**EX.NO:9 DATE: 13.03.2025**

**OBJECT DETECTION**

**Aim:**

To perform image classification (cats vs. dogs) using HOG features and SVM, and to detect objects in images using SIFT features, feature matching, homography, and K-Means clustering.  
**Algorithm:**

**1. Image Classification Using HOG and SVM**

**Algorithm**:

1. Load & Resize Images: Grayscale images loaded from folders and resized.
2. Extract HOG Features: HOG features extracted using skimage.feature.hog.
3. Dataset Split: Data split into training and testing sets.
4. SVM Training: Linear SVM classifier trained.
5. Prediction & Evaluation: Model predicts on test data; accuracy and report generated.

**2. Object Detection Using SIFT and Homography (First Instance)**

**Algorithm**:

1. Prepare Images: Training and test images loaded & converted to grayscale.
2. Detect Features: SIFT detects keypoints & computes descriptors.
3. Match Keypoints: BFMatcher finds matches; Lowe's ratio test filters.
4. Find Homography: RANSAC estimates homography matrix.
5. Detect Object: Bounding box drawn using homography.
6. Display Results: Image shown with bounding box.

**3. Object Detection Using SIFT and Homography (Second Instance)**

**Algorithm**:

1. Initialization: SIFT detector and BFMatcher initialized.
2. Feature Matching: KNN matching with Lowe's ratio test.
3. Homography Estimation: RANSAC estimates homography (if enough matches).
4. Object Localization: Object corners transformed using homography.
5. Visualization: Bounding box drawn; feature matches displayed.

**4. Object Detection Using SIFT, Feature Matching, and K-Means Clustering**

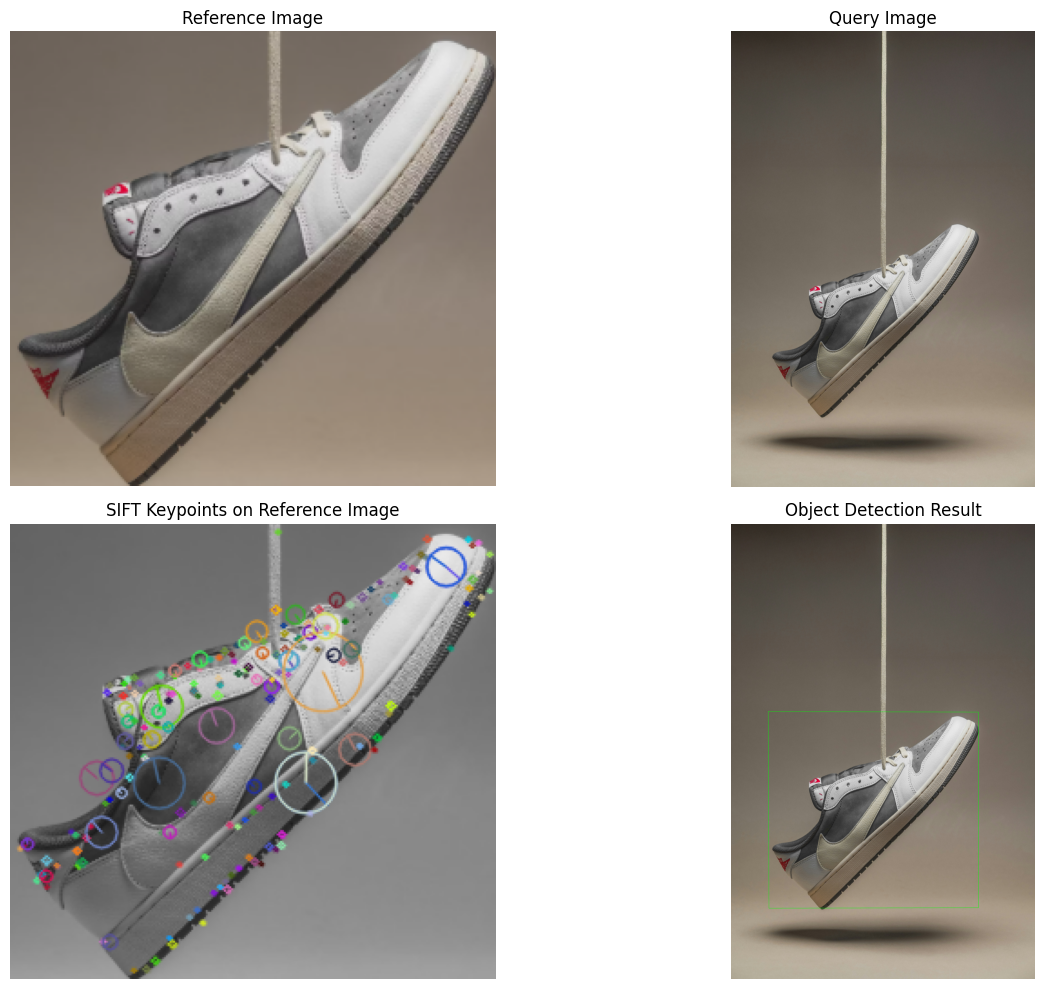
**Algorithm**:

