Lab3.2 Report

Bag of Visual Words (BoVW) for Image Classification

Course: Computer Vision

Date: October 26, 2025

Student: ROUBACHE Islam

1. Objective

The purpose of this laboratory work is to implement the **Bag of Visual Words (BoVW)** model for image classification. The technique is conceptually similar to the **Bag of Words (BoW)** approach used in NLP, but here, image features extracted from local regions are treated as "visual words." The main goal is to transform images into numerical histograms of visual patterns and classify them using a machine learning model.

2. Dataset and Tools

- **Dataset:** UC Merced Land Use Dataset (21 categories, 2100 images total)
- **Programming Language:** Python
- **Libraries:** OpenCV, NumPy, Scikit-learn, Matplotlib

3. Implementation Code

3.1 Importing Required Libraries

```
import os
import cv2
import numpy as np
from sklearn.cluster import KMeans
from sklearn.svm import SVC
from sklearn.preprocessing import normalize
from sklearn.metrics import accuracy_score, confusion_matrix,
    ConfusionMatrixDisplay
import matplotlib.pyplot as plt
```

3.2 Data Loading and Preprocessing

```
data_dir = r"UCMerced_LandUse"
classes = sorted([d for d in os.listdir(data_dir) if os.path.isdir(os.path.join(
    data_dir, d))])
image_size = (256, 256)
num_classes = len(classes)

images, labels = [], []
for idx, class_name in enumerate(classes):
    class_dir = os.path.join(data_dir, class_name)
    for image_file in sorted(os.listdir(class_dir)):
        img_path = os.path.join(class_dir, image_file)
        img = cv2.imread(img_path)
        if img is not None:
            img = cv2.resize(img, image_size)
            images.append(img)
            labels.append(idx)
images, labels = np.array(images), np.array(labels)
```

3.3 Train-Test Split (by Class)

```
def train_test_split(X, y, test_size=0.2, random_state=None):
    if random_state is not None:
        np.random.seed(random_state)
    indices = np.arange(len(X))
    np.random.shuffle(indices)
    test_size = int(len(X) * test_size)
    train_size = len(X) - test_size
    X_train = X[indices[:train_size]]
    X_test = X[indices[train_size:]]
    y_train = y[indices[:train_size]]
    y_test = y[indices[train_size:]]
    return X_train, X_test, y_train, y_test

X_train, X_test, y_train, y_test = [], [], [], []
for class_idx in range(num_classes):
    class_images = images[labels == class_idx]
    class_labels = labels[labels == class_idx]
    X_train_cls, X_test_cls, y_train_cls, y_test_cls = train_test_split(
        class_images, class_labels, test_size=0.2, random_state=42)
    X_train.extend(X_train_cls); y_train.extend(y_train_cls)
    X_test.extend(X_test_cls); y_test.extend(y_test_cls)

X_train, y_train, X_test, y_test = map(np.array, [X_train, y_train, X_test,
    y_test])
```

3.4 Feature Extraction using SIFT

```
def visualize_keypoints(image, keypoints):
    img_with_keypoints = cv2.drawKeypoints(
        image, keypoints, None, flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS
            )
    return cv2.cvtColor(img_with_keypoints, cv2.COLOR_BGR2RGB)
```

3.5 Descriptor and Histogram Creation

```
def descriptors(X_train):
    desc_list = []
    for img in X_train:
        gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        sift = cv2.SIFT_create()
        kp, desc = sift.detectAndCompute(gray, None)
        if desc is not None:
            desc_list.append(desc)
    return np.vstack(desc_list)

def img_hist(x, kmeans, num_clusters):
    hists = []
    for img in x:
        gray = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
        kp, desc = cv2.SIFT_create().detectAndCompute(gray, None)
        if desc is not None:
            words = kmeans.predict(desc)
            hist, _ = np.histogram(words, bins=np.arange(num_clusters + 1))
            hists.append(hist)
        else:
            hists.append(np.zeros(num_clusters))
    return normalize(np.array(hists), norm='l2')
```

