Project Report: Flower Image Classification using CNN

Name	**ID number**
Aditya Patel	2021A7PS2433P
Sugeet Sood	2021B4A71746P

November 17, 2024

Made as a part of the course
Deep Learning (CS F425)

Contents

1 Introduction

The field of computer vision has come a long way, especially with the rise of deep learning and convolutional neural networks (CNNs). For this project, we focused on flower classification using CNNs, where the goal was to classify images of flowers into 60 different categories. The dataset we worked with had 3,600 RGB images, split into training and validation sets, with each image sized at 256x256 pixels.

Our main objective was to design a CNN-based solution that could achieve high accuracy on the validation dataset. While we did train some custom architectures from scratch, we also used transfer learning to take advantage of pre-trained CNN models. This approach helped us speed up the training process, improve performance, and extract meaningful features from the images more effectively. We tested several pre-trained models, fine-tuning them and adapting them specifically for the flower classification task.

In this report, we walk through the entire process, from data pre-processing and feature extraction to model training and evaluation. We also dive into the transfer learning methods we used, the pre-trained models we experimented with, and how they impacted accuracy and performance. By the end of this project, we gained hands-on experience with CNN architectures and learned how they can be tailored to specific domains. Along the way, we also tackled the challenges that come with multi-class classification in image datasets.

2 Dataset Overview

The dataset used in this project comprises images of flowers, categorized into 60 distinct classes. Each class contains five examples, making a total of 300 images. We also included a validation set of pictures with 10 images per type to test the model performance on an unseen set of data.
One noteworthy aspect of this dataset is the inherent challenge posed by categorising certain flowers, which may exhibit a variety of colours but are classified under a single label. For instance, a flower species may appear in shades of red, yellow, or white, yet all colour variations are grouped under the same class

3 Attempts on Other Model Architectures

Throughout the process of optimizing our image classification model, we explored multiple architectures to identify the one best suited to our dataset. This journey involved using both well-established and custom approaches, each with unique advantages and limitations.

3.1 Attempt with EfficientNet Model

We started out using EfficientNet because it's well-regarded for balancing accuracy with computational efficiency. The model uses a compound scaling method to adjust width, depth, and resolution, which sounded like a good fit for our needs. However, even after spending a lot of time fine-tuning the hyper-parameters, it didn't perform as well as we'd hoped. It struggled to hit the accuracy we were aiming for, which made it clear that it wasn't adapting well to the unique aspects of our dataset. Since EfficientNet is designed more for general-purpose tasks, it might not have been flexible enough to pick up the specific features in our data. In the end, its feature extraction just wasn't strong enough for our classification problem.

3.2 Attempt with ResNet Model

After that, we gave ResNet a try. This architecture is known for introducing residual connections, which help solve the vanishing gradient problem by improving how gradients flow during training. Thanks to its skip connections, ResNet made the training process more stable and consistent, which was promising. However, the accuracy was significantly better than EfficientNet but it wasn't up to the mark on an absolute level. One possible reason for this could be ResNet's difficulty in capturing the more intricate features of our dataset. Our data seemed to demand more complex interactions across layers, and ResNet's design might not have been able to take full advantage of the unique characteristics in our dataset.

3.3 Attempt with DenseNet201

After these iterations, we selected DenseNet as the architecture for our final implementation. Specifically, we chose DenseNet-201 due to its depth and the balance it offers between computational efficiency and model complexity. DenseNet-201's densely connected layers allow each layer to receive inputs from all preceding layers, facilitating efficient feature reuse and better gradient flow. This architecture proved beneficial for our flower classification task, as it allowed for comprehensive data representation and captured subtle differences between flower categories. Unlike ResNet, which uses skip connections primarily for gradient issues, DenseNet's dense connectivity encouraged feature exploitation at various levels of abstraction, making it well-suited for our dataset. The resultant higher accuracy indicated that DenseNet effectively learned and generalized well, capturing intricate patterns within our diverse flower dataset.

This iterative process of experimentation highlighted the importance of model selection in achieving high performance. DenseNet emerged as the most suitable solution due to its efficient feature reuse and robust gradient flow, leading to significant improvements in accuracy and generalization. This experience underscored the need for careful consideration of architectural strengths concerning dataset-specific challenges, ultimately helping us achieve a model that met our performance objectives for flower classification.

