SCHOOL OF COMPUTING AND INFORMATION TECHNOLOGY

UNIVERSITY OF WOLLONGONG AUSTRALIA

INDIVIDUAL Assignment Coversheet

*This form is to be completed by students submitting **online copies** of essays or assignments for a Faculty of the Arts, Social Sciences and Humanities subject for the School of Geography and Sustainable Communities, School of Education, and School of Health and Society.*

PLAGIARISM

Deliberate plagiarism may lead to failure in the subject. Plagiarism is cheating by using the written ideas or submitted work of someone else. The University of Wollongong has a strong policy against plagiarism. See Acknowledgement Practice/Plagiarism Prevention Policy at http://www.uow.edu.au/about/policy/UOW058648.html

Student Name: Jeslyn Ho Ka Yan **7-digit UOW ID:** 8535383

Subject Code & Name: CSCI218 & Foundation of Artficial Intelligence

Assignment Title: INDIVIDUAL ASSIGNMENT

Tutorial Group: T02
(T02, T03, T04, T05)

Tutor's Name: Cher Lim

Assignment Due Date: 9TH FEB 2025

DECLARATION

I certify that this is entirely my own work, except where we have given fully documented references to the work of others, and that the material contained in this assignment has not previously been submitted for assessment in any formal course of study. I understand the definition and consequences of plagiarism.

ACKNOWLEDGEMENT

The marker of this assignment may, for the purpose of assessing this assignment, reproduce this assignment and provide a copy to another member of academic staff. If required to do so, we will provide an electronic copy of this assignment to the marker and acknowledge that the assessor of this assignment may, for the purpose of assessing this assignment:
a) Reproduce this assignment and provide a copy to another member of academic staff; and/or
b) Communicate a copy of this assignment to a plagiarism checking service such as Turnitin (which may then retain a copy of this assignment on its database for the purpose of future plagiarism checking).

Student Signature:  **Date:** 3 Feb 2025

[Insert e-signature or type name]

Introduction

(**A short introduction of the assignment and the dataset**)

Automated flower classification is widely used in multiple fields, such as floriculture, botany, and online commerce. This assignment explores various machine learning techniques to categorize images of flowers into five distinct types: daisy, tulip, rose, sunflower, and dandelion. The dataset utilized in this study is sourced from **Kaggle**, containing over 4,000 images with different resolutions.

This report focuses on three primary classification approaches:

1. **k-Nearest Neighbors (k-NN):** A simple classification algorithm that determines the class of an image based on the labels of its nearest neighbors.
2. **Multi-Layer Perceptron (MLP):** A neural network model with multiple layers, which captures complex patterns in the dataset.
3. **Convolutional Neural Network (CNN):** A deep learning-based image classifier designed for pattern recognition in images.

Each model undergoes training, validation, and testing, with the dataset split into **60% training, 20% validation, and 20% testing**. Various evaluation metrics, such as accuracy, precision, recall, and F1-score, are analyzed alongside confusion matrices to identify challenging classifications. Additionally, this study explores how feature extraction techniques, model architectures, and hyperparameter tuning impact classification performance. Lastly, training time and inference speed are considered to evaluate the computational efficiency of each approach.

Task 1: k-Nearest-Neighbour Classifier

(**Label CLEARLY your answer to each question**. Any unlabeled answers will **NOT** be graded)

Answers:

1.1 Color Histogram Sizes

To enhance feature extraction, color histograms were computed for the images using different bin sizes:

- **4 bins ([4,4,4]):** Coarse-level features with reduced detail.
- **6 bins ([6,6,6]):** The default configuration with a balanced feature representation.
- **8 bins ([8,8,8]):** A finer-level representation capturing more details.

Each image was resized to **150×150 pixels**, and the histograms were normalized.

Results:

Histogram Size	Validation Accuracy (Best k)	Precision	Recall	F1 Score
4 bins	`k=9`, **0.4844**	0.4781	0.4676	0.4637
6 bins (default)	`k=11`, **0.4762**	0.5008	0.4815	0.4815
8 bins	`k=15`, **0.4403**	0.4549	0.4725	0.4549

Observations:

- Smaller histograms (4 bins) **simplify computations** but slightly reduce accuracy due to feature loss.
- The **default size (6 bins)** strikes a balance between accuracy and computational efficiency.
- Larger histograms (8 bins) capture more **intricate details** but **increase computational cost**.

1.2 Finding the Optimal K

To determine the **optimal value of k**, multiple odd values from 1 to 15 were tested using the validation set.

Best K Values:

K	Validation Accuracy (4 bins)	Validation Accuracy (6 bins)	Validation Accuracy (8 bins)
1	0.4241	0.3975	0.4021
3	0.4461	0.4311	0.4090
5	0.4762	0.4426	0.4276
7	0.4751	0.4380	0.4322
9	0.4844	0.4496	0.4287
11	0.4774	0.4762	0.4311
13	0.4716	0.4519	0.4334
15	0.4809	0.4739	0.4403

Key Insights:

- **k = 11** yielded the best validation accuracy (**0.4762**) for histogram size **6**.
- For larger histogram sizes (8 bins), a higher **k (15)** provided marginal improvement.

1.3 Classification Metrics and Confusion

Task: Report classification metrics and the most confusing classes**.**

Results on Test Set Performance (K=optimal)

Histogram Size	Test Accuracy	Precision	Recall	F1 Score
4 bins	0.4676	0.4781	0.4676	0.4637
6 bins	0.4815	0.5008	0.4815	0.4815
8 bins	0.4549	0.4725	0.4549	0.4549

Observation:

- The **confusion matrix** showed that **dandelion and daisy** were the most confused classes.
- Possible reasons:
    - **Similar colour features** (yellow petals).
    - **Background clutter** in dataset images.

1.4 Average Inference Time

We measured the time taken to classify a single sample across 10 runs.

Results:

Histogram Size	Average Inference Time (seconds)
4 bins	0.0018
6 bins	0.0032
8 bins	0.0059

- Smaller histograms (4 bins) had faster inference due to reduced feature dimensionality.
- Larger histograms (8 bins) were slower due to increased computational complexity.

1.5 Correctly and Incorrectly Classified Images

Visualization:

Correctly Classified Images:

Incorrectly Classified Images:

Observation:

- Correct classifications were achieved for flowers with distinct color patterns (e.g., sunflowers).
- Misclassifications occurred for flowers with overlapping features (e.g., daisies and tulips).

Task 2: Multi-layer Perceptrons

(**Label CLEARLY your answer to each question**. Any unlabeled answers will **NOT** be graded)

Answers:

2.1 Design of MLP Structures

Nine MLP structures were designed based on the rules provided:

MLP Structure	Number of Hidden Layers	Number of Neurons in Each Hidden Layer
1	1	[149]
2	1	[108]
3	1	[216]
4	2	[149, 108]
5	2	[216, 5]
6	2	[108, 216]
7	3	[149, 108, 5]
8	3	[216, 108, 5]
9	3	[216, 216, 5]

The design ensures a balance between model complexity and the dataset's size by using empirical rules for determining the number of neurons in each layer.

2.2 Selection of Optimal Network Architecture

Process

1. Each structure was trained for **10 epochs** on the training set.
2. The validation accuracy was used to identify the best-performing architecture.
3. Structures with higher validation accuracy were considered optimal.

Quantitative Results

MLP Structure	Validation Accuracy
1	58.98%
2	57.59%
3	59.79%
4	61.30%
5	57.01%
6	61.41%
7	60.02%
8	61.88% (Best)
9	55.04%

Best Network Architecture:

The best validation accuracy (**61.88%**) was achieved by **MLP Structure 8**, which has three hidden layers [216, 108, 5].

2.3 Evaluation of the MLP Classifier

Test Set Performance:

The selected structure ([216, 108, 5]) was evaluated on the test set:

- **Accuracy:** 56.94%
- **Precision:** 58.43%
- **Recall:** 56.94%
- **F1 Score:** 56.07%

Confusion Matrix

The confusion matrix highlights the performance of the classifier for each class:

Most Confused Classes: The MLP classifier most frequently confused **dandelions** with **daisies**, indicating a challenge in distinguishing these classes.