3.6 BoVW Model with Multiple Cluster Sizes

```
def bovw_experiment(X_train, X_test, y_train, y_test, descriptors, cluster_list):
    for num_clusters in cluster_list:
        print(f"\n=== Running BoVW with {num_clusters} clusters ===")
        kmeans = KMeans(n_clusters=num_clusters, random_state=42)
        kmeans.fit(descriptors)
        print(f"Visual Vocabulary: {num_clusters} visual words created.")

        X_train_hist = img_hist(X_train, kmeans, num_clusters)
        X_test_hist = img_hist(X_test, kmeans, num_clusters)

        svm = SVC(kernel='linear', random_state=42)
        svm.fit(X_train_hist, y_train)
        y_pred = svm.predict(X_test_hist)

        acc = accuracy_score(y_test, y_pred)
        print(f"Accuracy: {acc * 100:.2f}%")

        cm = confusion_matrix(y_test, y_pred)
        disp = ConfusionMatrixDisplay(cm, display_labels=range(21))
        disp.plot(cmap='Greens')
        plt.title(f"Confusion Matrix - {num_clusters} Clusters")
        plt.savefig(f"confusion_matrix_{num_clusters}.png", bbox_inches='tight')
        plt.close()

desc_all = descriptors(X_train)
bovw_experiment(X_train, X_test, y_train, y_test, desc_all, [50, 100, 500])
```

4. Results and Discussion

4.1 Performance Summary

Number of Visual Words	Accuracy (%)
50	61.67
100	65.24
500	72.62

The accuracy improves as the number of clusters increases because a larger visual vocabulary captures finer local patterns and increases class separability.

4.2 Confusion Matrices

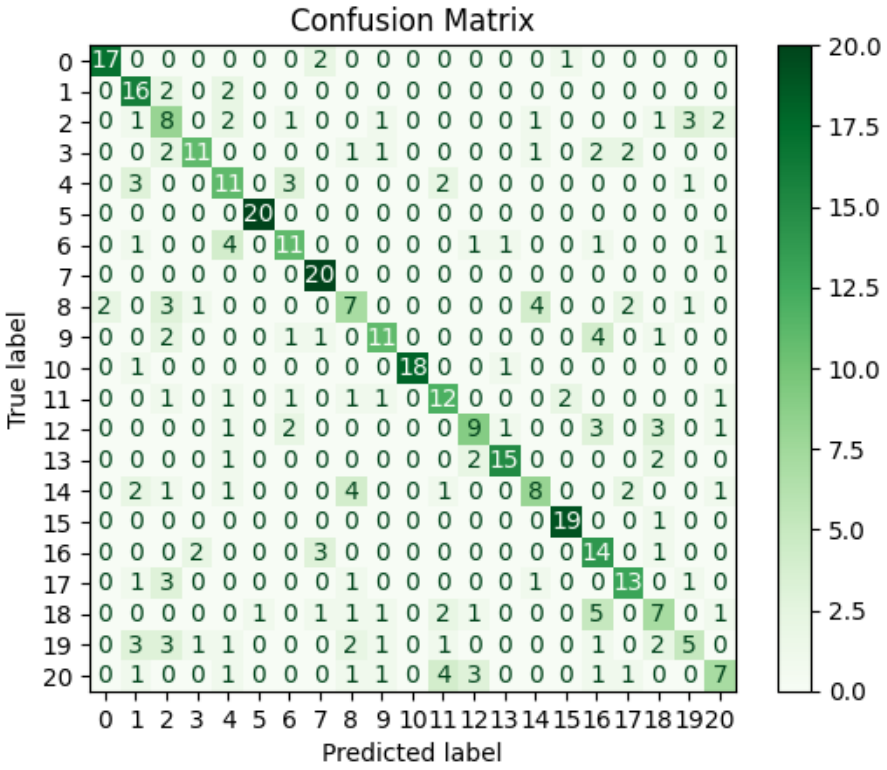

Figure 1: Confusion Matrix for 50 visual words (Accuracy: 61.67%)

