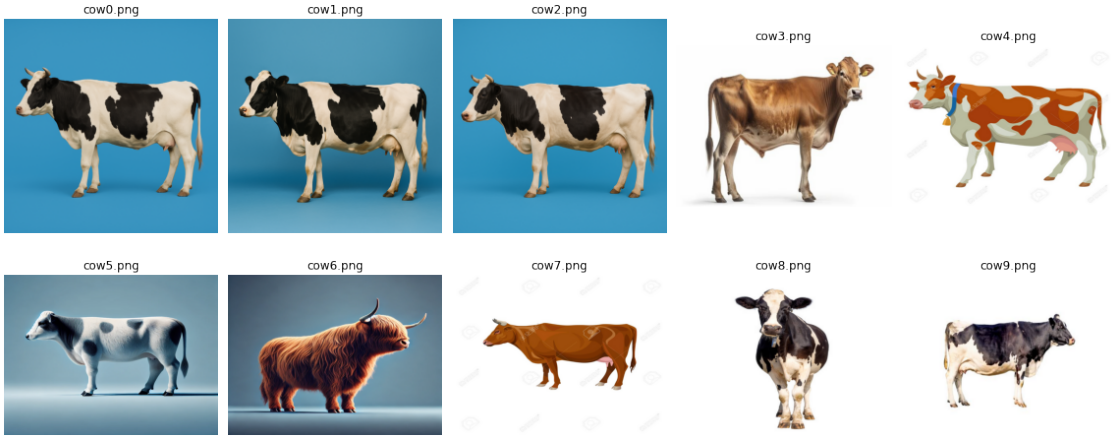
Goal: Extract features, apply non-maximum suppression (NMS) so that no low-response keypoint survives within the radius (`size / 2`) of a stronger keypoint, and visualise both the raw and pruned sets.

2.0.1 Pipeline

1. Load & display images
2. Extract keypoints/descriptors
3. Visualise **all** raw keypoints
4. Apply NMS
5. Visualise **retained** keypoints
6. Save the NMS-pruned descriptors and keypoint metadata

Dependencies: `opencv-python`, `numpy`, `matplotlib`.

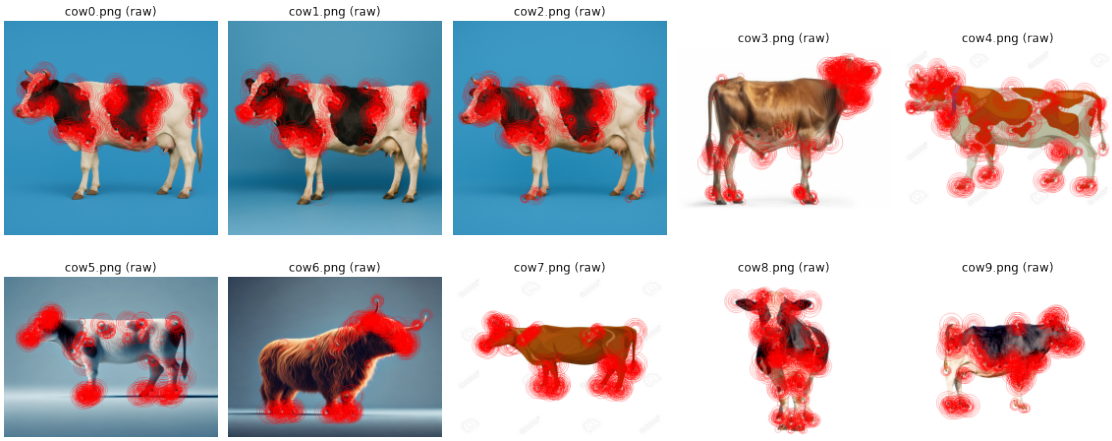
[1]:
```
# 1. Load & display images
```



[2]:
```
# 2. Extract keypoints/descriptors
```

```
Keypoints per image (raw): [1000, 1000, 1000, 1000, 1000, 1000, 1000, 1000,
1000, 1000]
```

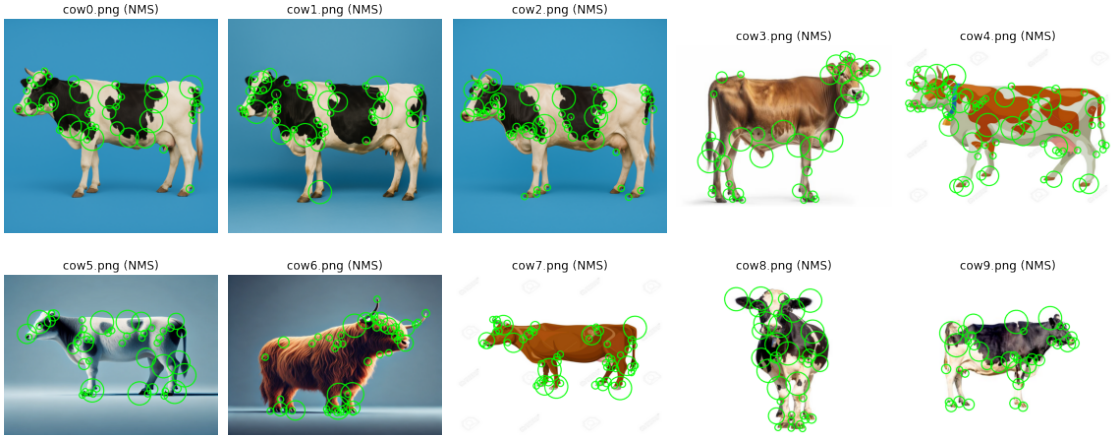
[3]:
```
# 3. Visualise all raw keypoints
```