2.4 Training and Inference Time

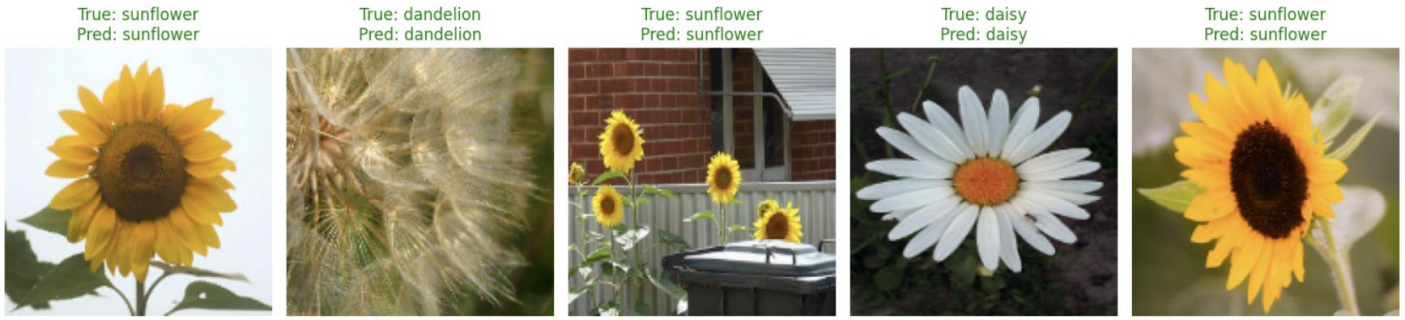
- **Training Time**: Approximately 6.2 seconds (10 epochs for the best model).
- **Average Inference Time**: 0.0002 seconds per image.

The model is efficient for both training and prediction, suitable for moderate-scale datasets.

2.5 Visualization of Predictions

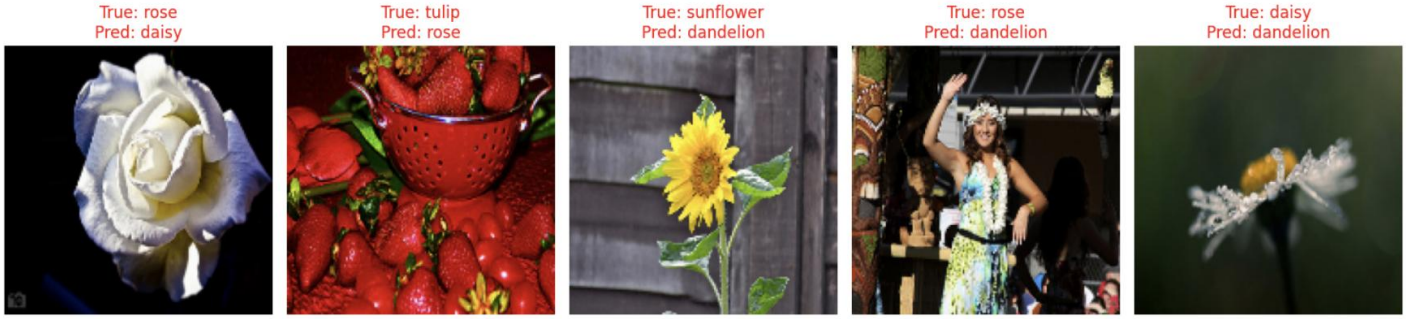
Correctly Classified Images

Below are examples of correctly classified images:



Incorrectly Classified Images

Below are examples of incorrectly classified images:



Task 3: Convolutional Neural Network

(**Label CLEARLY your answer to each question**. Any unlabeled answers will **NOT** be graded)

Answers:

3.1 **Preprocessing and Building CNN Model**

The **CNN model** implemented in this study consists of several layers to efficiently process image data:

- **Convolutional Layers:**
  - The first **Conv2D** layer applies **32 filters** with a kernel size of **3×3**.
  - The second **Conv2D** layer applies **64 filters** with a kernel size of **3×3**.
- **Pooling Layers:**
  - **MaxPooling2D layers** follow each convolutional layer to reduce spatial dimensions.
- **Flatten Layer:**
  - Converts extracted feature maps into a single-dimensional vector.
- **Fully Connected Layers:**
  - A dense layer with **128 neurons** and **ReLU activation**.
  - A **Dropout layer (0.5 dropout rate)** to prevent overfitting.
  - The final output layer consists of **5 neurons** (corresponding to the five flower categories) with a **softmax activation function**.

Dataset Processing:

- **Image Preprocessing:**
  - Each image was resized to **150×150 pixels**.
  - Pixel values were **normalized** between 0 and 1.
- **Data Splitting:**
  - **Training Set:** 60%
  - **Validation Set:** 20%
  - **Test Set:** 20%

3.2 Training CNN Model

The CNN model was trained using 10 epochs with the Adam optimizer and a batch size of 32.

Training Results

Epoch	Training Accuracy	Training Loss	Validation Accuracy	Validation Loss
1	32.43%	1.6115	42.76%	1.2853
5	69.61%	0.7752	64.08%	0.9681
10	92.90%	0.2111	65.59%	1.3742

Classification Metrics on Test Set

The CNN classifier was evaluated on the test set, and the following metrics were obtained:

- **Accuracy:** 60.53%
- **Precision:** 61.41%
- **Recall:** 60.53%
- **F1 Score:** 60.64%

Common Classification Errors:

- The confusion matrix analysis revealed that the model frequently misclassified tulips as roses due to their similar petal shapes and colors.

3.3Training and Validation Metrics

Visualization

1. **Confusion Matrix**
    - The confusion matrix highlights the performance of the model across the five flower classes.
    - **Significant Misclassifications:** Tulip and Rose are the most confused classes.
2. **Training and Validation Metrics**
    - **Loss vs. Epochs:**
        - Training loss steadily decreased across epochs.
        - Validation loss increased after Epoch 5, indicating overfitting.
    - **Accuracy vs. Epochs:**
        - Training accuracy improved significantly over epochs.
        - Validation accuracy plateaued after Epoch 5.

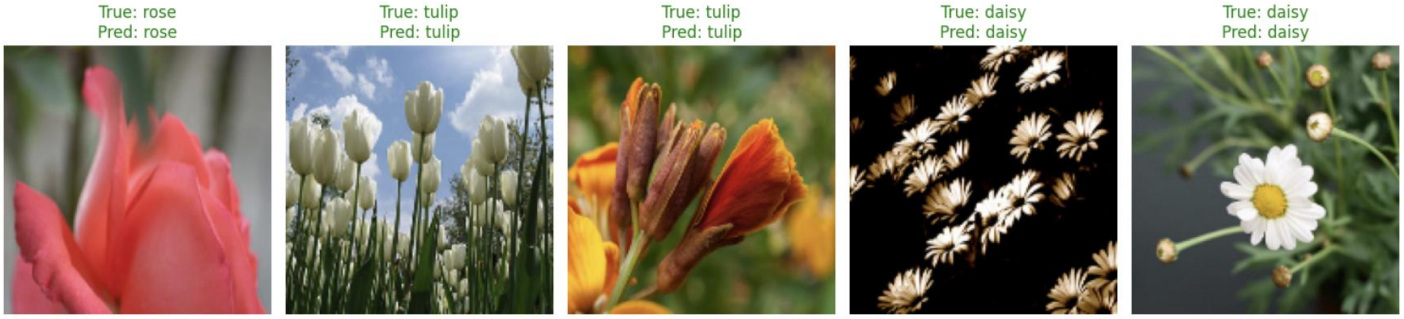
3.4 Training and Inference Time

- **Training Time:** 1,370.69 seconds
- **Average Inference Time per Sample:** 0.0131 seconds