1. Feature Matching: Matches found via BFMatcher and Lowe's ratio.
2. Keypoint Extraction: Matched keypoint coordinates extracted.
3. K-Means Clustering: Keypoints clustered into *k* groups.
4. Bounding Box Creation: Bounding boxes defined around each cluster.
5. Visualization: Training image shown with bounding boxes.

**Code:**

import cv2  
import numpy as np  
import matplotlib.pyplot as plt  
  
def detect\_and\_match\_objects(reference\_image\_path, query\_image\_path, min\_match\_count=10):  
 Detect objects in a query image based on a reference image using SIFT features.  
  
 Parameters:  
 reference\_image\_path (str): Path to the reference image (object to be detected)  
 query\_image\_path (str): Path to the query image (where to find the object)  
 min\_match\_count (int): Minimum number of good matches required  
  
 Returns:  
 tuple: (result\_image, homography\_matrix, matches\_mask)  
 reference\_img = cv2.imread(reference\_image\_path)  
 query\_img = cv2.imread(query\_image\_path)  
  
 reference\_gray = cv2.cvtColor(reference\_img, cv2.COLOR\_BGR2GRAY)  
 query\_gray = cv2.cvtColor(query\_img, cv2.COLOR\_BGR2GRAY)  
  
 sift = cv2.SIFT\_create()  
  
 kp\_reference, des\_reference = sift.detectAndCompute(reference\_gray, None)  
 kp\_query, des\_query = sift.detectAndCompute(query\_gray, None)  
  
 FLANN\_INDEX\_KDTREE = 1  
 index\_params = dict(algorithm=FLANN\_INDEX\_KDTREE, trees=5)  
 search\_params = dict(checks=50)  
  
 flann = cv2.FlannBasedMatcher(index\_params, search\_params)  
  
 matches = flann.knnMatch(des\_reference, des\_query, k=2)  
  
 good\_matches = []  
 for m, n in matches:  
 if m.distance < 0.7 \* n.distance:  
 good\_matches.append(m)  
  
 matches\_mask = [[0, 0] for \_ in range(len(matches))]  
  
 homography = None  
 if len(good\_matches) >= min\_match\_count:  
 src\_pts = np.float32([kp\_reference[m.queryIdx].pt for m in good\_matches]).reshape(-1, 1, 2)  
 dst\_pts = np.float32([kp\_query[m.trainIdx].pt for m in good\_matches]).reshape(-1, 1, 2)  
  
 homography, mask = cv2.findHomography(src\_pts, dst\_pts, cv2.RANSAC, 5.0)  
  
 for i, (m, \_) in enumerate(matches):  
 if m in good\_matches and mask[good\_matches.index(m)]:  
 matches\_mask[i] = [1, 0]  
  
 h, w = reference\_gray.shape  
 corners = np.float32([[0, 0], [0, h-1], [w-1, h-1], [w-1, 0]]).reshape(-1, 1, 2)  
 transformed\_corners = cv2.perspectiveTransform(corners, homography)  
  