[4]:
```
# 4. Apply NMS
```

```
Keypoints per image (after NMS): [46, 57, 70, 37, 75, 50, 53, 56, 35, 37]
```

[5]:
```
# 5. Visualise retained keypoints
```



[2]:
```
# 6. Save the NMS-pruned descriptors and keypoint metadata
```

3 Feature-Space Clustering and Spatial Analysis

Assumes the variables created in the previous code cells are present: * `images` – list of RGB images * `image_files` – corresponding filenames * `keypoints_nms` – list of non-maximum-suppressed keypoint lists per image * `descriptors_nms` – matching list of NumPy descriptor matrices (`uint8`, $32 \times N$)

Run this notebook in the *same* kernel session **after** executing the extraction notebook, or paste these cells below the earlier ones.

3.0.1 Pipeline

7. Cluster keypoint descriptors (enhanced descriptors)
8. Compute center of gravity of keypoints and displacement vector to center of gravity in every "object" (here: cow) training image
9. Save per-cluster & displacement data ("visual words"/"codebook entries" of the objects
10. Generate a check plot for each training image

[7]:
```
# 7. Cluster keypoint descriptors (enhanced descriptors)
```

```
Clustering 516 descriptors into K=16 clusters …
Cluster assignment done.
```

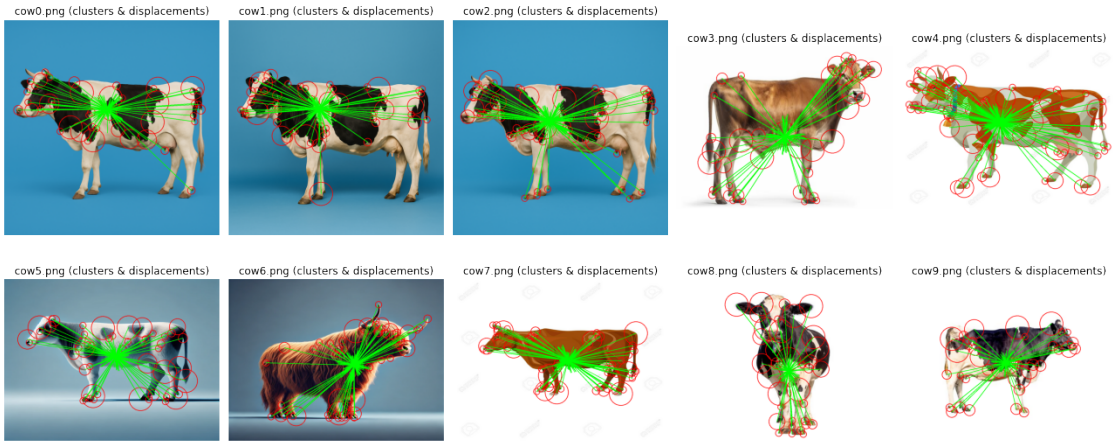
[8]:
```
# 8. Compute center of gravity of keypoints and displacement vector
#    to center of gravity in every "object" (here: cow) training image
```

```
Computed centres of gravity and displacement vectors.
```

[9]:
```
# 9. Save per-cluster & displacement data ("visual words"/"codebook entries" of␣
↪the objects
```

```
Saved per-feature cluster & displacement data to /media/hellwich/Herakles/hellwi
ch/hellwich/lehre/Aut_BA/2025Exercises/Exercise2/orb_features_cluster
```

[3]:
```
# 10. Check plot features and displacement vectors for each training image
```



4 Let the Features Vote with all their Displacement Vectors:

5 Cluster-wise Displacement Field Visualisation

Essentially, here we apply the detection pipeline to the training images. In general it has to be **applied to the test images**, i.e. in the detection step. We nevertheless do it in order to conduct a plausibility test.

Prerequisite: `images`, `image_files`, `keypoints_nms`, `displacements_per_image`, `cluster_labels_per_image`.