3.5 Visualization of Predictions

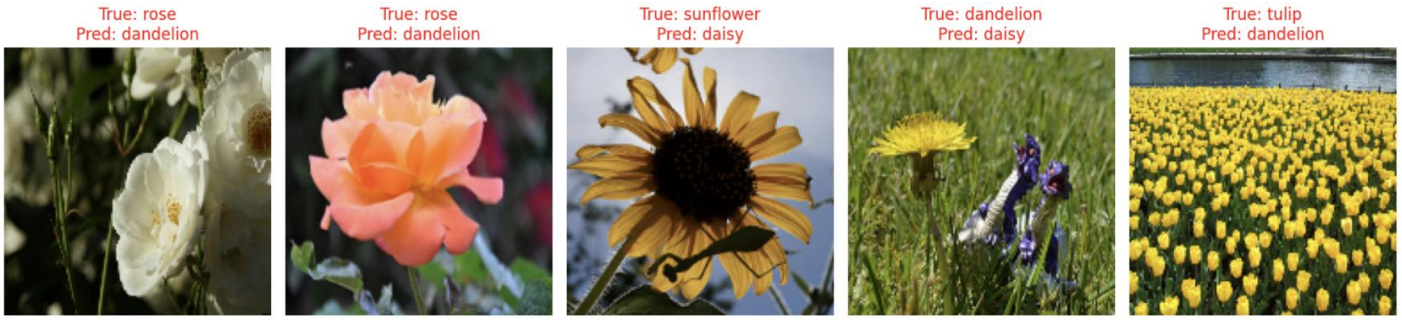
Correctly Classified Images

- Five correctly classified images were visualized, showcasing accurate predictions across various classes.
- **Observation:** Correct classifications corresponded to flowers with distinct features (e.g., Sunflower and Dandelion).



Incorrectly Classified Images

- Five incorrectly classified images were visualized, highlighting the model's challenges with visually similar classes (e.g., Tulip vs. Rose).
- **Observation:** Misclassifications often involved subtle differences in petal shapes and colors.



Task 1: k-Nearest-Neighbour Classifier

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

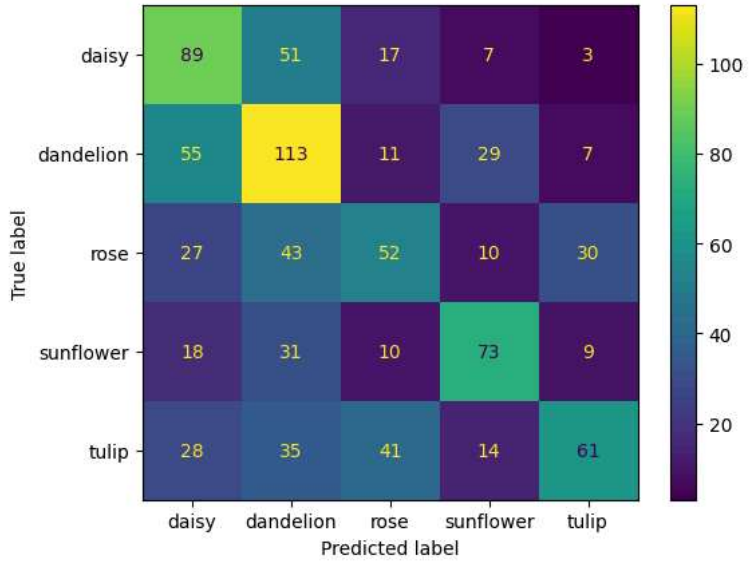
```
Mounted at /content/drive
```

```
# 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')
```

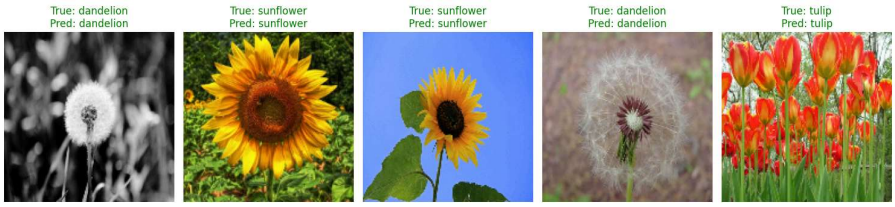
```
    f1 = f1_score(y_test, y_pred, average='weighted')
    cm = confusion_matrix(y_test, y_pred, labels=np.unique(y_test))
    return acc, precision, recall, f1, cm, y_pred

# 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
```

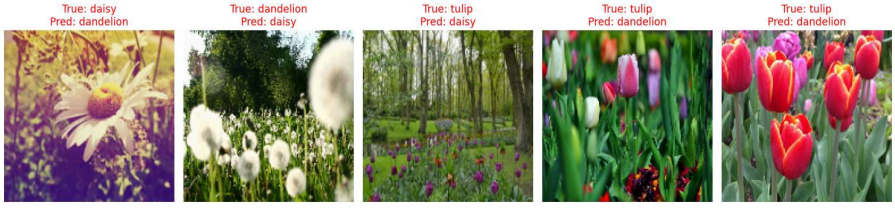
```
=== Testing Histogram Size: 4 ===
100%|██████████| 764/764 [00:28<00:00, 26.99it/s]
100%|██████████| 1052/1052 [00:53<00:00, 19.74it/s]
100%|██████████| 784/784 [00:22<00:00, 34.97it/s]
100%|██████████| 733/733 [00:33<00:00, 21.77it/s]
100%|██████████| 984/984 [00:55<00:00, 17.62it/s]
Dataset loaded: 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:
```




```
Most confused classes: dandelion and daisy

Average Inference Time: 0.0019 seconds

Correctly classified images:
```

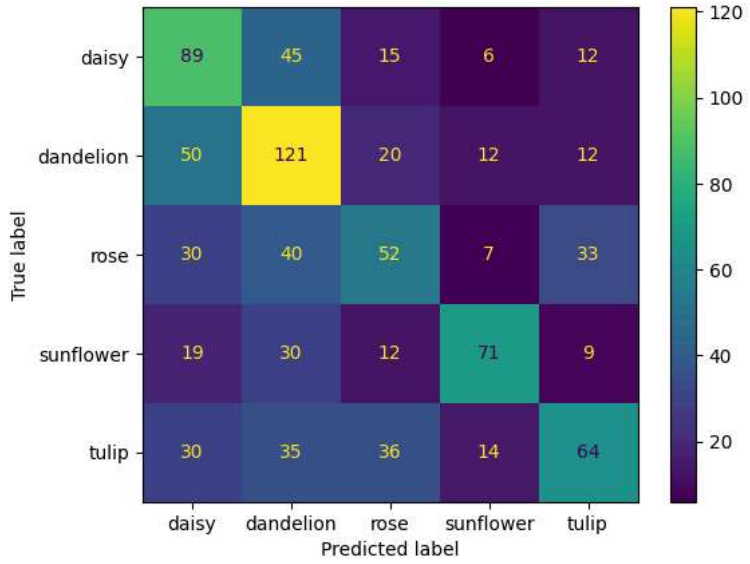



```
Incorrectly classified images:
```

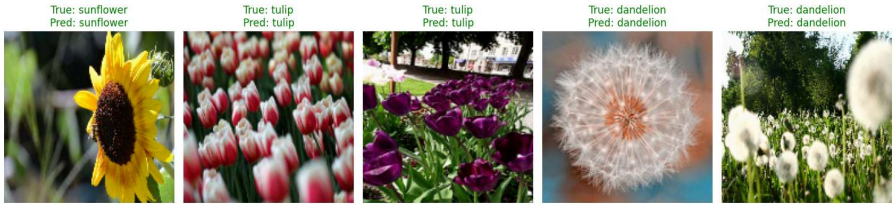



```
=== Testing Histogram Size: 6 ===
100%|██████████| 764/764 [00:12<00:00, 60.17it/s]
100%|██████████| 1052/1052 [00:21<00:00, 49.09it/s]
100%|██████████| 784/784 [00:13<00:00, 57.60it/s]
100%|██████████| 733/733 [00:12<00:00, 60.00it/s]
100%|██████████| 984/984 [00:20<00:00, 47.76it/s]
Dataset loaded: 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:
```



