Lab 3 Report:SIFT Feature Extraction for Flower Image Classification

Islam Roubache

October 2025

1 Objective

The goal of this lab is to implement a complete pipeline for flower image classification using traditional computer vision methods. We use the **SIFT (Scale-Invariant Feature Transform)** algorithm to extract features and a **K-Nearest Neighbors (KNN)** model for classification.

2 Tools and Libraries

The experiment was implemented in Python using the following libraries:

- OpenCV (for SIFT feature extraction and image processing)
- NumPy (for numerical operations)
- scikit-learn (for KNN, data splitting, and evaluation)
- Matplotlib and Seaborn (for visualization)
- Pillow (for image display)

3 Configuration and Constants

```python

# CONFIGURATION
TRAIN_DATA_PATH = "train"
FLOWER_CLASSES = ["daisy","dandelion","rose","sunflower","tulip"]
TEST_SIZE = 0.20
RANDOM_STATE = 42
KNN_NEIGHBORS = 5
AUGMENT = True

SAMPLE_IMAGE_PATH = "test/Image_60.jpg"

CONF_MATRIX_OUT = "confusion_matrix.png"
VAL_EXAMPLES_OUT = "val_examples.png"
SAMPLE_SIFT_OUT = "sample_sift_keypoints.png"
```

Listing 1: Experiment configuration

4 Dataset Preparation

Each class folder (e.g., `train/daisy/`) contains several JPEG images. We iterate through these directories to build lists of file paths and labels.

```
train_root = Path(TRAIN_DATA_PATH)
all_image_paths = []
all_labels = []

for cls_idx, cls_name in enumerate(FLOWER_CLASSES):
    folder = train_root / cls_name
    if not folder.exists():
        print(f"Warning: missing folder {folder} - skipping")
        continue
    jpgs = sorted(folder.glob("*.jpg"))
    for p in jpgs:
        all_image_paths.append(str(p))
        all_labels.append(cls_idx)

print(f"Found {len(all_image_paths)} images across {len(set(all_labels))
    } classes.")
```

Listing 2: Loading image paths and labels

5 SIFT Feature Extraction and Augmentation

We define the SIFT detector and a small augmentation function to increase variability.

```
sift = cv2.SIFT_create()

def augment_image_cv2(img):
    outs = [img]
    outs.append(cv2.flip(img, 1))
    outs.append(cv2.rotate(img, cv2.ROTATE_90_CLOCKWISE))
    outs.append(cv2.rotate(img, cv2.ROTATE_90_COUNTERCLOCKWISE))
    return outs
```

Listing 3: SIFT creation and image augmentation

Now we compute descriptors for all images (and augmented versions), then store the **mean descriptor** (128-dimensional vector) for each image.

```
features = []
labels = []
feat_image_paths = []
skipped_no_kp = 0

print("Extracting SIFT descriptors and computing mean-per-image features ...")
for p, lbl in tqdm(zip(all_image_paths, all_labels), total=len(all_image_paths)):
    img = cv2.imread(p, cv2.IMREAD_GRAYSCALE)
    if img is None:
        continue

    if AUGMENT:
        imgs_to_process = augment_image_cv2(img)
    else:
        imgs_to_process = [img]

    for aug_i, im in enumerate(imgs_to_process):
        kps, desc = sift.detectAndCompute(im, None)
        if desc is None or len(desc) == 0:
            skipped_no_kp += 1
            continue
        mean_feat = desc.mean(axis=0)
        features.append(mean_feat)
        labels.append(lbl)
        feat_image_paths.append(f"{p}::aug{aug_i}" if AUGMENT else p)

features = np.array(features)
labels = np.array(labels)
print(f"Features extracted: {features.shape[0]} (skipped {skipped_no_kp})")
```

Listing 4: Extracting SIFT descriptors and computing mean features

6 Train / Validation Split

We split data into 80% for training and 20% for validation.

```
print("Splitting into train/validation...")
idx = np.arange(len(features))
train_idx, val_idx = train_test_split(idx, test_size=TEST_SIZE,
                                      stratify=labels, random_state=RANDOM_STATE)

X_train = features[train_idx]
X_val = features[val_idx]
y_train = labels[train_idx]
y_val = labels[val_idx]
```

Listing 5: Splitting dataset into train and validation sets

7 KNN Training

We encode class names and train the KNN model.

```
le = LabelEncoder()
le.fit(FLOWER_CLASSES)

y_train_names = [FLOWER_CLASSES[int(i)] for i in y_train]
y_train_enc = le.transform(y_train_names)

knn = KNeighborsClassifier(n_neighbors=KNN_NEIGHBORS)
knn.fit(X_train, y_train_enc)
print("KNN trained on mean-SIFT features.")
```