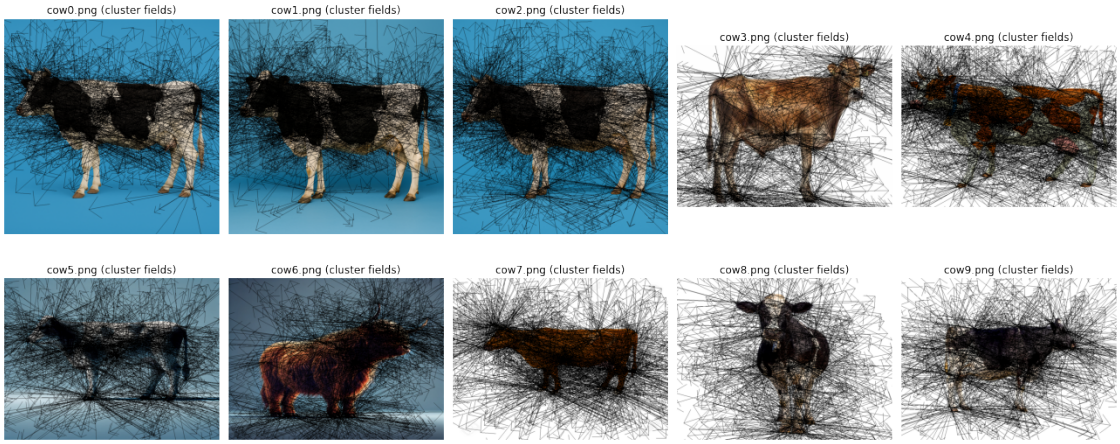
5.0.1 Pipeline:

11. Build a lookup table that is mapping **all** displacement vectors of a k-means cluster to a keypoint whose feature vector let's it belong to that cluster.
12. Draw, for every keypoint in every image, **the entire displacement set of its cluster** — giving a dense field that reveals spatial patterns the visual word is linked to.

[11]:
```
# 11. Build a lookup table that is mapping **all** displacement vectors of a␣
↪k-means cluster to a keypoint whose feature vector let's it belong to that␣
↪cluster.
```

```
Prepared displacement lists for 16 clusters.
```

[12]:
```
# 12. Draw, for every keypoint in every image, **the entire displacement set of␣
↪its cluster** - giving a dense field that reveals spatial patterns the␣
↪visual word is linked to.
```



6 Save *all* enriched feature data to file

Pack every datum you now have — i.e. keypoint metadata, descriptors, cluster labels, displacement vectors, **and** the global displacement lists per cluster — into self-contained, human-readable files.

For instance:

- For each image write a compressed **`.npz`**:
  - `x`, `y`, `size`, `angle`, `response`, `cluster`, `dx`, `dy`, `descriptors` (Your features may have different data.)
- A single **`cluster_displacements.json`** maps each k-means label to the list of `[dx, dy]` vectors observed in that cluster.

[13]:
```
# 13. Save all enriched feature data to file
```

```
Wrote:
 • cluster_centers.npy   -  (K,32) centroid matrix
 • cluster_details.json  -  centroids + full member/displacement map
Saved enriched per-image feature archives and global cluster displacement list
to /media/hellwich/Herakles/hellwich/hellwich/lehre/Aut_BA/2025Exercises/Exercis
e2/orb_allfeatures
```

[14]:
```
# 14. Make sure you are also able to read the data of the files back into memory
```

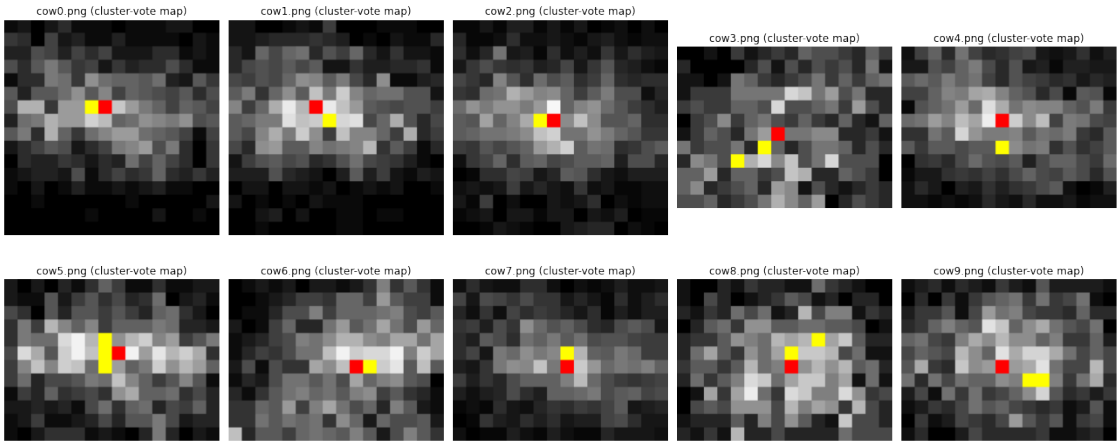
7 Voting Procedure

Implement a voting procedure.

For instance:

15. Put a low resolution grid of voting bins across your image, cast the keypoint's votes to the bins, count the votes and display them. It may be adequate to indicate the grid cell with maximum votes in a special way.

[ ]:
```
# 15. Put a low resolution grid of voting bins across your image, cast the␣
↪keypoint's votes to the bins, count the votes and display them.
```



[ ]:
```

```