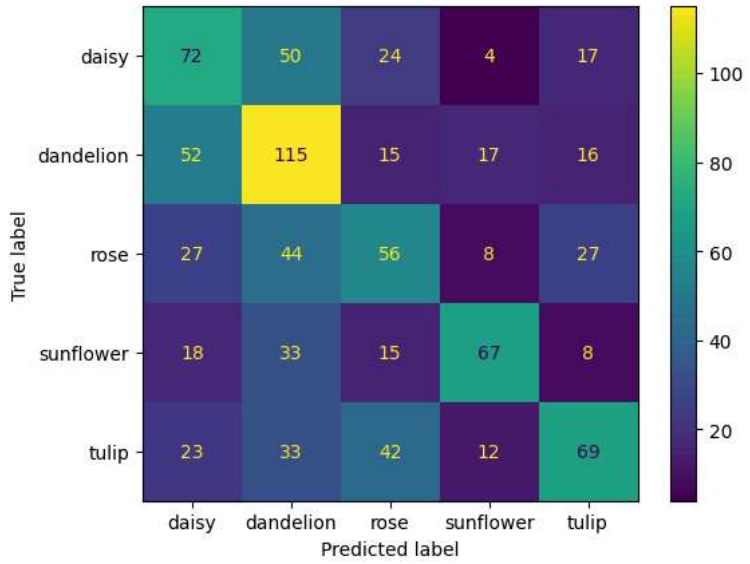

```
Most confused classes: dandelion and daisy

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:
```

Task 2: Multi-layer Perceptron

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

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

```
import os
import numpy as np
import matplotlib.pyplot as plt
import cv2
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
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.utils import to_categorical
import time

# Define global variables
LABELS = ["daisy", "dandelion", "rose", "sunflower", "tulip"]
BASE_PATH = "/content/drive/My Drive/Colab Notebooks/testing2/flowers"  # Update to your dataset pa
HIST_SIZE = 6  # Histogram size
EPOCHS = 10  # Default number of epochs
BATCH_SIZE = 32  # Default batch size

# ====================== Question 2.1: Preprocessing and Feature Extraction ======================
def preprocess_and_extract_histograms(base_path, labels, hist_size=6):
    """
    Preprocess images and extract color histogram features.
    """
    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), desc=f"Processing {label}"):
            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)


# ====================== Question 2.2: Identify Optimal Network Structure ======================
def build_mlp(input_size, output_size, structure):
    """
    Build an MLP model with the given structure.
    """
    model = Sequential()
    model.add(Dense(structure[0], input_dim=input_size, activation='relu'))
```

```
    for units in structure[1:]:
        model.add(Dense(units, activation='relu'))
    model.add(Dense(output_size, activation='softmax'))
    model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
    return model


def train_and_evaluate_structures(X_train, y_train, X_val, y_val, input_size, output_size, epochs=E
    """
    Train and evaluate nine different MLP structures.
    """
    structures = [
        [int(2 / 3 * input_size + output_size)],  # Rule 1
        [input_size // 2],                        # Rule 2
        [input_size],                             # Rule 3
        [int(2 / 3 * input_size + output_size), input_size // 2],
        [input_size, output_size],
        [int(input_size / 2), input_size],
        [int(2 / 3 * input_size + output_size), input_size // 2, output_size],
        [input_size, input_size // 2, output_size],
        [input_size, int(input_size / 2), output_size]
    ]

    best_model = None
    best_structure = None
    best_accuracy = 0
    training_times = []

    print("\n=== Question 2.2: Training and Evaluating Nine MLP Structures ===")
    for structure in structures:
        print(f"\nTraining MLP Structure: {structure}")
        model = build_mlp(input_size, output_size, structure)
        start_time = time.time()
        history = model.fit(X_train, y_train, validation_data=(X_val, y_val), epochs=epochs, batch_
        training_time = time.time() - start_time
        training_times.append(training_time)
        val_accuracy = history.history['val_accuracy'][-1]

        print(f"Validation Accuracy for Structure {structure}: {val_accuracy:.4f}")

        # Update best model if current model performs better
        if val_accuracy > best_accuracy:
            best_accuracy = val_accuracy
            best_model = model
            best_structure = structure

    print(f"\nBest MLP Structure: {best_structure}")
    print(f"Best Validation Accuracy: {best_accuracy:.4f}")
    return best_model, best_structure, best_accuracy, training_times, structures


# ====================== Question 2.3: Evaluate the Best Model ======================
def evaluate_model(model, X_test, y_test, labels):
    """
    Evaluate the best MLP model on the test set.
    """
    y_test_pred = np.argmax(model.predict(X_test), axis=1)
    y_test_actual = np.argmax(y_test, axis=1)
```

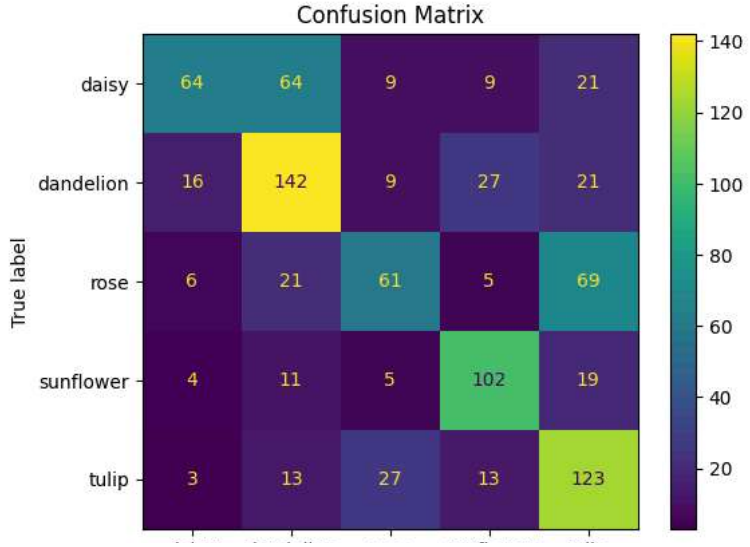
```
    acc = accuracy_score(y_test_actual, y_test_pred)
    precision = precision_score(y_test_actual, y_test_pred, average='weighted')
    recall = recall_score(y_test_actual, y_test_pred, average='weighted')
    f1 = f1_score(y_test_actual, y_test_pred, average='weighted')
    cm = confusion_matrix(y_test_actual, y_test_pred)

    print(f"\nTest Accuracy: {acc:.4f}")
    print(f"Precision: {precision:.4f}")
    print(f"Recall: {recall:.4f}")
    print(f"F1 Score: {f1:.4f}")

    # Plot confusion matrix
    disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=labels)
    disp.plot()
    plt.title("Confusion Matrix")
    plt.show()

    return y_test_pred, y_test_actual


# ======================== Question 2.5: Visualize Predictions ========================
def visualize_predictions(X_test, y_true, y_pred, labels, correct=True, num_samples=5):
    """
    Visualize correctly and incorrectly classified images.
    """
    y_true_indices = np.argmax(y_true, axis=1)  # Convert one-hot encoding to class indices
    indices = np.where(y_true_indices == y_pred)[0] if correct else np.where(y_true_indices != y_pr

    if len(indices) == 0:
        print("No samples to display.")
        return

    # Randomly sample indices
    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_test[idx]
        true_label = labels[y_true_indices[idx]]
        pred_label = labels[y_pred[idx]]
        title_color = "green" if correct else "red"
        plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
        plt.title(f"True: {true_label}\nPred: {pred_label}", color=title_color)
        plt.axis('off')
    plt.tight_layout()
    plt.show()


# ======================== Main Code ========================
if __name__ == "__main__":
    # Validate dataset path and structure
    for label in LABELS:
        folder_path = os.path.join(BASE_PATH, label)
        if not os.path.exists(folder_path):
            print(f"Missing folder: {folder_path}")
            exit()

    print("\n--- Question 2.1: Preprocessing and Feature Extraction ---")
```