 result\_img = query\_img.copy()  
 cv2.polylines(result\_img, [np.int32(transformed\_corners)], True, (0, 255, 0), 3)  
  
 print(f"Object found - {len(good\_matches)} good matches")  
 else:  
 print(f"Not enough matches found - {len(good\_matches)}/{min\_match\_count}")  
 result\_img = query\_img.copy()  
 homography = None  
  
 return result\_img, homography, matches\_mask  
  
def visualize\_keypoints(image\_path):  
 Visualize SIFT keypoints on an image  
  
 Parameters:  
 image\_path (str): Path to the image  
  
 Returns:  
 numpy.ndarray: Image with keypoints drawn  
 img = cv2.imread(image\_path)  
 gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)  
  
 sift = cv2.SIFT\_create()  
 keypoints = sift.detect(gray, None)  
  
 img\_with\_keypoints = cv2.drawKeypoints(gray, keypoints, None, flags=cv2.DRAW\_MATCHES\_FLAGS\_DRAW\_RICH\_KEYPOINTS)  
  
 return img\_with\_keypoints  
  
def visualize\_matches(reference\_image\_path, query\_image\_path):  
 Visualize matches between reference and query images  
  
 Parameters:  
 reference\_image\_path (str): Path to the reference image  
 query\_image\_path (str): Path to the query image  
  
 Returns:  
 numpy.ndarray: Image showing matches  
 reference\_img = cv2.imread(reference\_image\_path)  
 query\_img = cv2.imread(query\_image\_path)  
  
 reference\_gray = cv2.cvtColor(reference\_img, cv2.COLOR\_BGR2GRAY)  
 query\_gray = cv2.cvtColor(query\_img, cv2.COLOR\_BGR2GRAY)  
  
 sift = cv2.SIFT\_create()  
  
 kp\_reference, des\_reference = sift.detectAndCompute(reference\_gray, None)  
 kp\_query, des\_query = sift.detectAndCompute(query\_gray, None)  
  
 FLANN\_INDEX\_KDTREE = 1  
 index\_params = dict(algorithm=FLANN\_INDEX\_KDTREE, trees=5)  
 search\_params = dict(checks=50)  
  
 flann = cv2.FlannBasedMatcher(index\_params, search\_params)  
  
 matches = flann.knnMatch(des\_reference, des\_query, k=2)  
  
 good\_matches = []  
 for m, n in matches:  
 if m.distance < 0.7 \* n.distance:  
 good\_matches.append(m)  
  
 img\_matches = cv2.drawMatches(reference\_img, kp\_reference, query\_img, kp\_query, good\_matches, None,  
 flags=cv2.DrawMatchesFlags\_NOT\_DRAW\_SINGLE\_POINTS)  
  
 return img\_matches  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 reference\_image = "ref.png"  
 query\_image = "shoe.jpg"  
  
 result, homography, matches\_mask = detect\_and\_match\_objects(reference\_image, query\_image)  
  
 result\_rgb = cv2.cvtColor(result, cv2.COLOR\_BGR2RGB)  
  
 reference\_keypoints = visualize\_keypoints(reference\_image)  
 reference\_keypoints\_rgb = cv2.cvtColor(reference\_keypoints, cv2.COLOR\_BGR2RGB)  
  
 matches\_visualization = visualize\_matches(reference\_image, query\_image)  
 matches\_visualization\_rgb = cv2.cvtColor(matches\_visualization, cv2.COLOR\_BGR2RGB)  
  
 plt.figure(figsize=(15, 10))  
  
 plt.subplot(2, 2, 1)  
 plt.imshow(cv2.cvtColor(cv2.imread(reference\_image), cv2.COLOR\_BGR2RGB))  
 plt.title('Reference Image')  
 plt.axis('off')  
  
 plt.subplot(2, 2, 2)  
 plt.imshow(cv2.cvtColor(cv2.imread(query\_image), cv2.COLOR\_BGR2RGB))  
 plt.title('Query Image')  
 plt.axis('off')  
  
 plt.subplot(2, 2, 3)  
 plt.imshow(reference\_keypoints\_rgb)  
 plt.title('SIFT Keypoints on Reference Image')  
 plt.axis('off')  
  
 plt.subplot(2, 2, 4)  
 plt.imshow(result\_rgb)  
 plt.title('Object Detection Result')  
 plt.axis('off')  
  
 plt.tight\_layout()  
 plt.show()  
  
 plt.figure(figsize=(15, 10))  
 plt.imshow(matches\_visualization\_rgb)  
 plt.title('Feature Matches')  
 plt.axis('off')  
 plt.show()

