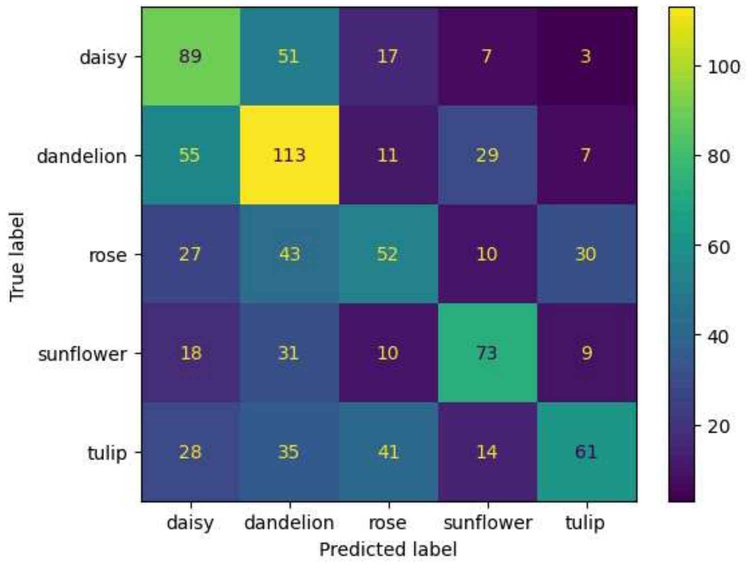
Task 1: k-Nearest-Neighbour Classifier

```
from google.colab import drive
drive.mount('/content/drive')
```

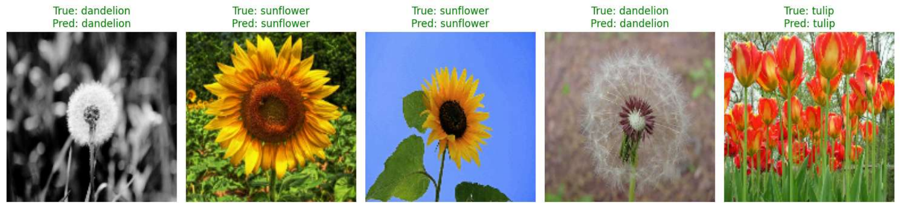
```
Mounted at /content/drive
```

```
# Import required libraries
import os
import numpy as np
import matplotlib.pyplot as plt
import cv2
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matr
from tqdm import tqdm
import time

# Function to preprocess images and extract color histograms
def preprocess_and_extract_histograms(base_path, labels, hist_size=6):
    X_histograms = []
    X_original = []
    y = []
    for label in labels:
        folder_path = os.path.join(base_path, label)
        for img_name in tqdm(os.listdir(folder_path)):
            img_path = os.path.join(folder_path, img_name)
            image = cv2.imread(img_path, cv2.IMREAD_COLOR)
            if image is None:
                continue
            image_resized = cv2.resize(image, (150, 150))
            hist = cv2.calcHist([image_resized], [0, 1, 2], None, [hist_size, hist_size, hist_size]
            hist = cv2.normalize(hist, hist).flatten()
            X_histograms.append(hist)
            X_original.append(image_resized)  # Save original image
            y.append(label)
    return np.array(X_histograms), np.array(X_original), np.array(y)

# Function to find the optimal k using validation data
def find_optimal_k(X_train, y_train, X_val, y_val, k_values):
    validation_accuracies = []
    for k in k_values:
        knn = KNeighborsClassifier(n_neighbors=k)
        knn.fit(X_train, y_train)
        y_val_pred = knn.predict(X_val)
        acc = accuracy_score(y_val, y_val_pred)
        validation_accuracies.append(acc)
        print(f"k={k}: Validation Accuracy = {acc:.4f}")
    optimal_k = k_values[np.argmax(validation_accuracies)]
    return optimal_k, validation_accuracies

# Function to evaluate the model and report metrics
def evaluate_model(knn, X_test, y_test):
    y_pred = knn.predict(X_test)
    acc = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred, average='weighted')
    recall = recall_score(y_test, y_pred, average='weighted')
    f1 = f1_score(y_test, y_pred, average='weighted')
    cm = confusion_matrix(y_test, y_pred, labels=np.unique(y_test))
    return acc, precision, recall, f1, cm, y_pred

# Function to compute average inference time
def compute_inference_time(knn, X_test, num_runs=10):
    times = []
    for _ in range(num_runs):
        start = time.time()
        for i in range(len(X_test)):
            knn.predict([X_test[i]])
        end = time.time()
        avg_time = (end - start) / len(X_test)
        times.append(avg_time)
    return np.mean(times)

# Function to visualize correctly or incorrectly classified images
def visualize_original_images(X_original, y_test, y_pred, correct=True, num_samples=5):
    indices = np.where(y_test == y_pred)[0] if correct else np.where(y_test != y_pred)[0]
