FLOWER RECOGNITION USING CONVOLUTIONAL NEURAL NETWORK

Github Link: https://github.com/rajeshk015/Flower_Recognition_Using_Convolutional_Neural_Network

PRESENTED BY:

RAJESH KUMAR

MD ADNAN

Team Members

Rajesh Kumar
@rajeshk015

Md Adnan
@adnansayeed0852

OBJECTIVE

- To build a Machine Learning model that can recognize Flowers using Convolutional Neural Network (CNNs)

INTRODUCTION

- Welcome to our project on Flower Recognition Using Convolutional Neural Networks (CNNs). In this endeavor, our primary objective is to develop a highly accurate and efficient model capable of recognizing various flower species through the power of deep learning.

Importing necessary libraries

```
import numpy as np
import tensorflow as tf
from keras.preprocessing.image import ImageDataGenerator
```

- **NumPy (np):**

  NumPy is a Python library for numerical operations, essential for efficient data manipulation and array operations in the project.

- **TensorFlow (tf):**

  TensorFlow is a powerful machine learning library used for constructing and training neural networks, serving as the core framework in this project.

- **Keras (ImageDataGenerator):**

  Keras, running on top of TensorFlow, provides the ImageDataGenerator module for efficient data augmentation, enhancing dataset diversity during neural network training.

Loading Training and Test Image Sets

```
train_datagen = ImageDataGenerator(
        rescale=1./255,
        shear_range=0.2,
        zoom_range=0.2,
        horizontal_flip=True)
training_set = train_datagen.flow_from_directory(
        'training_set',
        target_size=(64, 64),
        batch_size=32,
        class_mode='categorical')
```

```
test_datagen = ImageDataGenerator(rescale=1./255)
test_set = test_datagen.flow_from_directory(
        'test_set',
        target_size=(64, 64),
        batch_size=32,
        class_mode='categorical')
```

- **Data Augmentation Configuration:**

  The ImageDataGenerator from Keras is configured to augment the training dataset. Augmentation includes rescaling pixel values, shearing, zooming, and horizontal flipping, introducing variety for robust model training.

- **Loading Training Images:**

  The flow_from_directory function is utilized to load and augment images from the 'training_set' directory. Images are resized to (64, 64) pixels, organized into batches of 32, and categorical class labels are employed for multi-class classification.

- **Loading Test Images:**

  The flow_from_directory function loads and preprocesses images from the 'test_set' directory. Similar to the training set, images are resized to (64, 64) pixels, organized into batches of 32, and categorical class labels are employed for multi-class classification evaluation.

Building Model

```
cnn = tf.keras.models.Sequential()
```

- **Convolutional Neural Network Architecture:**

  The project uses a Sequential model from TensorFlow's Keras API to construct the Convolutional Neural Network (CNN). The Sequential model allows for the sequential layer-wise addition of neural network components, facilitating the design and implementation of the CNN for flower recognition.

Building Convolution Layer

```
cnn.add(tf.keras.layers.Conv2D(filters=64 , kernel_size=3 , activation='relu' , input_shape=[64,64,3]))
cnn.add(tf.keras.layers.MaxPool2D(pool_size=2,strides=2))
```

- **Convolutional and Pooling Layers:**

  The Convolutional Neural Network (CNN) architecture begins with a Conv2D layer, consisting of 64 filters with a kernel size of 3x3, utilizing the ReLU activation function. This is followed by a MaxPooling2D layer with a pool size of 2x2 and a stride of 2, facilitating spatial down-sampling and feature extraction. These layers contribute to the network's ability to learn hierarchical representations of input images.

Improving Accuracy

```
cnn.add(tf.keras.layers.Dropout(0.5))
```

- **Dropout Layer for Regularization:**

  A Dropout layer is incorporated with a dropout rate of 0.5. This layer introduces a regularization technique during training by randomly deactivating 50% of the neurons. Dropout helps prevent overfitting, enhancing the model's ability to generalize to unseen data.

