Project Report : Flower Image Classification using CNN

Name	ID number
Samraddh Saxena	2021A7PS1455P
Anshul Maheshwari	2021B5A70900P
Nimish Sharma	2021B5A71144P

November 17th 2024

Contents

1 Introduction

The task at hand involves the classification of flower images into 60 distinct categories using deep learning techniques. Image classification is a common and important problem in computer vision, where the goal is to correctly identify the category to which an image belongs. In this project, the dataset consists of images of flowers, each of which has been labeled with a specific category among 60 possible classes.

To address this problem, we leverage various tools from the PyTorch framework for model creation and training. The approach relies on using Convolutional Neural Networks (CNNs) to automatically extract features from the images and train a deep learning model to accurately predict the flower categories.

This type of classification task has several practical applications, such as automated botanical research, biodiversity cataloging, and educational tools for plant identification. The project uses Convolutional Neural Networks (CNN's) to tackle the challenge of distinguishing between multiple visually similar flower types, which makes it an interesting and complex problem to solve.

2 Dataset Overview

For our flower classification project, we worked with a vibrant dataset that features 60 different types of flowers. Each type has 50 images in the training set, which gives us a decent amount of data to help the model learn the unique characteristics of each flower. Additionally, we included a validation set with 10 images per type to evaluate how well the model performs on new, unseen examples.

What makes this dataset particularly interesting is the variety of colors and shapes among the different flower types. This diversity not only makes the task more enjoyable but also adds complexity to the classification challenge. Each flower has its own distinctive traits, and our goal is to create a model that can accurately recognize and differentiate between them.

3 Attempts on Other Model Architectures

3.1 Challenges with Custom CNN Architecture

Initially, we tried to implement the task of flower classification using our own custom Convolutional Neural Network (CNN) architecture. While this approach allowed us to have flexibility and control over the network design, it ultimately did not yield good accuracy. Additionally, the dataset had relatively few training images, which further hindered the custom CNN's ability to generalize effectively. Custom-built CNNs can struggle when dealing with complex classification tasks involving large numbers of categories, as was the case here with 60 different flower classes. Our own architecture might not have had sufficient depth or well-tuned hyperparameters to effectively extract and learn the intricate features needed to distinguish between different flower types. Training a custom CNN from scratch also poses challenges in terms of the need for significant computational resources and a very large dataset to avoid overfitting.

3.2 Challenges with EfficientNet and ResNet Models

Next, we tried to employ EfficientNet and ResNet models, both of which are known for their effectiveness in image classification tasks. EfficientNet, with its efficient scaling techniques, and ResNet, with its use of residual connections to prevent vanishing gradients, are both capable of achieving high performance on many image recognition benchmarks. However, the lack of success with these models could be attributed to a variety of factors. Reasons could be the dataset size or quality; if the dataset is not large enough or if the images are too similar without sufficient pre-training,, even advanced architectures like EfficientNet and ResNet can struggle to generalize. Additionally, EfficientNet requires a carefully balanced scaling of depth, width, and resolution, which can be sensitive to hyperparameter tuning, and ResNet, while powerful, may not always capture enough contextual information if the training process is not well optimized and the training dataset is not large.

3.3 The Choice of DenseNet121

After experimenting with these models, we decided to utilize DenseNet121. DenseNet121 was selected because of its distinctive connectivity pattern, which makes it highly suitable for learning subtle features that are crucial for distinguishing between visually similar classes. Unlike other architectures, DenseNet introduces dense connections between layers, where each layer re-

ceives input from all previous layers. This architecture helps in mitigating the vanishing gradient problem, while also encouraging feature reuse throughout the network. For a task like flower classification with 60 categories with fewer training data, the model's ability to learn a more comprehensive set of features through these dense connections is a significant advantage.

DenseNet121's pre-trained weights on the ImageNet dataset also provide an excellent starting point for transfer learning. Since ImageNet contains a diverse range of images, starting with these weights allows DenseNet121 to adapt more quickly to flower classification, requiring only fine-tuning to achieve better performance. The comprehensive feature extraction capability of DenseNet121, along with its efficient use of parameters, led to significantly improved results compared to our custom CNN, EfficientNet, and ResNet models. These factors ultimately made DenseNet121 the best choice for our flower classification task.