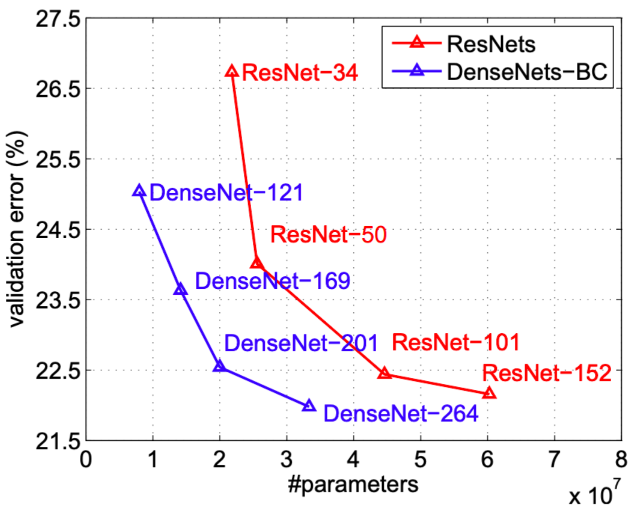


Figure 1: DenseNet-BC requires about 1/3 of the parameters as ResNet to achieve comparable accuracy.[1]

4 Model Architecture

On testing the aforementioned models, we finally decided to use DenseNet. Let's look at its architecture to better understand its working and why it works ideally for this project.

DenseNet concatenates the identity mappings which results in the data preservation in an efficient manner. All the inputs from the preceding layers are concatenated and then passed into the non-linear transforms. These transforms typically include a series of Batch Normalisation followed by a Rectified Linear Unit and finally a 3x3 convolution layer. To address the issues generated by map sizes of different layers, we need to use transition layers, which include BN, 1x1 convolution layer followed by a 2x2 average pooling layer.

The Bottleneck Layer and Growth Rate both result in reducing the number of parameters, the Bottleneck layer consists of 1x1 convolution layer which is placed before each 3x3 convolution in the Dense Block. This allows for computational efficiency by reducing the feature maps.

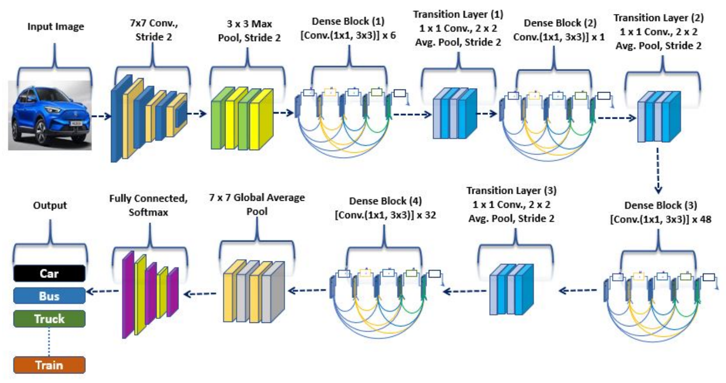

Input Image
7x7 Conv., Stride 2
3 x 3 Max Pool, Stride 2
Dense Block (1) [Conv.(1x1, 3x3)] x 6
Transition Layer (1) 1 x 1 Conv., 2 x 2 Avg. Pool, Stride 2
Dense Block (2) Conv.(1x1, 3x3)] x 1
Transition Layer (2) 1 x 1 Conv., 2 x 2 Avg. Pool, Stride 2
Dense Block (3) [Conv.(1x1, 3x3)] x 48
Transition Layer (3) 1 x 1 Conv., 2 x 2 Avg. Pool, Stride 2
Dense Block (4) [Conv.(1x1, 3x3)] x 32
7 x 7 Global Average Pool
Fully Connected, Softmax
Output
Car
Bus
Truck
Train

Figure 2: DenseNet Architecture[2]

In DenseNet, the growth rate, usually denoted as k, refers to how many new feature maps each layer adds to the network. When you get to layer l, the total number of input feature maps is calculated as $k_0 + k(l-1)$. Typically, this growth rate is kept small (eg. k = 12) because every layer already has access to all the feature maps from the previous layers. This setup allows DenseNet to capture the overall state of the network without much redundancy. Unlike other convolution networks, they save resources by reusing

earlier feature maps instead of repeatedly learning the same features.
There are different implementations of the DenseNet architecture, if we wanted to restrain the number of parameters even more we could have chosen to go with DenseNet-121 or DenseNet-169. We found empirically that DenseNet-201 performed better than the others for our use case.