    sample_indices = np.random.choice(indices, min(num_samples, len(indices)), replace=False)

    plt.figure(figsize=(15, 10))
    for i, idx in enumerate(sample_indices):
        plt.subplot(1, num_samples, i + 1)
        img = X_original[idx]
        plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
        plt.title(f"True: {y_test[idx]}\nPred: {y_pred[idx]}", color="green" if correct else "red")
        plt.axis('off')
    plt.tight_layout()
    plt.show()

# Main code
if __name__ == "__main__":
    BASE_PATH = "/content/drive/My Drive/Colab Notebooks/testing2/flowers"  # Dataset path
    LABELS = ["daisy", "dandelion", "rose", "sunflower", "tulip"]
    HIST_SIZES = [4, 6, 8]  # Test different histogram sizes

    for hist_size in HIST_SIZES:
        print(f"\n=== Testing Histogram Size: {hist_size} ===")
        # Load dataset
        X_histograms, X_original, y = preprocess_and_extract_histograms(BASE_PATH, LABELS, hist_siz
        print(f"Dataset loaded: {len(X_histograms)} samples")

        # Split dataset
        X_train, X_temp, y_train, y_temp = train_test_split(X_histograms, y, test_size=0.4, random_
        X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state

        # Split original images for visualization
        _, X_original_temp, _, y_original_temp = train_test_split(X_original, y, test_size=0.4, ran
        _, X_original_test, _, y_test_original = train_test_split(X_original_temp, y_original_temp,

        # Find optimal k
        k_values = range(1, 16, 2)
        optimal_k, validation_accuracies = find_optimal_k(X_train, y_train, X_val, y_val, k_values)
        print(f"\nOptimal k: {optimal_k}\n")

        # Train the final k-NN model
        knn = KNeighborsClassifier(n_neighbors=optimal_k)
        knn.fit(X_train, y_train)
        knn.fit(X_train, y_train)

        # Evaluate on test set
        acc, precision, recall, f1, cm, y_pred = evaluate_model(knn, X_test, y_test)
        print(f"\nTest Accuracy: {acc:.4f}")
        print(f"Precision: {precision:.4f}")
        print(f"Recall: {recall:.4f}")
        print(f"F1 Score: {f1:.4f}")

        # Analyze confusion matrix
        print("\nConfusion Matrix:")
        disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=np.unique(y_test))
        disp.plot()
        plt.show()

        # Find the most confused class pair
        cm_max_confusion = np.unravel_index(np.argmax(cm - np.diag(np.diag(cm))), cm.shape)
        print(f"\nMost confused classes: {LABELS[cm_max_confusion[0]]} and {LABELS[cm_max_confusion

        # Compute average inference time
        avg_inference_time = compute_inference_time(knn, X_test)
        print(f"\nAverage Inference Time: {avg_inference_time:.4f} seconds")

        # Visualize correctly classified images
        print("\nCorrectly classified images:")
        visualize_original_images(X_original_test, y_test_original, y_pred, correct=True, num_sampl

        # Visualize incorrectly classified images
        print("\nIncorrectly classified images:")