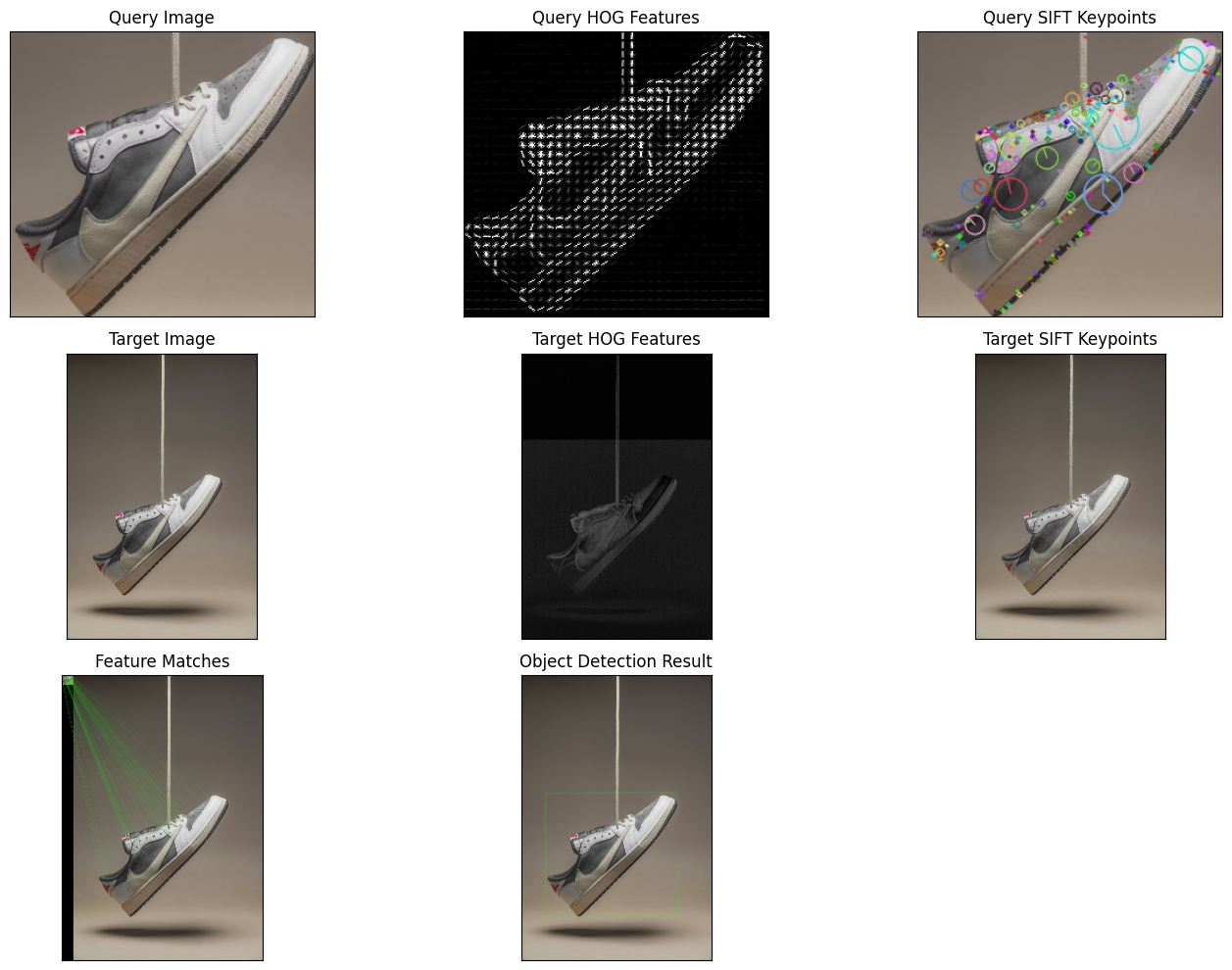
Object found - 91 good matches





import cv2  
import numpy as np  
import matplotlib.pyplot as plt  
from skimage.feature import hog  
from skimage import exposure  
from scipy import ndimage  
  