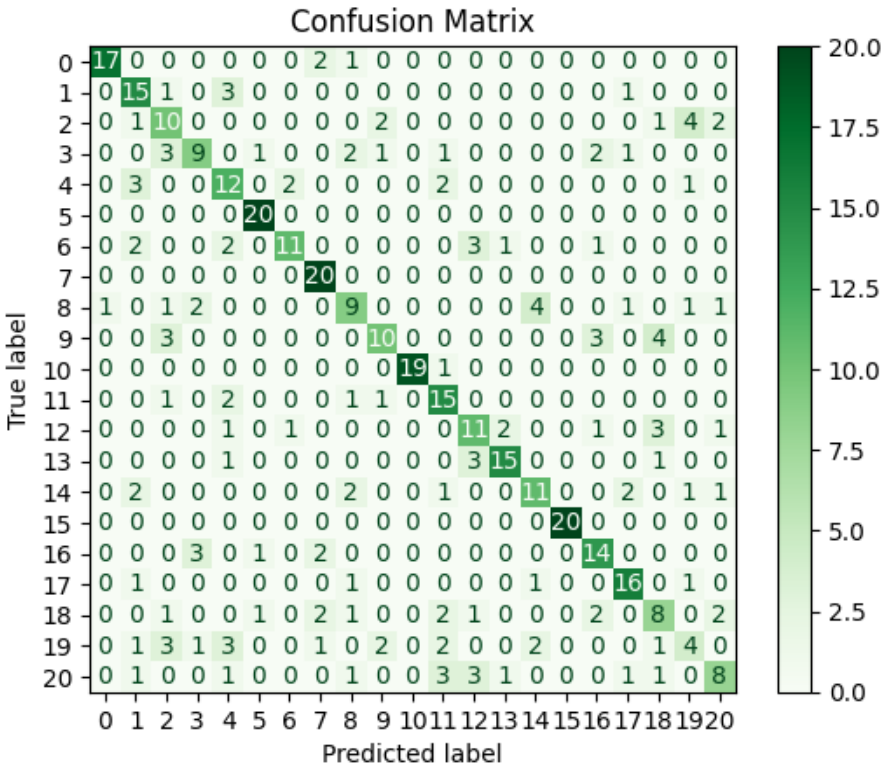

Figure 2: Confusion Matrix for 100 visual words (Accuracy: 65.24%)

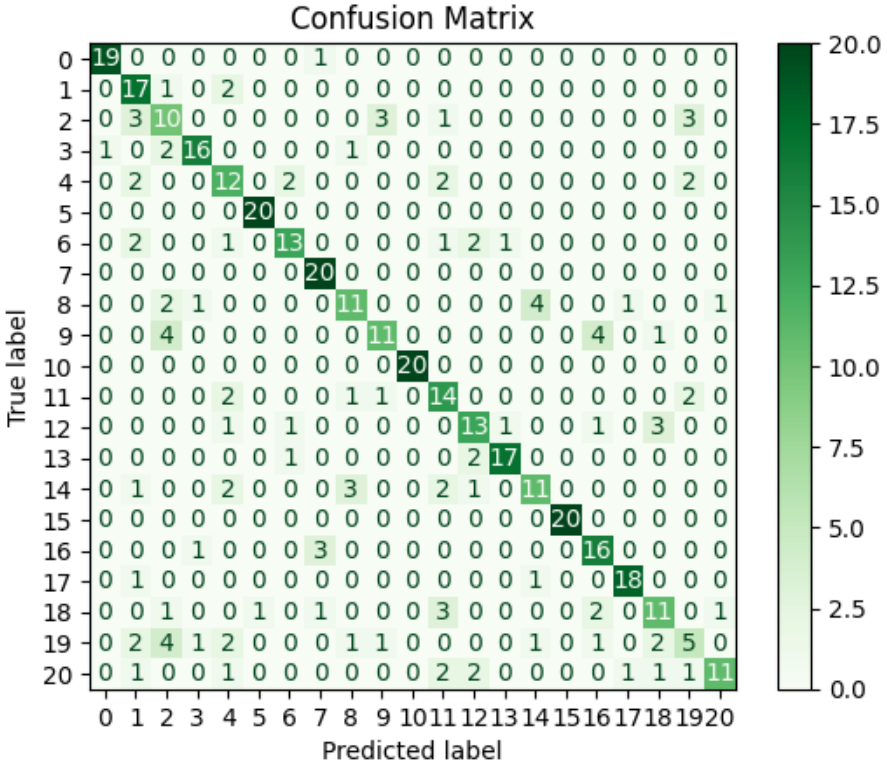

Figure 3: Confusion Matrix for 500 visual words (Accuracy: 72.62%)

5. Conclusion

The BoVW model achieved its best accuracy of 72.62% using 500 clusters. The results clearly show that increasing the number of visual words enhances performance, at the cost of computational complexity. This confirms the effectiveness of the BoVW pipeline combining SIFT descriptors, K-Means clustering, and SVM classification.