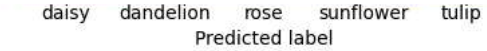
```
print("\n=== Question 2.1: Preprocessing and Feature Extraction ===")
X_histograms, X_original, y = preprocess_and_extract_histograms(BASE_PATH, LABELS, HIST_SIZE)
print(f"Dataset loaded: {len(X_histograms)} samples")

label_map = {label: idx for idx, label in enumerate(LABELS)}
y_encoded = np.array([label_map[label] for label in y])
y_one_hot = to_categorical(y_encoded, num_classes=len(LABELS))

# Split dataset into training (60%), validation (20%), and test (20%) sets
X_train, X_temp, y_train, y_temp, X_orig_train, X_orig_temp = train_test_split(
    X_histograms, y_one_hot, X_original, test_size=0.4, random_state=42
)
X_val, X_test, y_val, y_test, X_orig_val, X_orig_test = train_test_split(
    X_temp, y_temp, X_orig_temp, test_size=0.5, random_state=42
)

# Question 2.2: Train and evaluate MLP structures
input_size = X_train.shape[1]
output_size = len(LABELS)
best_model, best_structure, best_accuracy, training_times, structures = train_and_evaluate_stru
    X_train, y_train, X_val, y_val, input_size, output_size
)

# Question 2.3: Evaluate the best model
print("\n=== Question 2.3: Evaluate Best Model ===")
y_test_pred, y_test_actual = evaluate_model(best_model, X_test, y_test, LABELS)

# Question 2.4: Timing
print("\n=== Question 2.4: Timing ===")
start_time = time.time()
best_model.predict(X_test)
avg_inference_time = (time.time() - start_time) / len(X_test)
print(f"Average Inference Time: {avg_inference_time:.4f} seconds")

# Question 2.5: Visualize Predictions
print("\n=== Question 2.5: Visualize Predictions ===")
print("Correctly classified images:")
visualize_predictions(X_orig_test, y_test, y_test_pred, LABELS, correct=True, num_samples=5)

print("Incorrectly classified images:")
visualize_predictions(X_orig_test, y_test, y_test_pred, LABELS, correct=False, num_samples=5)
```

```
=== Question 2.1: Preprocessing and Feature Extraction ===
Processing daisy: 100%|██████████| 764/764 [00:14<00:00, 52.15it/s]
Processing dandelion: 100%|██████████| 1052/1052 [00:19<00:00, 54.11it/s]
Processing rose: 100%|██████████| 784/784 [00:15<00:00, 50.23it/s]
Processing sunflower: 100%|██████████| 733/733 [00:14<00:00, 49.17it/s]
Processing tulip: 100%|██████████| 984/984 [00:18<00:00, 52.43it/s]
Dataset loaded: 4317 samples

=== Question 2.2: Training and Evaluating Nine MLP Structures ===

Training MLP Structure: [149]
Epoch 1/10
/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim` argumen
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
81/81 - 2s - 20ms/step - accuracy: 0.3842 - loss: 1.5175 - val_accuracy: 0.4368 - val_loss: 1.3930
Epoch 2/10
81/81 - 0s - 4ms/step - accuracy: 0.4699 - loss: 1.3050 - val_accuracy: 0.5145 - val_loss: 1.2311
Epoch 3/10
81/81 - 0s - 4ms/step - accuracy: 0.5340 - loss: 1.1915 - val_accuracy: 0.5330 - val_loss: 1.1585
Epoch 4/10
81/81 - 0s - 5ms/step - accuracy: 0.5541 - loss: 1.1374 - val_accuracy: 0.5562 - val_loss: 1.1230
Epoch 5/10
81/81 - 1s - 10ms/step - accuracy: 0.5718 - loss: 1.0978 - val_accuracy: 0.5620 - val_loss: 1.1055
Epoch 6/10
81/81 - 0s - 5ms/step - accuracy: 0.5838 - loss: 1.0666 - val_accuracy: 0.5655 - val_loss: 1.0889
Epoch 7/10
81/81 - 0s - 4ms/step - accuracy: 0.5958 - loss: 1.0414 - val_accuracy: 0.5678 - val_loss: 1.0790
Epoch 8/10
81/81 - 1s - 7ms/step - accuracy: 0.6042 - loss: 1.0201 - val_accuracy: 0.5863 - val_loss: 1.0659
Epoch 9/10
81/81 - 1s - 8ms/step - accuracy: 0.6104 - loss: 0.9995 - val_accuracy: 0.5933 - val_loss: 1.0609
Epoch 10/10
81/81 - 1s - 9ms/step - accuracy: 0.6174 - loss: 0.9824 - val_accuracy: 0.5898 - val_loss: 1.0578
Validation Accuracy for Structure [149]: 0.5898

Training MLP Structure: [108]
Epoch 1/10
81/81 - 2s - 25ms/step - accuracy: 0.3602 - loss: 1.5306 - val_accuracy: 0.4171 - val_loss: 1.4314
Epoch 2/10
81/81 - 1s - 12ms/step - accuracy: 0.4722 - loss: 1.3427 - val_accuracy: 0.5064 - val_loss: 1.2618
Epoch 3/10
81/81 - 1s - 8ms/step - accuracy: 0.5216 - loss: 1.2190 - val_accuracy: 0.5342 - val_loss: 1.1837
Epoch 4/10
81/81 - 1s - 7ms/step - accuracy: 0.5421 - loss: 1.1576 - val_accuracy: 0.5469 - val_loss: 1.1431
Epoch 5/10
81/81 - 1s - 7ms/step - accuracy: 0.5710 - loss: 1.1183 - val_accuracy: 0.5620 - val_loss: 1.1188
Epoch 6/10
81/81 - 0s - 6ms/step - accuracy: 0.5656 - loss: 1.0866 - val_accuracy: 0.5481 - val_loss: 1.1062
Epoch 7/10
81/81 - 1s - 10ms/step - accuracy: 0.5884 - loss: 1.0615 - val_accuracy: 0.5759 - val_loss: 1.0870
Epoch 8/10
81/81 - 1s - 7ms/step - accuracy: 0.5954 - loss: 1.0415 - val_accuracy: 0.5747 - val_loss: 1.0864
Epoch 9/10
81/81 - 1s - 7ms/step - accuracy: 0.6058 - loss: 1.0228 - val_accuracy: 0.5782 - val_loss: 1.0707
Epoch 10/10
81/81 - 1s - 9ms/step - accuracy: 0.6131 - loss: 1.0046 - val_accuracy: 0.5759 - val_loss: 1.0679
Validation Accuracy for Structure [108]: 0.5759

Training MLP Structure: [216]
Epoch 1/10
81/81 - 2s - 24ms/step - accuracy: 0.3761 - loss: 1.4971 - val_accuracy: 0.4496 - val_loss: 1.3679
Epoch 2/10
81/81 - 0s - 4ms/step - accuracy: 0.5035 - loss: 1.2756 - val_accuracy: 0.5342 - val_loss: 1.2001
Epoch 3/10
81/81 - 0s - 4ms/step - accuracy: 0.5413 - loss: 1.1707 - val_accuracy: 0.5516 - val_loss: 1.1432
Epoch 4/10
81/81 - 0s - 5ms/step - accuracy: 0.5625 - loss: 1.1143 - val_accuracy: 0.5678 - val_loss: 1.1105
Epoch 5/10
81/81 - 0s - 4ms/step - accuracy: 0.5780 - loss: 1.0758 - val_accuracy: 0.5724 - val_loss: 1.0940
Epoch 6/10
81/81 - 0s - 5ms/step - accuracy: 0.5977 - loss: 1.0465 - val_accuracy: 0.5771 - val_loss: 1.0831
Epoch 7/10
81/81 - 1s - 8ms/step - accuracy: 0.6066 - loss: 1.0202 - val_accuracy: 0.5771 - val_loss: 1.0727
Epoch 8/10
81/81 - 1s - 7ms/step - accuracy: 0.6185 - loss: 0.9970 - val_accuracy: 0.5794 - val_loss: 1.0617
Epoch 9/10
81/81 - 1s - 10ms/step - accuracy: 0.6274 - loss: 0.9744 - val_accuracy: 0.5829 - val_loss: 1.0572
Epoch 10/10
81/81 - 1s - 8ms/step - accuracy: 0.6297 - loss: 0.9566 - val_accuracy: 0.5979 - val_loss: 1.0485
Validation Accuracy for Structure [216]: 0.5979

Training MLP Structure: [149, 108]
Epoch 1/10
```

```
81/81 - 3s - 34ms/step - accuracy: 0.3710 - loss: 1.4674 - val_accuracy: 0.4600 - val_loss: 1.2669
Epoch 2/10
81/81 - 1s - 10ms/step - accuracy: 0.5166 - loss: 1.1856 - val_accuracy: 0.5492 - val_loss: 1.1310
Epoch 3/10
81/81 - 1s - 9ms/step - accuracy: 0.5695 - loss: 1.0855 - val_accuracy: 0.5782 - val_loss: 1.0777
Epoch 4/10
81/81 - 1s - 7ms/step - accuracy: 0.5988 - loss: 1.0234 - val_accuracy: 0.5805 - val_loss: 1.0532
Epoch 5/10
81/81 - 1s - 7ms/step - accuracy: 0.6174 - loss: 0.9831 - val_accuracy: 0.5782 - val_loss: 1.0561
Epoch 6/10
81/81 - 0s - 6ms/step - accuracy: 0.6340 - loss: 0.9409 - val_accuracy: 0.5643 - val_loss: 1.0606
Epoch 7/10
81/81 - 0s - 5ms/step - accuracy: 0.6398 - loss: 0.9121 - val_accuracy: 0.5979 - val_loss: 1.0317