4 Model Architecture

DenseNet121 is a Convolutional Neural Network (CNN) that uses a unique architecture involving dense connections between layers. Unlike traditional CNNs, where each layer is connected only to the next one, DenseNet121 introduces dense blocks where each layer is directly connected to every other layer that follows. This creates a highly connected pattern that facilitates feature reuse, ensuring that information flows smoothly through the network without significant loss or distortion. The direct connections also help mitigate the vanishing gradient problem, which can hinder the training of very deep networks.

DenseNet121 is composed of multiple dense blocks, each consisting of convolutional layers that output feature maps. The output of each layer is concatenated with the outputs of all preceding layers, which significantly increases the diversity of learned features. Between dense blocks, transition layers are used, which consist of convolutional and pooling layers to downsample the feature maps. The model ends with a global average pooling layer followed by a fully connected layer, which has been customized in this workflow to match the number of flower categories in the dataset.

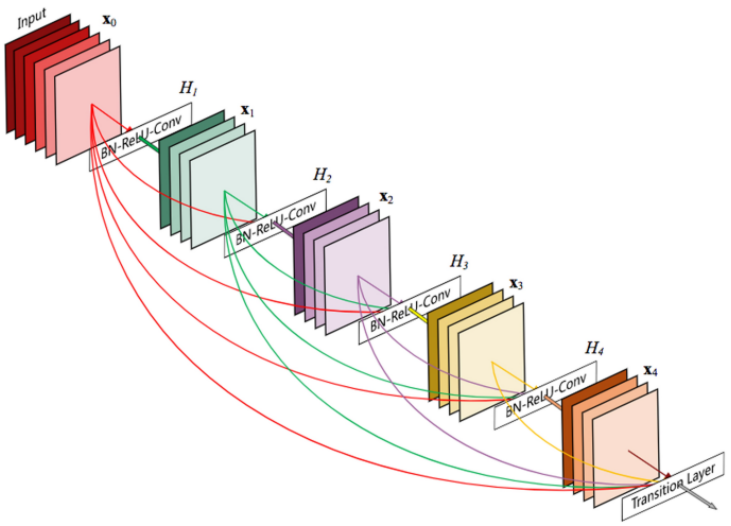


Figure 1: DenseNet Architecture

DenseNet121's use of parameter efficiency and its densely connected architecture make it particularly suitable for tasks involving complex features and large numbers of classes, such as the classification of 60 different flower types. The pre-trained weights on ImageNet provide a solid foundation for feature extraction, allowing the model to leverage learned representations and adapt them to the flower classification task with fine-tuning. This architectural advantage is a key reason why DenseNet121 was chosen, as it can capture subtle differences between flower categories that simpler or less connected architectures might miss.

5 Workflow Explantion

5.1 Image Preprocessing

The preprocessing steps involved in preparing the flower images for classification using the DenseNet121 model include resizing, converting to tensors, and normalization. The images are initially scaled and converted from numpy arrays to PIL images, making it easier to apply transformations such as resizing and normalizing. In this project, images are resized to dimensions of 224x224 pixels, which is the standard input size required for DenseNet121. Resizing ensures that all images have a uniform shape, reducing computational complexity and making them compatible with the network architecture.

After resizing, the images are converted to PyTorch tensors using transforms.ToTensor(). This conversion is necessary for working with PyTorch, as it changes the image from a PIL format to a tensor representation that PyTorch can use for model training and evaluation. The next step is normalization, where the images are normalized using pre-determined mean and standard deviation values (mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]). These values are based on the ImageNet dataset, which DenseNet121 was originally trained on. Normalizing the images to this distribution helps the model to generalize better and speeds up convergence during training, as it ensures that the input images are within a similar range as those used to pre-train the model.