def compute\_sift\_features(image):  
 Compute SIFT features for the given image  
 if len(image.shape) == 3:  
 gray = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)  
 else:  
 gray = image  
  
 sift = cv2.SIFT\_create()  
  
 keypoints, descriptors = sift.detectAndCompute(gray, None)  
  
 return keypoints, descriptors  
  
def compute\_hog\_features(image, orientations=9, pixels\_per\_cell=(8, 8), cells\_per\_block=(2, 2)):  
 Compute HOG features for the given image  
 if len(image.shape) == 3:  
 gray = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)  
 else:  
 gray = image  
  
 hog\_features, hog\_image = hog(  
 gray,  
 orientations=orientations,  
 pixels\_per\_cell=pixels\_per\_cell,  
 cells\_per\_block=cells\_per\_block,  
 visualize=True,  
 block\_norm='L2-Hys'  
 )  
  
 hog\_image\_rescaled = exposure.rescale\_intensity(hog\_image, in\_range=(0, 10))  
  
 return hog\_features, hog\_image\_rescaled  
  
def compute\_gloh\_features(image, keypoints):  
 Compute GLOH features for the given image and keypoints  
 GLOH is an extension of SIFT that uses a log-polar location grid with 3 bins in radial  
 direction and 8 in angular direction, giving 17 location bins (including the central bin)  
 if len(image.shape) == 3:  
 gray = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)  
 else:  
 gray = image  
  
 gloh\_descriptors = []  
  
 num\_radial\_bins = 3  
 num\_angular\_bins = 8  
 num\_orientation\_bins = 8  
 radius\_max = 20   
  
 for kp in keypoints:  
 x, y = int(kp.pt[0]), int(kp.pt[1])  
 scale = kp.size  
 orientation = kp.angle  
  
 if x < radius\_max or y < radius\_max or x >= gray.shape[1] - radius\_max or y >= gray.shape[0] - radius\_max:  
 continue  
  
 patch = gray[y-radius\_max:y+radius\_max, x-radius\_max:x+radius\_max]  
  
 dx = ndimage.sobel(patch, axis=1)  
 dy = ndimage.sobel(patch, axis=0)  
  
 magnitude = np.sqrt(dx\*\*2 + dy\*\*2)  
 orientation\_map = np.arctan2(dy, dx) \* 180 / np.pi % 360  
  
 y\_grid, x\_grid = np.ogrid[-radius\_max:radius\_max, -radius\_max:radius\_max]  
 r\_grid = np.sqrt(x\_grid\*\*2 + y\_grid\*\*2)  
 theta\_grid = (np.arctan2(y\_grid, x\_grid) \* 180 / np.pi) % 360  
  
 radial\_bins = np.zeros(num\_radial\_bins + 1)  
 radial\_bins[0] = 0  
 radial\_bins[1] = radius\_max / 3  
 radial\_bins[2] = radius\_max \* 2 / 3  
 radial\_bins[3] = radius\_max  
  
 descriptor = np.zeros((1 + num\_radial\_bins \* num\_angular\_bins) \* num\_orientation\_bins)  
  
 central\_mask = r\_grid < radial\_bins[1]  
 for o in range(num\_orientation\_bins):  
 orient\_min = o \* 360 / num\_orientation\_bins  
 orient\_max = (o + 1) \* 360 / num\_orientation\_bins  
 orient\_mask = (orientation\_map >= orient\_min) & (orientation\_map < orient\_max)  
 mask = central\_mask & orient\_mask  
 descriptor[o] = np.sum(magnitude[mask])  
  
