Flower Classification Project Documentation

1. Overview

This project develops a Convolutional Neural Network (CNN) to classify images of flowers into five categories: daisy, dandelion, rose, sunflower, and tulip. The implementation uses TensorFlow for building and training the CNN, NumPy for numerical computations, and Streamlit for potential web-based deployment. The dataset comprises 4,317 images across the five flower categories, stored in a directory named flowers/.

The project is implemented in a Jupyter Notebook (FinalProject.ipynb) and includes data preprocessing, model training, evaluation, and a prediction function for classifying new images. This document provides a comprehensive explanation of the project's components, workflow, and code.

---

2. Project Objectives

- Build a CNN model to classify flower images with high accuracy.
- Preprocess the dataset to ensure compatibility with the CNN model.
- Visualize data and model performance using Matplotlib and Seaborn.
- Implement a prediction function to classify new images.
- Prepare for web deployment using Streamlit (though not fully implemented in the provided code).

---

3. Dataset

The dataset is stored in the flowers/ directory, with subdirectories for each flower class:

- **Daisy**: 764 images
- **Dandelion**: 1,052 images
- **Rose**: 784 images
- **Sunflower**: 733 images
- **Tulip**: 984 images
- **Total**: 4,317 images

The dataset is split as follows:

- **Training set:** 80% (3,454 images)
- **Validation set**: 20% (863 images)

---

4. Project Workflow

The project follows these steps:

1. **Install Dependencies**: Install required libraries (TensorFlow, NumPy, Streamlit).
2. **Import Libraries**: Import modules for data processing, model building, and visualization.
3. **Data Exploration**: Analyze the dataset to understand the number of images per class.
4. **Data Preprocessing**: Load and preprocess images for training and validation.
5. **Model Building**: Define and compile the CNN model.
6. **Model Training**: Train the model with early stopping and model checkpointing.
7. **Model Evaluation**: Evaluate the model using metrics like accuracy and confusion matrix.
8. **Prediction**: Implement a function to classify new images.
9. **Visualization**: Display predictions and model performance metrics.

---

5. Code Explanation

5.1 Install Dependencies

The first cell installs the required libraries:

bash

```
!pip install tensorflow numpy streamlit
```

- **TensorFlow**: For building and training the CNN model.
- **NumPy**: For numerical operations and array manipulations.
- **Streamlit**: For creating a web interface (not implemented in the provided code).

5.2 Import Libraries

The second cell imports necessary Python libraries:

```
import os

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

import tensorflow as tf

from tensorflow.keras import layers

from tensorflow.keras.preprocessing import image

from tensorflow.keras.preprocessing.image import load_img,
ImageDataGenerator

from tensorflow.keras.models import Sequential, load_model

from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dense,
Dropout, Flatten

from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint

from sklearn.metrics import confusion_matrix, classification_report
```

- **os**: For file system operations.
- **NumPy**: For numerical computations.
- **Matplotlib/Seaborn**: For plotting images and performance metrics.
- **TensorFlow/Keras**: For building and training the CNN.
- **ImageDataGenerator**: For data augmentation and image loading.
- **Sequential/load_model**: For creating and loading CNN models.
- **Conv2D/MaxPooling2D/Flatten/Dense/Dropout**: CNN architecture layers.
- **EarlyStopping/ModelCheckpoint**: Callbacks to optimize training.
- **confusion_matrix/classification_report**: For model evaluation.

5.3 Data Exploration

The third cell counts the number of images in each class:

```
count = 0
dirs = os.listdir('flowers/')
for dir in dirs:
    files = list(os.listdir('flowers/' + dir))
    print(dir + ' Folder has ' + str(len(files)) + ' Images')
    count = count + len(files)
print('Images Folder has ' + str(count) + ' Images')
```

Output:

```
Text daisy Folder has 764 Images
dandelion Folder has 1052 Images
rose Folder has 784 Images
sunflower Folder has 733 Images
tulip Folder has 984 Images
Images Folder has 4317 Images
```

This confirms the dataset’s structure and accessibility.

5.4 Data Preprocessing

The fourth cell loads the dataset using image_dataset_from_directory:

```
base_dir = 'flowers/'
img_size = 128
batch = 64

train_ds = tf.keras.utils.image_dataset_from_directory(
    base_dir,
    validation_split=0.2,
    subset="training",
    seed=123,
    image_size=(img_size, img_size),
    batch_size=batch
)
val_ds = tf.keras.utils.image_dataset_from_directory(
    base_dir,
    validation_split=0.2,
    subset="validation",
    seed=123,
    image_size=(img_size, img_size),
    batch_size=batch
)

flower_names = train_ds.class_names
print(flower_names)
```

- **base_dir**: Path to the dataset directory.
- **img_size**: Resizes images to 128x128 pixels.

- **batch**: Sets batch size to 64.
- **validation_split**: Allocates 20% for validation.
- **seed**: Ensures reproducibility.
- **image_size**: Specifies target image size.
- **batch_size**: Number of images per batch.
- **flower_names**: Extracts class names (['daisy', 'dandelion', 'rose', 'sunflower', 'tulip']).