5 Workflow Explanation

5.1 Image Preprocessing

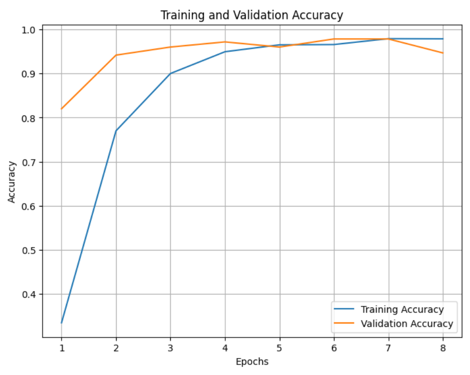
Image preprocessing is an essential step in preparing images for training a deep learning model. One of the first things we do is normalize the pixel values, which means scaling them from their original range of 0-255 (used for RGB images) down to a range of 0 to 1. This is done by simply dividing each pixel value by 255. The reason for this is that having all values on the same scale makes it easier for the model to learn effectively, as it prevents large gradients and ensures that all features contribute equally. Another important part of preprocessing is data augmentation. This involves applying random transformations to the images, like rotating, zooming, flipping, or shifting them slightly. These variations mimic the kinds of changes we might see in real-world scenarios and help the model generalize better by making it less likely to overfit to the training data. Lastly, the images need to be resized to match the input dimensions expected by the model. For instance, a model like DenseNet201 requires images to be 224x224 pixels with 3 channels for RGB. Resizing and reshaping the images ensures they are ready to be fed into the model, making these preprocessing steps vital for a smooth and effective training process.

5.2 Training Process & Choice of Hyperparameters

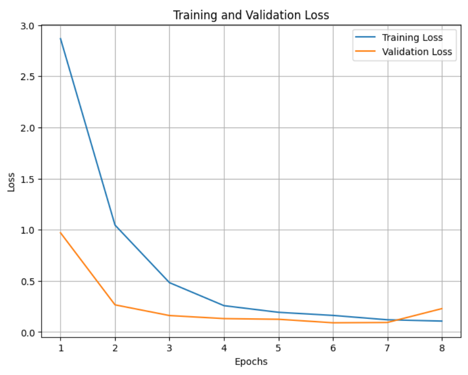
The choice of hyperparameters played a crucial role in the model's performance and training efficiency. The model was trained with a batch size of 16, a learning rate of 1e-4, and the Adam optimizer, which is well-suited for handling complex optimization problems due to its adaptive learning capabilities. The loss function used was categorical crossentropy, a standard choice for multi-class classification tasks, ensuring the model minimizes the difference between predicted and actual probabilities. The model stopped training early due to the early stopping callback, which monitored the validation loss and prevented overfitting by halting training when no significant improvement was observed.

6 Results and Evaluation

The DenseNet201 model was trained for 10 epochs and we tested the performance of the model using accuracy as the metric. The peak in the validation accuracy was 97.83% which was also the final accuracy. The highest training accuracy reached was 97.90% which finally came out to be 97.87%



(a) Training and Validation Accuracy vs Epochs



(b) Training and Validation Loss vs Epochs

Figure 3: Plots for Loss and Accuracy

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

7 Conclusion

In this project, after trying out various CNN models we narrowed down our approach with the DenseNet-201 model and successfully implemented it to classify the images of flowers into 60 different categories. Using the technique of transfer learning and by making appropriately tailoring the model to our use case we were able to achieve an accuracy of 95.67%.

References

[1] Huang, Gao Liu, Zhuang van der Maaten, Laurens Weinberger, Kilian. (2017). Densely Connected Convolutional Networks. 10.1109/CVPR.2017.243.

[2] Akhtar, Malik Mahum, Rabbia Shafique Butt, Faisal Amin, Rashid El-Sherbeeny, Ahmed Lee, Seongkwan Shaikh, Sarang. (2022). A Robust Framework for Object Detection in a Traffic Surveillance System. Electronics. 11. 3425. 10.3390/electronics11213425.