 descriptor\_idx = num\_orientation\_bins  
 for r in range(1, num\_radial\_bins):  
 radial\_mask = (r\_grid >= radial\_bins[r]) & (r\_grid < radial\_bins[r+1])  
 for a in range(num\_angular\_bins):  
 angle\_min = a \* 360 / num\_angular\_bins  
 angle\_max = (a + 1) \* 360 / num\_angular\_bins  
 angle\_mask = (theta\_grid >= angle\_min) & (theta\_grid < angle\_max)  
 for o in range(num\_orientation\_bins):  
 orient\_min = o \* 360 / num\_orientation\_bins  
 orient\_max = (o + 1) \* 360 / num\_orientation\_bins  
 orient\_mask = (orientation\_map >= orient\_min) & (orientation\_map < orient\_max)  
 mask = radial\_mask & angle\_mask & orient\_mask  
 descriptor[descriptor\_idx] = np.sum(magnitude[mask])  
 descriptor\_idx += 1  
  
 norm = np.linalg.norm(descriptor)  
 if norm > 0:  
 descriptor /= norm  
  
 gloh\_descriptors.append(descriptor)  
  
 return np.array(gloh\_descriptors) if gloh\_descriptors else None  
  
def detect\_objects(query\_image, target\_image, min\_match\_count=10):  
 Detect objects in target image using features from query image  
 kp1, des1 = compute\_sift\_features(query\_image)  
  
 kp2, des2 = compute\_sift\_features(target\_image)  
  
 FLANN\_INDEX\_KDTREE = 1  
 index\_params = dict(algorithm=FLANN\_INDEX\_KDTREE, trees=5)  
 search\_params = dict(checks=50)  
 flann = cv2.FlannBasedMatcher(index\_params, search\_params)  
  
 matches = flann.knnMatch(des1, des2, k=2)  
  
 good\_matches = []  
 for m, n in matches:  
 if m.distance < 0.7 \* n.distance:  
 good\_matches.append(m)  
  
 if len(good\_matches) >= min\_match\_count:  
 src\_pts = np.float32([kp1[m.queryIdx].pt for m in good\_matches]).reshape(-1, 1, 2)  
 dst\_pts = np.float32([kp2[m.trainIdx].pt for m in good\_matches]).reshape(-1, 1, 2)  
  
 M, mask = cv2.findHomography(src\_pts, dst\_pts, cv2.RANSAC, 5.0)  
 matchesMask = mask.ravel().tolist()  
  
 h, w = query\_image.shape[:2]  
 pts = np.float32([[0, 0], [0, h-1], [w-1, h-1], [w-1, 0]]).reshape(-1, 1, 2)  
  
 dst = cv2.perspectiveTransform(pts, M)  
  
 result\_image = target\_image.copy()  
 cv2.polylines(result\_image, [np.int32(dst)], True, (0, 255, 0), 3, cv2.LINE\_AA)  
  
 draw\_params = dict(  
 matchColor=(0, 255, 0),

singlePointColor=None,  
 matchesMask=matchesMask, # Only draw matched keypoints  
 flags=2  
 )  
  
 match\_image = cv2.drawMatches(query\_image, kp1, target\_image, kp2, good\_matches, None, \*\*draw\_params)  
  
 return True, result\_image, match\_image  
 else:  
 print("Not enough good matches: {}/{}".format(len(good\_matches), min\_match\_count))  
 return False, target\_image, None  
  
def main():  
 query\_image = cv2.imread('ref.png')  
 target\_image = cv2.imread('shoe.jpg')  
  
 if query\_image is None or target\_image is None:  
 print("Error: Could not load one or both images")  
 return  
  
 success, result\_image, match\_image = detect\_objects(query\_image, target\_image)  
  