Epoch 8/10
81/81 - 1s - 7ms/step - accuracy: 0.6595 - loss: 0.8743 - val_accuracy: 0.5886 - val_loss: 1.0316
Epoch 9/10
81/81 - 0s - 5ms/step - accuracy: 0.6691 - loss: 0.8511 - val_accuracy: 0.6072 - val_loss: 1.0118
Epoch 10/10
81/81 - 1s - 7ms/step - accuracy: 0.6846 - loss: 0.8167 - val_accuracy: 0.6130 - val_loss: 1.0116
Validation Accuracy for Structure [149, 108]: 0.6130

Training MLP Structure: [216, 5]
Epoch 1/10
81/81 - 2s - 24ms/step - accuracy: 0.2614 - loss: 1.5374 - val_accuracy: 0.3569 - val_loss: 1.4496
Epoch 2/10
81/81 - 1s - 13ms/step - accuracy: 0.4147 - loss: 1.3883 - val_accuracy: 0.4519 - val_loss: 1.3460
Epoch 3/10
81/81 - 0s - 6ms/step - accuracy: 0.4795 - loss: 1.2969 - val_accuracy: 0.4832 - val_loss: 1.2840
Epoch 4/10
81/81 - 1s - 10ms/step - accuracy: 0.5189 - loss: 1.2326 - val_accuracy: 0.5145 - val_loss: 1.2313
Epoch 5/10
81/81 - 1s - 6ms/step - accuracy: 0.5293 - loss: 1.1780 - val_accuracy: 0.5261 - val_loss: 1.1971
Epoch 6/10
81/81 - 1s - 7ms/step - accuracy: 0.5533 - loss: 1.1292 - val_accuracy: 0.5469 - val_loss: 1.1577
Epoch 7/10
81/81 - 1s - 9ms/step - accuracy: 0.5737 - loss: 1.0866 - val_accuracy: 0.5504 - val_loss: 1.1297
Epoch 8/10
81/81 - 1s - 8ms/step - accuracy: 0.5853 - loss: 1.0505 - val_accuracy: 0.5574 - val_loss: 1.1097
Epoch 9/10
81/81 - 1s - 7ms/step - accuracy: 0.5992 - loss: 1.0204 - val_accuracy: 0.5585 - val_loss: 1.0983
Epoch 10/10
81/81 - 1s - 8ms/step - accuracy: 0.6093 - loss: 0.9911 - val_accuracy: 0.5701 - val_loss: 1.0767
Validation Accuracy for Structure [216, 5]: 0.5701

Training MLP Structure: [108, 216]
Epoch 1/10
81/81 - 2s - 30ms/step - accuracy: 0.3668 - loss: 1.4697 - val_accuracy: 0.4751 - val_loss: 1.2466
Epoch 2/10
81/81 - 0s - 6ms/step - accuracy: 0.5124 - loss: 1.1820 - val_accuracy: 0.5365 - val_loss: 1.1148
Epoch 3/10
81/81 - 1s - 7ms/step - accuracy: 0.5533 - loss: 1.0881 - val_accuracy: 0.5678 - val_loss: 1.0853
Epoch 4/10
81/81 - 1s - 10ms/step - accuracy: 0.5942 - loss: 1.0279 - val_accuracy: 0.5689 - val_loss: 1.0659
Epoch 5/10
81/81 - 0s - 6ms/step - accuracy: 0.6093 - loss: 0.9811 - val_accuracy: 0.5782 - val_loss: 1.0636
Epoch 6/10
81/81 - 1s - 7ms/step - accuracy: 0.6336 - loss: 0.9366 - val_accuracy: 0.5689 - val_loss: 1.0697
Epoch 7/10
81/81 - 1s - 7ms/step - accuracy: 0.6506 - loss: 0.9025 - val_accuracy: 0.5991 - val_loss: 1.0195
Epoch 8/10
81/81 - 0s - 5ms/step - accuracy: 0.6633 - loss: 0.8720 - val_accuracy: 0.5852 - val_loss: 1.0267
Epoch 9/10
81/81 - 1s - 7ms/step - accuracy: 0.6784 - loss: 0.8326 - val_accuracy: 0.5991 - val_loss: 1.0261
Epoch 10/10
81/81 - 1s - 7ms/step - accuracy: 0.6973 - loss: 0.8026 - val_accuracy: 0.6176 - val_loss: 1.0298
Validation Accuracy for Structure [108, 216]: 0.6176

Training MLP Structure: [149, 108, 5]
Epoch 1/10
81/81 - 2s - 27ms/step - accuracy: 0.3278 - loss: 1.5263 - val_accuracy: 0.4264 - val_loss: 1.4144
Epoch 2/10
81/81 - 0s - 5ms/step - accuracy: 0.4591 - loss: 1.3031 - val_accuracy: 0.4670 - val_loss: 1.2538
Epoch 3/10
81/81 - 0s - 5ms/step - accuracy: 0.4985 - loss: 1.1764 - val_accuracy: 0.4878 - val_loss: 1.1736
Epoch 4/10
81/81 - 1s - 10ms/step - accuracy: 0.5274 - loss: 1.1003 - val_accuracy: 0.5272 - val_loss: 1.1273
Epoch 5/10
81/81 - 1s - 8ms/step - accuracy: 0.5622 - loss: 1.0472 - val_accuracy: 0.5597 - val_loss: 1.0879
Epoch 6/10
81/81 - 1s - 7ms/step - accuracy: 0.6019 - loss: 0.9972 - val_accuracy: 0.5886 - val_loss: 1.0495
Epoch 7/10
81/81 - 1s - 8ms/step - accuracy: 0.6274 - loss: 0.9474 - val_accuracy: 0.6049 - val_loss: 1.0413
Epoch 8/10
81/81 - 1s - 7ms/step - accuracy: 0.6386 - loss: 0.9109 - val_accuracy: 0.5991 - val_loss: 1.0281
Epoch 9/10
```

```
81/81 - 1s - 8ms/step - accuracy: 0.6583 - loss: 0.8828 - val_accuracy: 0.6072 - val_loss: 1.0309
Epoch 10/10
81/81 - 1s - 7ms/step - accuracy: 0.6780 - loss: 0.8425 - val_accuracy: 0.6141 - val_loss: 1.0069
Validation Accuracy for Structure [149, 108, 5]: 0.6141

Training MLP Structure: [216, 108, 5]
Epoch 1/10
81/81 - 2s - 29ms/step - accuracy: 0.3846 - loss: 1.4948 - val_accuracy: 0.4670 - val_loss: 1.3224
Epoch 2/10
81/81 - 0s - 5ms/step - accuracy: 0.4927 - loss: 1.2392 - val_accuracy: 0.4948 - val_loss: 1.1940
Epoch 3/10
81/81 - 0s - 5ms/step - accuracy: 0.5201 - loss: 1.1358 - val_accuracy: 0.5191 - val_loss: 1.1348
Epoch 4/10
81/81 - 0s - 5ms/step - accuracy: 0.5456 - loss: 1.0685 - val_accuracy: 0.5284 - val_loss: 1.1054
Epoch 5/10
81/81 - 0s - 6ms/step - accuracy: 0.5757 - loss: 1.0113 - val_accuracy: 0.5388 - val_loss: 1.0867
Epoch 6/10
81/81 - 1s - 7ms/step - accuracy: 0.6031 - loss: 0.9572 - val_accuracy: 0.5620 - val_loss: 1.0614
Epoch 7/10
81/81 - 0s - 5ms/step - accuracy: 0.6305 - loss: 0.9100 - val_accuracy: 0.5829 - val_loss: 1.0330
Epoch 8/10
81/81 - 0s - 6ms/step - accuracy: 0.6521 - loss: 0.8700 - val_accuracy: 0.5898 - val_loss: 1.0210
Epoch 9/10
81/81 - 0s - 4ms/step - accuracy: 0.6718 - loss: 0.8307 - val_accuracy: 0.5875 - val_loss: 1.0397
Epoch 10/10
81/81 - 1s - 8ms/step - accuracy: 0.6880 - loss: 0.7983 - val_accuracy: 0.6188 - val_loss: 1.0135
Validation Accuracy for Structure [216, 108, 5]: 0.6188

Training MLP Structure: [216, 108, 5]
Epoch 1/10
81/81 - 2s - 26ms/step - accuracy: 0.2961 - loss: 1.5197 - val_accuracy: 0.4148 - val_loss: 1.4279
Epoch 2/10
81/81 - 0s - 6ms/step - accuracy: 0.4452 - loss: 1.3696 - val_accuracy: 0.4461 - val_loss: 1.3300
Epoch 3/10
81/81 - 1s - 7ms/step - accuracy: 0.4795 - loss: 1.2585 - val_accuracy: 0.4797 - val_loss: 1.2345
Epoch 4/10
81/81 - 1s - 9ms/step - accuracy: 0.5274 - loss: 1.1604 - val_accuracy: 0.5214 - val_loss: 1.1718
Epoch 5/10
81/81 - 1s - 8ms/step - accuracy: 0.5591 - loss: 1.0945 - val_accuracy: 0.5284 - val_loss: 1.1463
Epoch 6/10
81/81 - 1s - 8ms/step - accuracy: 0.5795 - loss: 1.0323 - val_accuracy: 0.5342 - val_loss: 1.1377
Epoch 7/10
81/81 - 1s - 15ms/step - accuracy: 0.5911 - loss: 0.9892 - val_accuracy: 0.5435 - val_loss: 1.1046
Epoch 8/10
81/81 - 1s - 8ms/step - accuracy: 0.6127 - loss: 0.9468 - val_accuracy: 0.5527 - val_loss: 1.1044
Epoch 9/10
81/81 - 1s - 15ms/step - accuracy: 0.6236 - loss: 0.9089 - val_accuracy: 0.5632 - val_loss: 1.0867
Epoch 10/10
81/81 - 1s - 7ms/step - accuracy: 0.6398 - loss: 0.8779 - val_accuracy: 0.5504 - val_loss: 1.1252
Validation Accuracy for Structure [216, 108, 5]: 0.5504

Best MLP Structure: [216, 108, 5]
Best Validation Accuracy: 0.6188

=== Question 2.3: Evaluate Best Model ===
27/27 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step

Test Accuracy: 0.5694
Precision: 0.5843
Recall: 0.5694
F1 Score: 0.5607
```






```
=== Question 2.4: Timing ===
27/27 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
Average Inference Time: 0.0002 seconds

=== Question 2.5: Visualize Predictions ===
Correctly classified images:
```


```
Incorrectly classified images:
```