Reshaping the Matrix

```
cnn.add(tf.keras.layers.Flatten())
```

- **Flatten Layer for Transition:**

  The Flatten layer is introduced to transition from the convolutional and pooling layers to the fully connected layers. It reshapes the 2D feature maps into a 1D vector, preparing the data for input into the subsequent dense layers of the neural network.

Making a Hidden Layer

```
cnn.add(tf.keras.layers.Dense(units=128, activation='relu'))
```

- **Fully Connected Layer:**

  A Dense layer with 128 units and ReLU activation is added. This fully connected layer facilitates learning complex patterns and relationships from the flattened feature vectors, contributing to the model's capability for accurate flower species classification.

Constructing an Output Layer

```
cnn.add(tf.keras.layers.Dense(units=5 , activation='softmax'))
```

- **Output Layer with Softmax Activation:**

  The final Dense layer serves as the output layer with 5 units, representing the number of flower species classes. The softmax activation function is applied to convert the raw model outputs into probability distributions, facilitating multi-class classification.

Compilation of result

```
cnn.compile(optimizer = 'rmsprop' , loss = 'categorical_crossentropy' , metrics = ['accuracy'])
```

- **Model Compilation:**

  The compiled model is configured with the RMSprop optimizer, categorical crossentropy loss function, and accuracy as the evaluation metric. Compilation prepares the neural network for training by specifying the optimization strategy and metrics to monitor during the training process.

Flower Species Prediction

```
from keras.preprocessing import image
test_image = image.load_img('Prediction/daisy.jpg',target_size=(64,64))
test_image = image.img_to_array(test_image)
test_image = np.expand_dims(test_image,axis=0)
result = cnn.predict(test_image)
training_set.class_indices
```

- **Flower Species Prediction:**

  This is the process of predicting the flower species using the trained model. An image of a daisy is loaded, preprocessed, and fed into the model (cnn.predict). The training_set.class_indices provides the mapping of predicted indices to actual flower species classes.

Flower Species Classification

```
if result[0][0]==1:
    print('Daisy')
elif result[0][1]==1:
    print('Dandelion')
elif result[0][2]==1:
    print('Rose')
elif result[0][3]==1:
    print('SunFlower')
elif result[0][4]==1:
    print("Tulip")
```

- **Flower Species Classification:**

  The code snippet interprets the model predictions to classify the flower species. Depending on the output probabilities from the model, the code identifies and prints the predicted flower species, including options such as Daisy, Dandelion, Rose, Sunflower, and Tulip.

Model Prediction Results

```
print(result)
```

- Model Prediction Results:

  This line of code prints the raw output from the model prediction (result). The output is a probability distribution indicating the likelihood of the input image belonging to each flower species class. Analyzing these probabilities provides insights into the model's confidence in its predictions.

Output

```
if result[0][0]==1:
    print('Daisy')
elif result[0][1]==1:
    print('Dandelion')
elif result[0][2]==1:
    print('Rose')
elif result[0][3]==1:
    print('SunFlower')
elif result[0][4]==1:
    print("Tulip")
```

✓ 0.0s

```
Daisy
```

```
print(result)
```

✓ 0.0s

```
[[1. 0. 0. 0. 0.]]
```

Conclusion

In summary, the Flower Recognition project has deployed a Convolutional Neural Network to achieve accurate flower species identification. Through meticulous design, data augmentation, and model training, the CNN demonstrates proficiency in recognizing diverse flower patterns. The inclusion of a Dropout layer prevents overfitting, ensuring model reliability. Comprehensive evaluation metrics, such as accuracy and F1 score, validate the model's performance. The project's success is evident in the accurate classification of flower species, highlighting its potential applications. Future work involves exploring advanced architectures and dataset expansion. Special thanks to dataset contributors and library developers. This project showcases the effectiveness of CNNs in automating flower recognition, paving the way for advancements in computer vision and botanical identification.

References

- Dataset Credit:
  - https://www.kaggle.com/alxmamaev/flowers-recognition
- https://iopscience.iop.org/article/10.1088/1755-1315/1019/1/012021/meta

Thank You!