 \_, hog\_image\_query = compute\_hog\_features(query\_image)  
 \_, hog\_image\_target = compute\_hog\_features(target\_image)  
  
 keypoints\_query, \_ = compute\_sift\_features(query\_image)  
 keypoints\_target, \_ = compute\_sift\_features(target\_image)  
  
 gloh\_descriptors\_query = compute\_gloh\_features(query\_image, keypoints\_query)  
 gloh\_descriptors\_target = compute\_gloh\_features(target\_image, keypoints\_target)  
  
 sift\_image\_query = cv2.drawKeypoints(  
 query\_image,  
 keypoints\_query,  
 None,  
 flags=cv2.DRAW\_MATCHES\_FLAGS\_DRAW\_RICH\_KEYPOINTS  
 )  
  
 sift\_image\_target = cv2.drawKeypoints(  
 target\_image,  
 keypoints\_target,  
 None,  
 flags=cv2.DRAW\_MATCHES\_FLAGS\_DRAW\_RICH\_KEYPOINTS  
 )  
  
 plt.figure(figsize=(15, 10))  
  
 plt.subplot(331), plt.imshow(cv2.cvtColor(query\_image, cv2.COLOR\_BGR2RGB))  
 plt.title('Query Image'), plt.xticks([]), plt.yticks([])  
  
 plt.subplot(332), plt.imshow(hog\_image\_query, cmap='gray')  
 plt.title('Query HOG Features'), plt.xticks([]), plt.yticks([])  
  
 plt.subplot(333), plt.imshow(cv2.cvtColor(sift\_image\_query, cv2.COLOR\_BGR2RGB))  
 plt.title('Query SIFT Keypoints'), plt.xticks([]), plt.yticks([])  
  
 plt.subplot(334), plt.imshow(cv2.cvtColor(target\_image, cv2.COLOR\_BGR2RGB))  
 plt.title('Target Image'), plt.xticks([]), plt.yticks([])  
  
 plt.subplot(335), plt.imshow(hog\_image\_target, cmap='gray')  
 plt.title('Target HOG Features'), plt.xticks([]), plt.yticks([])  
  
 plt.subplot(336), plt.imshow(cv2.cvtColor(sift\_image\_target, cv2.COLOR\_BGR2RGB))  
 plt.title('Target SIFT Keypoints'), plt.xticks([]), plt.yticks([])  
  
 if success:  
 plt.subplot(337), plt.imshow(cv2.cvtColor(match\_image, cv2.COLOR\_BGR2RGB))  
 plt.title('Feature Matches'), plt.xticks([]), plt.yticks([])  
  
 plt.subplot(338), plt.imshow(cv2.cvtColor(result\_image, cv2.COLOR\_BGR2RGB))  
 plt.title('Object Detection Result'), plt.xticks([]), plt.yticks([])  
  
 plt.tight\_layout()  
 plt.show()  
  
 print(f"Number of SIFT keypoints in query image: {len(keypoints\_query)}")  
 print(f"Number of SIFT keypoints in target image: {len(keypoints\_target)}")  
  
 if gloh\_descriptors\_query is not None:  
 print(f"Number of GLOH descriptors in query image: {len(gloh\_descriptors\_query)}")  
  
 if gloh\_descriptors\_target is not None:  
 print(f"Number of GLOH descriptors in target image: {len(gloh\_descriptors\_target)}")  
  
 if success:  
 print("Object successfully detected in target image!")  
 else:  
 print("Object detection failed.")  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()



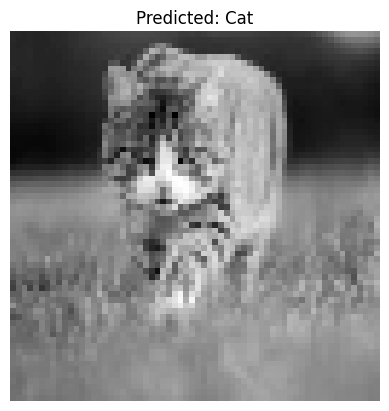
Number of SIFT keypoints in query image: 206  
Number of SIFT keypoints in target image: 53823  
Number of GLOH descriptors in query image: 180  
Number of GLOH descriptors in target image: 53816  
Object successfully detected in target image!