Output:

```
Found 4317 files belonging to 5 classes.

Using 3454 files for training.

Found 4317 files belonging to 5 classes.

Using 863 files for validation.

['daisy', 'dandelion', 'rose', 'sunflower', 'tulip']
```

5.5 Prediction Function

The fifth cell defines functions to load, preprocess, and predict the class of a single image:

```
def load_and_preprocess_image(img_path, target_size=(128, 128)):
    img = image.load_img(img_path, target_size=target_size)
    img_array = image.img_to_array(img)
    img_array = tf.expand_dims(img_array, axis=0)  # batch dimension
    img_array = img_array / 255.0
    return img_array
def predict_image(model, img_path, class_names):
    img_array = load_and_preprocess_image(img_path)
    prediction = model.predict(img_array)
    predicted_class = class_names[np.argmax(prediction)]
    confidence = np.max(prediction)
    plt.imshow(image.load_img(img_path))
    plt.title(f"Prediction: {predicted_class} ({confidence:.2f})")
    plt.axis("off")
    plt.show()
    return predicted_class, confidence
predict_image(cnn_model_1, 'flowers/dandelion/148180650_19a4b410db.jpg', ['daisy', 'dandelion', 'rose', 'sunflower', 'tulip'])
```

•
•
•
•
•
•

- **load_and_preprocess_image**:
  - Loads an image using load_img.
  - Converts it to a NumPy array with img_to_array.
  - Adds a batch dimension using expand_dims.
  - Normalizes pixel values to [0, 1] by dividing by 255.
- **predict_image**:
  - Preprocesses the image.
  - Uses the trained model (cnn_model_1) to predict the class.
  - Extracts the predicted class and confidence score.
  - Displays the image with the prediction and confidence using Matplotlib.

```
('dandelion', 0.4543447)
```

The image is displayed with the title "Prediction: dandelion (0.45)".

Note: The model (cnn_model_1) is not defined in the provided code, indicating missing model training code.

---

6. Missing Components

The provided notebook excerpt lacks:

- **Model Definition and Training**: The CNN model (cnn_model_1) is referenced but not defined. This would typically include:
  - Creating a Sequential model.
  - Adding Conv2D, MaxPooling2D, Flatten, Dense, and Dropout layers.
  - Compiling with an optimizer (e.g., Adam), loss function (e.g., categorical crossentropy), and metrics (e.g., accuracy).
  - Training using model.fit with train_ds and val_ds, possibly with EarlyStopping and ModelCheckpoint.
- **Model Evaluation**: Code for evaluating the model on the validation set, likely using confusion_matrix and classification_report.
- **Streamlit Interface**: Code for deploying the model as a web application.

Based on the imported libraries and comments, the model likely uses a CNN with data augmentation (ImageDataGenerator) and callbacks like EarlyStopping and ModelCheckpoint.

---

7. Assumptions About the Model

The CNN model (cnn_model_1) is likely structured as follows:

- **Input Layer**: Accepts 128x128x3 images (RGB).
- **Convolutional Layers**: Multiple Conv2D layers with ReLU activation.
- **Pooling Layers**: MaxPooling2D layers to reduce spatial dimensions.
- **Flatten Layer**: Converts 2D feature maps to a 1D vector.
- **Dense Layers**: Fully connected layers, with a final layer of 5 units (one per class) and softmax activation.
- **Dropout**: To prevent overfitting.
- **Optimizer**: Adam or similar.
- **Loss Function**: Categorical crossentropy.
- **Metrics**: Accuracy.

The model is trained on train_ds and validated on val_ds, with data augmentation for robustness.

---

8. Results

The provided code demonstrates the prediction functionality:

- **Example Prediction**: The image flowers/dandelion/148180650_19a4b410db.jpg is classified as "dandelion" with a confidence of 0.45.
- **Visualization**: The image is displayed with the predicted class and confidence score.

The confidence score (0.45) suggests the model may require further tuning, additional training, or improved data augmentation.

---

9. Potential Improvements

1. **Model Architecture**:
    - Experiment with deeper architectures or pre-trained models (e.g., VGG16, ResNet) for transfer learning.
    - Adjust filters, kernel sizes, or layers.
2. **Data Augmentation**:
    - Apply transformations like rotation, flipping, zooming, or brightness adjustments using ImageDataGenerator.
3. **Hyperparameter Tuning**:
    - Tune learning rate, batch size, or epochs.
    - Adjust dropout rates or regularization.
4. **Evaluation**:
    - Generate a confusion matrix and classification report.
    - Compute precision, recall, and F1-score.
5. **Streamlit Interface**:
    - Implement a web interface for image uploads and predictions.
6. **Dataset Expansion**:
    - Increase dataset size or balance classes (e.g., sunflower has fewer images).

---

10. Conclusion

This project provides a foundational approach to image classification using a CNN. It includes data preprocessing, a prediction function, and visualization, with potential for web deployment via Streamlit. Although the provided code lacks model definition and training, the structure supports a robust workflow. Future work could enhance model accuracy, expand the dataset, and implement the Streamlit interface.

---

11. Requirements

To run the project, install the following libraries:

pip install tensorflow numpy streamlit matplotlib seaborn scikit-learn

The dataset should be organized in the flowers/ directory with subdirectories for each class.
