Task 3: Convolutional Neural Network

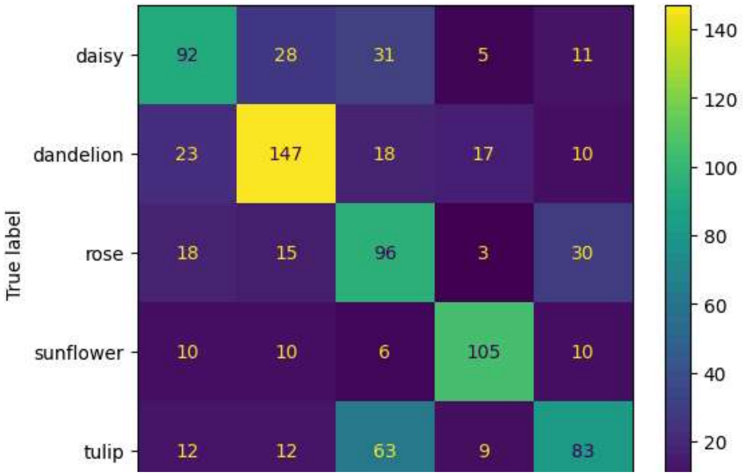
```
from google.colab import drive
drive.mount('/content/drive')
```

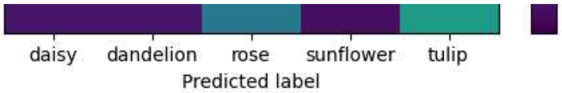
```
Mounted at /content/drive
```

```
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"
```

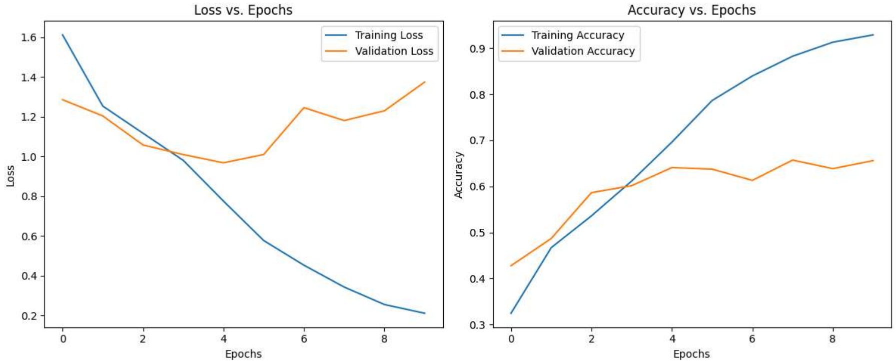
```
IMG_SIZE = 150

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
```

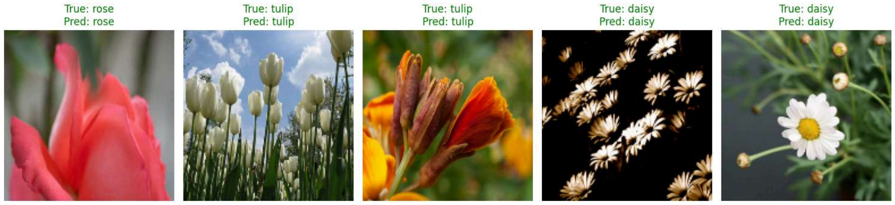




```
Most confused classes: tulip and rose

=== 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:
```