Listing 6: Label encoding and KNN training

8 Evaluation

We test the model on validation data and display metrics.

```
y_val_names = [FLOWER_CLASSES[int(i)] for i in y_val]
y_val_enc = le.transform(y_val_names)
y_val_pred_enc = knn.predict(X_val)
y_val_pred_names = le.inverse_transform(y_val_pred_enc)

acc = accuracy_score(y_val_enc, y_val_pred_enc)
print(f"Validation accuracy: {acc*100:.2f}%")

cm = confusion_matrix(y_val_enc, y_val_pred_enc)
plt.figure(figsize=(8,6))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
            xticklabels=le.classes_, yticklabels=le.classes_)
plt.title("Confusion Matrix (validation)")
plt.ylabel("True Label")
plt.xlabel("Predicted Label")
plt.tight_layout()
plt.savefig(CONF_MATRIX_OUT, dpi=200)
plt.close()
```

Listing 7: Validation evaluation and confusion matrix

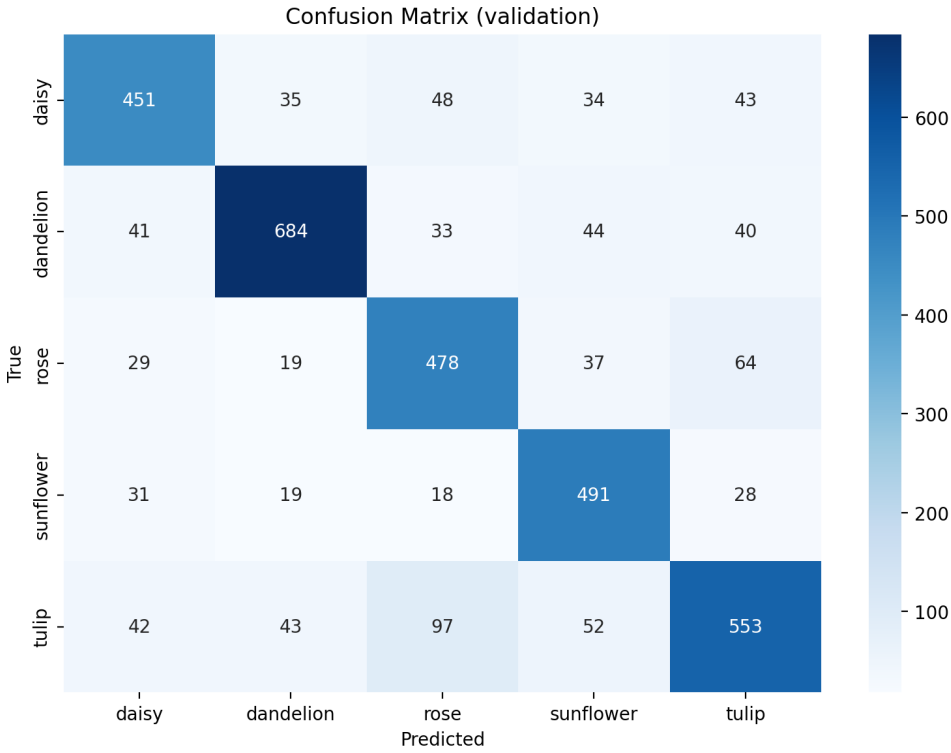

Figure 1: Validation confusion matrix.

9 Visualizing Results

A grid of validation examples (true vs predicted labels) was created to visualize performance.

```
n_show = min(8, len(X_val))
rng = np.random.default_rng(RANDOM_STATE)
sel = rng.choice(len(X_val), size=n_show, replace=False)
# Plot grid of samples with true and predicted labels
```

Listing 8: Displaying validation predictions

Figure 2: Example validation images with true (T) and predicted (P) labels.

10 SIFT Keypoint Visualization

To verify the extracted keypoints, we plot them on a sample image.

```
img_color = cv2.imread(SAMPLE_IMAGE_PATH)
img_gray = cv2.cvtColor(img_color, cv2.COLOR_BGR2GRAY)
kps, desc = sift.detectAndCompute(img_gray, None)
img_kp = cv2.drawKeypoints(img_color, kps, None,
            flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
plt.imshow(cv2.cvtColor(img_kp, cv2.COLOR_BGR2RGB))
plt.axis("off")
plt.title("SIFT Keypoints")
plt.show()
```

Listing 9: Visualizing SIFT keypoints on a sample

Figure 3: SIFT keypoints visualization on a sample image.

11 Conclusion

In this lab work, we implemented a full classical computer vision pipeline:

1. Loaded a labeled flower dataset.
2. Extracted SIFT features and computed per-image mean vectors.
3. Trained a KNN classifier with 5 neighbors.
4. Evaluated the model and obtained a validation accuracy of **77.47%**.

Although modern CNNs outperform such hand-crafted features, this exercise provides essential understanding of feature-based image classification and model evaluation in practice.