        visualize_original_images(X_original_test, y_test_original, y_pred, correct=False, num_samp
=== Testing Histogram Size: 4 ===
100%|██████████| 764/764 [00:28<00:00, 26.99it/s]
100%|██████████| 1052/1052 [00:53<00:00, 19.74it/s]
100%|██████████| 784/784 [00:22<00:00, 34.97it/s]
100%|██████████| 733/733 [00:33<00:00, 21.77it/s]
100%|██████████| 984/984 [00:55<00:00, 17.62it/s]
Dataset loaded: 4317 samples
k=1: Validation Accuracy = 0.4496
k=3: Validation Accuracy = 0.4635
k=5: Validation Accuracy = 0.4751
k=7: Validation Accuracy = 0.4855
k=9: Validation Accuracy = 0.4994
k=11: Validation Accuracy = 0.4855
k=13: Validation Accuracy = 0.4832
k=15: Validation Accuracy = 0.4832

Optimal k: 9


Test Accuracy: 0.4491
Precision: 0.4612
Recall: 0.4491
F1 Score: 0.4458

Confusion Matrix:
```



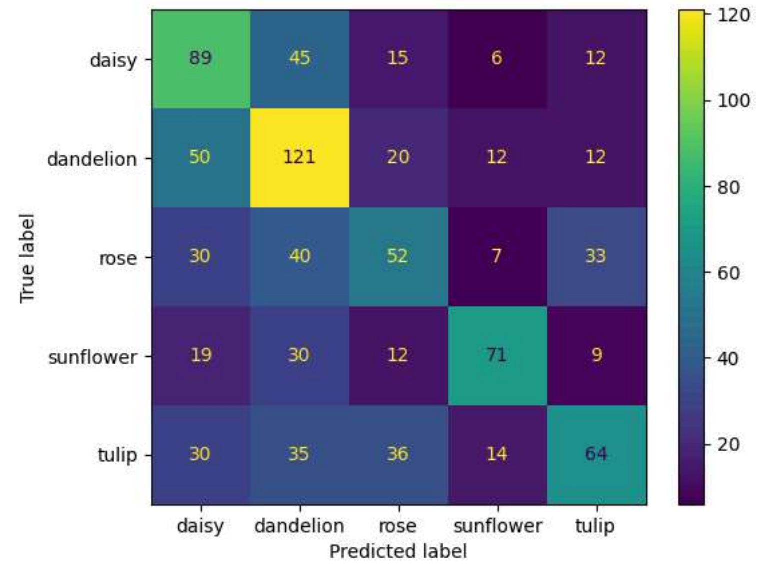
	daisy	dandelion	rose	sunflower	tulip
daisy	89	51	17	7	3
dandelion	55	113	11	29	7
rose	27	43	52	10	30
sunflower	18	31	10	73	9
tulip	28	35	41	14	61

```
Most confused classes: dandelion and daisy

Average Inference Time: 0.0019 seconds

Correctly classified images:
```



```
Incorrectly classified images:
```

```
=== Testing Histogram Size: 6 ===
100%|██████████| 764/764 [00:12<00:00, 60.17it/s]
100%|██████████| 1052/1052 [00:21<00:00, 49.09it/s]
100%|██████████| 784/784 [00:13<00:00, 57.60it/s]
100%|██████████| 733/733 [00:12<00:00, 60.00it/s]
100%|██████████| 984/984 [00:20<00:00, 47.76it/s]
Dataset loaded: 4317 samples
Dataset loaded: 4317 samples
k=1: Validation Accuracy = 0.4368
k=3: Validation Accuracy = 0.4299
k=5: Validation Accuracy = 0.4739
k=7: Validation Accuracy = 0.5006
k=9: Validation Accuracy = 0.4832
k=11: Validation Accuracy = 0.4751
k=13: Validation Accuracy = 0.4820
k=15: Validation Accuracy = 0.4925

Optimal k: 7


Test Accuracy: 0.4595
Precision: 0.4696
Recall: 0.4595
F1 Score: 0.4571

Confusion Matrix:
```

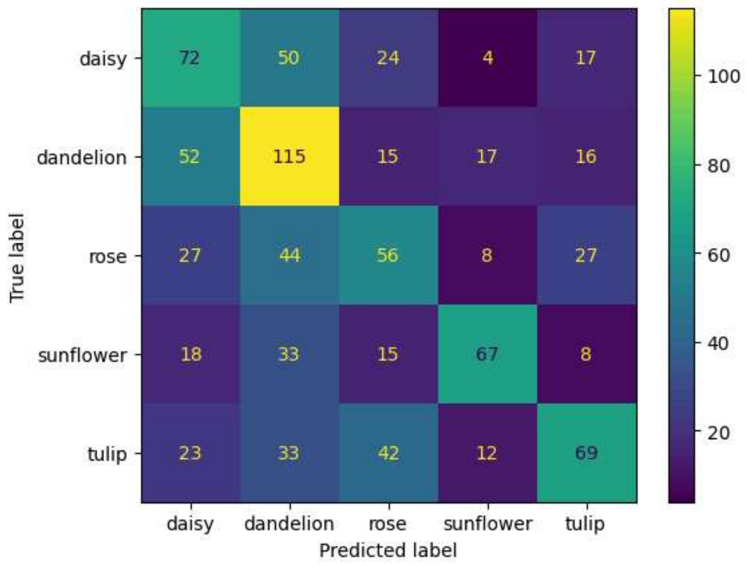


```
Most confused classes: dandelion and daisy

Average Inference Time: 0.0036 seconds

Correctly classified images:
```

```
Incorrectly classified images:
```

```
=== Testing Histogram Size: 8 ===
100%|██████████| 764/764 [00:07<00:00, 101.51it/s]
100%|██████████| 1052/1052 [00:11<00:00, 89.74it/s]
100%|██████████| 784/784 [00:07<00:00, 103.72it/s]
100%|██████████| 733/733 [00:09<00:00, 74.45it/s]
100%|██████████| 984/984 [00:10<00:00, 95.88it/s]
Dataset loaded: 4317 samples
k=1: Validation Accuracy = 0.4287
k=3: Validation Accuracy = 0.4206
k=5: Validation Accuracy = 0.4600
k=7: Validation Accuracy = 0.4670
k=9: Validation Accuracy = 0.4670
k=11: Validation Accuracy = 0.4600
k=13: Validation Accuracy = 0.4705
```

```
k=15: Validation Accuracy = 0.4820

Optimal k: 15


Test Accuracy: 0.4387
Precision: 0.4512
Recall: 0.4387
F1 Score: 0.4395

Confusion Matrix:
```



```
Most confused classes: dandelion and daisy

Average Inference Time: 0.0063 seconds

Correctly classified images:
```

```
Incorrectly classified images:
```