Task 3: Convolutional Neural Network

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

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

```
import os
import numpy as np
import matplotlib.pyplot as plt
import cv2
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matr
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from tensorflow.keras.utils import to_categorical
import time
from tqdm import tqdm

# Labels for flower classes
LABELS = ["daisy", "dandelion", "rose", "sunflower", "tulip"]

# ====================== Question 3.1: Design and Architecture ======================
def build_cnn(input_shape, num_classes):
    model = Sequential([
        Conv2D(32, (3, 3), activation='relu', input_shape=input_shape),
        MaxPooling2D((2, 2)),
        Conv2D(64, (3, 3), activation='relu'),
        MaxPooling2D((2, 2)),
        Flatten(),
        Dense(128, activation='relu'),
        Dropout(0.5),
        Dense(num_classes, activation='softmax')
    ])
    model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
    return model

# ====================== Preprocessing ======================
def preprocess_images(base_path, labels, img_size=150):
    X = []
    y = []
    for label in labels:
        folder_path = os.path.join(base_path, label)
        for img_name in tqdm(os.listdir(folder_path), desc=f"Processing {label}"):
            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, (img_size, img_size))
            X.append(image_resized)
            y.append(label)
    return np.array(X), np.array(y)

# ====================== Main Code ======================
if __name__ == "__main__":
    BASE_PATH = "/content/drive/My Drive/Colab Notebooks/testing2/flowers"
```

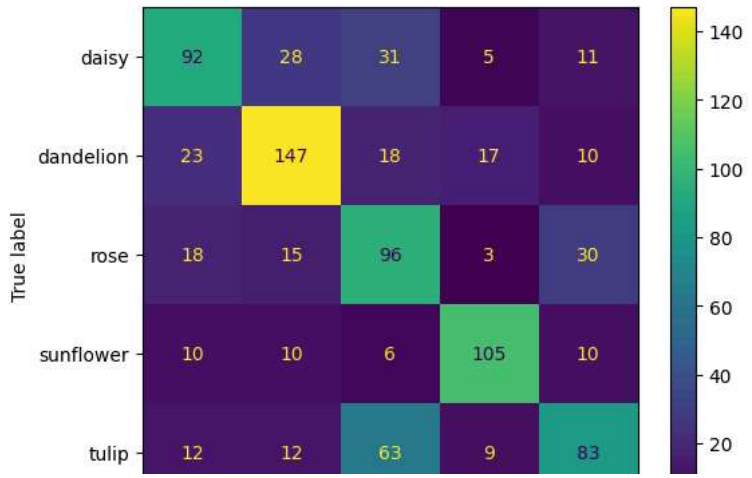
```
IMG_SIZE = 150

print("\n=== Question 3.1: Preprocessing and Building CNN Model ===")
# Load and preprocess data
X, y = preprocess_images(BASE_PATH, LABELS, IMG_SIZE)
print(f"Dataset loaded: {len(X)} samples")

# Encode labels and split dataset
label_map = {label: idx for idx, label in enumerate(LABELS)}
y_encoded = np.array([label_map[label] for label in y])
y_one_hot = to_categorical(y_encoded, num_classes=len(LABELS))
X_train, X_temp, y_train, y_temp = train_test_split(X, y_one_hot, test_size=0.4, random_state=4
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state=42)

# Normalize image data
X_train = X_train / 255.0
X_val = X_val / 255.0
X_test = X_test / 255.0

# Build CNN model
input_shape = (IMG_SIZE, IMG_SIZE, 3)
cnn = build_cnn(input_shape, len(LABELS))
cnn.summary()

# ====================== Question 3.2: Training and Classification Metrics ==================
print("\n=== Question 3.2: Training CNN Model ===")
start_time = time.time()
history = cnn.fit(X_train, y_train, validation_data=(X_val, y_val), epochs=10, batch_size=32, v
training_time = time.time() - start_time

print("\n=== Question 3.2: Evaluate Model on Test Set ===")
start_inference = time.time()
test_loss, test_acc = cnn.evaluate(X_test, y_test, verbose=0)
inference_time = (time.time() - start_inference) / len(X_test)
y_test_pred = np.argmax(cnn.predict(X_test), axis=1)
y_test_actual = np.argmax(y_test, axis=1)

acc = accuracy_score(y_test_actual, y_test_pred)
precision = precision_score(y_test_actual, y_test_pred, average='weighted')
recall = recall_score(y_test_actual, y_test_pred, average='weighted')
f1 = f1_score(y_test_actual, y_test_pred, average='weighted')
cm = confusion_matrix(y_test_actual, y_test_pred)

print(f"\nTest Accuracy: {acc:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F1 Score: {f1:.4f}")

# Confusion matrix
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=LABELS)
disp.plot()
plt.show()

# Identify the most confused pair of classes
most_confused = np.unravel_index(np.argmax(cm - np.diag(np.diag(cm))), cm.shape)
print(f"\nMost confused classes: {LABELS[most_confused[0]]} and {LABELS[most_confused[1]]}")

# ====================== Question 3.3: Training and Validation Plots ======================
print("\n=== Question 3.3: Training and Validation Loss/Accuracy ===")
```