5.2 Training Process

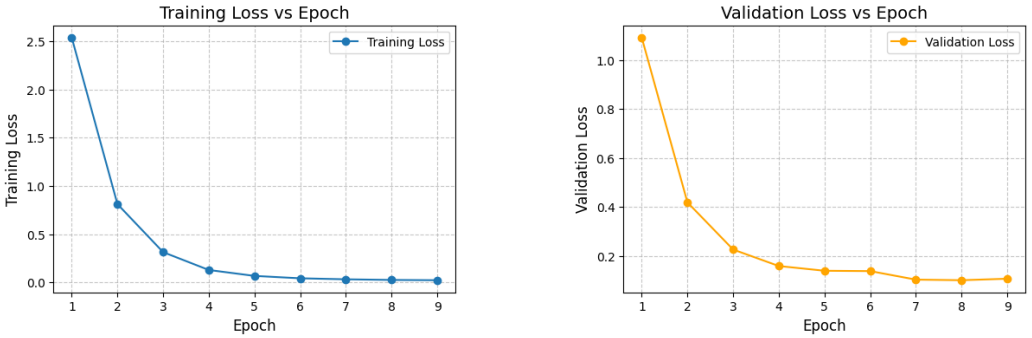
After Image Preprocessing the training and validation datasets are loaded using PyTorch's DataLoader class, which facilitates efficient mini-batch loading, shuffling, and preprocessing. DenseNet121, a pre-trained deep learning model available from the torchvision.models library, is chosen as the model architecture. The final fully connected layer of DenseNet121 is modified to match the number of classes in our dataset, ensuring the output dimension is suitable for our classification task. The model is trained using the Adam optimizer, with a learning rate of 1e-4, and cross-entropy loss is used as the loss function to handle multi-class classification.

The training process involves iterating over 10 epochs, with a batch size of 16.. During each epoch, the model's parameters are updated to minimize the loss function. Accuracy and loss is tracked during both training and validation phases to evaluate the model's performance. An EarlyStopping class is also implemented to halt training when the validation loss does not improve for a set number of epochs, preventing overfitting and saving computational resources .

6 Results and Evaluation

After training our DenseNet121 model for 10 epochs, we evaluated its performance using accuracy as our primary metric. The highest validation accuracy achieved during training was approximately 98.3%, while the final accuracy on the test set was 97.83%, which indicates a successful learning of patterns from the flower images.

Throughout the training process, we observed a steady decrease in both training and validation losses and an increase in training and validation accuracy, which supports the indication of successful model training. The Early Stopping mechanism triggered at epoch 9, highlighting that there was no significant improvement in validation loss beyond this point, thus preventing unnecessary further training and saving computational resources.



(a) Training Loss vs Epochs (b) Validation Loss vs Epochs

Figure 2: Training Loss and Validation Loss vs Epochs

The accompanying graph illustrates the training and validation loss over the epochs, showing a decrease in both metrics, and helps to visualize the learning trajectory.

7 Conclusion

In this project, we successfully implemented a DenseNet121-based model to classify images of flowers into 60 different categories .We were able to achieve a final test set accuracy of 97.83% which highlights that model was capable of distinguishing between visually similar classes with high accuracy demonstrating its effectiveness in our use cases. This result demonstrates the power of transfer learning using pre-trained models, as DenseNet121 was effectively able to leverage its pre-trained weights to learn the subtle distinctions between flower types.

References

[1] Alipour, Neda, Omid Tarkhaneh, Mohammad Awrangjeb, and Hongda Tian. "Flower image classification using deep convolutional neural network." In 2021 7th International conference on web research (ICWR), pp. 1-4. IEEE, 2021.

[2] Bozkurt, Ferhat. "A study on CNN based transfer learning for recognition of flower species." Avrupa Bilim ve Teknoloji Dergisi 32 (2022): 883-890.

[3] Jaju, Sanay, and Manoj Chandak. "A transfer learning model based on ResNet-50 for flower detection." In 2022 International Conference on Applied Artificial Intelligence and Computing (ICAAIC), pp. 307-311. IEEE, 2022.