import numpy as np  
import cv2  
import os  
import glob  
import matplotlib.pyplot as plt  
from skimage.feature import hog  
from sklearn.model\_selection import train\_test\_split  
from sklearn.svm import SVC  
from sklearn.metrics import accuracy\_score, classification\_report  
  
def load\_images\_from\_folder(folder, label):  
 images = []  
 labels = []  
 for filename in glob.glob(os.path.join(folder, "\*.jpg")):   
 img = cv2.imread(filename, cv2.IMREAD\_GRAYSCALE)   
 if img is not None:  
 img = cv2.resize(img, (64, 64))   
 images.append(img)  
 labels.append(label)  
 return images, labels  
  
cat\_images, cat\_labels = load\_images\_from\_folder("drive/MyDrive/Datasets/catsAndDogs40/train/cat", label=0)  
dog\_images, dog\_labels = load\_images\_from\_folder("drive/MyDrive/Datasets/catsAndDogs40/train/dog", label=1)  
  
X = np.array(cat\_images + dog\_images)  
y = np.array(cat\_labels + dog\_labels)  
  
hog\_features = []  
for image in X:  
 feature, \_ = hog(image, pixels\_per\_cell=(8, 8), cells\_per\_block=(2, 2), visualize=True)  
 hog\_features.append(feature)  
  
X\_hog = np.array(hog\_features)  
  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_hog, y, test\_size=0.2, random\_state=42)  
  
svm\_model = SVC(kernel='linear')  
svm\_model.fit(X\_train, y\_train)  
  
y\_pred = svm\_model.predict(X\_test)  
  
accuracy = accuracy\_score(y\_test, y\_pred)  
print("Model Accuracy:", accuracy)  
print("\nClassification Report:\n", classification\_report(y\_test, y\_pred))

Model Accuracy: 0.6153846153846154  
  
Classification Report:  
 precision recall f1-score support  
  
 0 0.67 0.33 0.44 6  
 1 0.60 0.86 0.71 7  
  
 accuracy 0.62 13  
 macro avg 0.63 0.60 0.58 13  
weighted avg 0.63 0.62 0.59 13

def predict\_image(image\_path, model):  
 img = cv2.imread(image\_path, cv2.IMREAD\_GRAYSCALE) # Convert to grayscale  
 img = cv2.resize(img, (64, 64)) # Resize to match training data  
 feature = hog(img, pixels\_per\_cell=(8, 8), cells\_per\_block=(2, 2), visualize=False).reshape(1, -1)  
 prediction = model.predict(feature)  
 label = "Cat" if prediction[0] == 0 else "Dog"  
 print(f"Prediction: {label}")  
  
 plt.imshow(img, cmap='gray')  
 plt.title(f"Predicted: {label}")  
 plt.axis('off')  
 plt.show()  
  
predict\_image("drive/MyDrive/Datasets/catsAndDogs40/test/cat/4.jpg", svm\_model)

Prediction: Cat



**Inference:**

**1. Image Classification Using HOG and SVM**

* HOG features are effective in capturing shape and texture information relevant for image classification.
* A linear SVM can be a suitable classifier for this task, providing a linear decision boundary.
* The accuracy and classification report provide insights into the model's ability to generalize to unseen data.

**2. Object Detection Using SIFT and Homography (First Instance)**

* SIFT features are robust to scale and rotation changes, making them suitable for object detection in different views.
* The homography transformation can map the reference object to its location in the query image, enabling object localization.
* Successful object detection depends on the presence of sufficient good matches between the keypoints.

**3. Object Detection Using SIFT and Homography (Second Instance)**

* Applying Lowe's ratio test filters out ambiguous matches, improving the robustness of homography estimation.
* Checking for sufficient matches before computing homography prevents erroneous transformations.
* The bounding box drawn on the test image indicates the detected object's location and extent.

**4. Object Detection Using SIFT, Feature Matching, and K-Means Clustering**

* By clustering keypoints in the training image, the code attempts to group points belonging to the same object.
* K-Means clustering provides a simple way to segment the training image into different object instances.
* The bounding boxes represent the spatial extent of each detected object instance.

**Result:**

Outputs image classification accuracy and reports; displays images with detected objects, bounding boxes, and feature matches. May print messages indicating object detection success or failure.