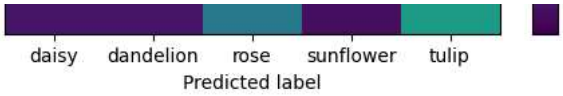
```
    print("\n=== Question 3.3: Training and Validation Loss/Accuracy ===")
    plt.figure(figsize=(12, 5))
    plt.subplot(1, 2, 1)
    plt.plot(history.history['loss'], label='Training Loss')
    plt.plot(history.history['val_loss'], label='Validation Loss')
    plt.title('Loss vs. Epochs')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.legend()

    plt.subplot(1, 2, 2)
    plt.plot(history.history['accuracy'], label='Training Accuracy')
    plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
    plt.title('Accuracy vs. Epochs')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.legend()

    plt.tight_layout()
    plt.show()

    # ====================== Question 3.4: Training and Inference Time ======================
    print("\n=== Question 3.4: Training and Inference Time ===")
    print(f"Training Time: {training_time:.4f} seconds")
    print(f"Average Inference Time per Sample: {inference_time:.6f} seconds")

    # ====================== Question 3.5: Visualization of Predictions ======================
    print("\n=== Question 3.5: Visualizing Predictions ===")
    def visualize_predictions(X, y_true, y_pred, labels, correct=True, num_samples=5):
        indices = np.where(y_true == y_pred)[0] if correct else np.where(y_true != 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):
            # Convert to uint8 and then from BGR to RGB
            img = (X[idx] * 255).astype(np.uint8)  # Scale back to 0-255 and convert to uint8
            img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
            plt.subplot(1, num_samples, i + 1)
            plt.imshow(img_rgb)
            plt.title(f"True: {labels[y_true[idx]]}\nPred: {labels[y_pred[idx]]}", color="green" if
            plt.axis('off')
        plt.tight_layout()
        plt.show()



    print("Correctly classified images:")
    visualize_predictions(X_test, y_test_actual, y_test_pred, LABELS, correct=True, num_samples=5)

    print("Incorrectly classified images:")
    visualize_predictions(X_test, y_test_actual, y_test_pred, LABELS, correct=False, num_samples=5)
```

```
=== Question 3.1: Preprocessing and Building CNN Model ===
Processing daisy: 100%|██████████| 764/764 [00:14<00:00, 54.25it/s]
Processing dandelion: 100%|██████████| 1052/1052 [00:21<00:00, 48.22it/s]
Processing rose: 100%|██████████| 784/784 [00:12<00:00, 64.00it/s]
Processing sunflower: 100%|██████████| 733/733 [00:12<00:00, 59.26it/s]
Processing tulip: 100%|██████████| 984/984 [00:19<00:00, 51.66it/s]
Dataset loaded: 4317 samples
/usr/local/lib/python3.11/dist-packages/keras/src/layers/convolutional/base_conv.py:107:
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Model: "sequential_1"
```

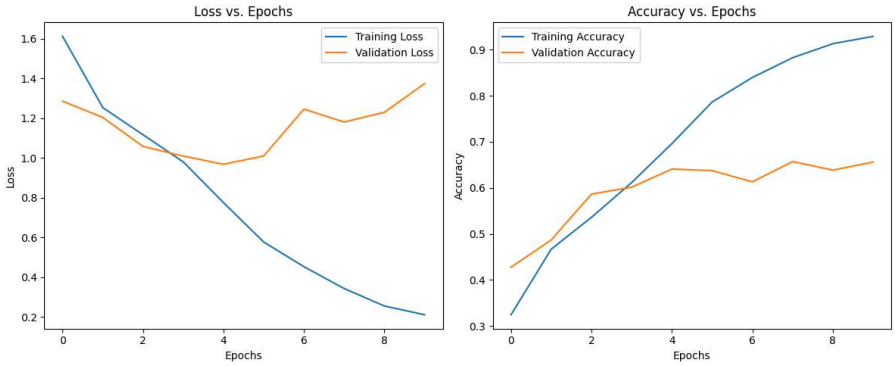
Layer (type)	Output Shape	Param #
conv2d_2 (Conv2D)	(None, 148, 148, 32)	896
max_pooling2d_2 (MaxPooling2D)	(None, 74, 74, 32)	0
conv2d_3 (Conv2D)	(None, 72, 72, 64)	18,496
max_pooling2d_3 (MaxPooling2D)	(None, 36, 36, 64)	0
flatten_1 (Flatten)	(None, 82944)	0
dense_2 (Dense)	(None, 128)	10,616,960
dropout_1 (Dropout)	(None, 128)	0
dense_3 (Dense)	(None, 5)	645

```
Total params: 10,636,997 (40.58 MB)
Trainable params: 10,636,997 (40.58 MB)
Non-trainable params: 0 (0.00 B)

=== Question 3.2: Training CNN Model ===
Epoch 1/10
81/81 - 123s - 2s/step - accuracy: 0.3243 - loss: 1.6115 - val_accuracy: 0.4276 - val_lo
Epoch 2/10
81/81 - 135s - 2s/step - accuracy: 0.4668 - loss: 1.2533 - val_accuracy: 0.4867 - val_lo
Epoch 3/10
81/81 - 125s - 2s/step - accuracy: 0.5359 - loss: 1.1167 - val_accuracy: 0.5863 - val_lo
Epoch 4/10
81/81 - 137s - 2s/step - accuracy: 0.6116 - loss: 0.9802 - val_accuracy: 0.6014 - val_lo
Epoch 5/10
81/81 - 154s - 2s/step - accuracy: 0.6961 - loss: 0.7752 - val_accuracy: 0.6408 - val_lo
Epoch 6/10
81/81 - 137s - 2s/step - accuracy: 0.7865 - loss: 0.5768 - val_accuracy: 0.6373 - val_lo
Epoch 7/10
81/81 - 140s - 2s/step - accuracy: 0.8398 - loss: 0.4528 - val_accuracy: 0.6130 - val_lo
Epoch 8/10
81/81 - 134s - 2s/step - accuracy: 0.8826 - loss: 0.3429 - val_accuracy: 0.6570 - val_lo
Epoch 9/10
81/81 - 151s - 2s/step - accuracy: 0.9131 - loss: 0.2550 - val_accuracy: 0.6385 - val_lo
Epoch 10/10
81/81 - 133s - 2s/step - accuracy: 0.9290 - loss: 0.2111 - val_accuracy: 0.6559 - val_lo

=== Question 3.2: Evaluate Model on Test Set ===
27/27 ━━━━━━━━━━━━━━━━━━━━ 11s 403ms/step

Test Accuracy: 0.6053
Precision: 0.6141
Recall: 0.6053
F1 Score: 0.6064
```

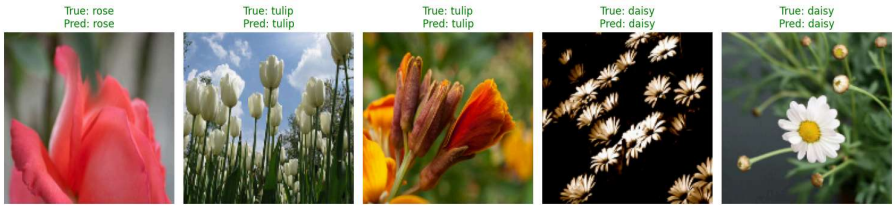





```
Most confused classes: tulip and rose

=== Question 3.3: Training and Validation Loss/Accuracy ===
```




```
=== Question 3.4: Training and Inference Time ===
Training Time: 1370.6880 seconds
Average Inference Time per Sample: 0.013097 seconds

=== Question 3.5: Visualizing Predictions ===
Correctly classified images:
```




```
Incorrectly classified images:
```