UNIVERSITY OF WOLVERHAMPTON

HERALD COLLEGE KATHMANDU

Artificial Intelligence and Machine Learning (6CS012)

Image Classification with Convolution Neural Network (CNN)

University ID: 2358554

Name: Sabin Chaulagain

Group: L6CG7

Module Leader: Mr. Siman Giri

Module Tutor: Ms. Durga

Submission Date: 20th May 2025

Abstract

The project aims at developing a deep learning model which can differentiate different categories of flowers from image data. There are 5,169 images classified according to five different species of flowers contained in the database. Images for training were Daisy, Dandelion, Rose, Sunflower and Tulip. This aims to develop a system with reliance on the Convolutional Neural Networks (CNNs) for the classification of various kinds of flowers in images.

Two sections are considered regarding the focal point of the project. From the beginning, a CNN model was developed from scratch, starting with a lean baseline architecture which has three convolutional and three fully connected layers. The model was extended by adding additional layers and dropout regularization to its depth. The two model versions were tested with such metrics as accuracy, precision, recall, and F1-score. Experimental results were drawn by using testing for both SGD and Adam optimizers to find which one of them facilitated fast training and convergence.

In the second phase of the study, pre-trained VGG16 model was used to allow transfer learning. Upper layers were adapted and adjusted for the problem of flower classification. The experiments showed that although the deep custom CNN managed to perform strongly, the VGG16 model with Adam optimization converged much faster and generalized better.

The research in conclusion shows that the application of transfer learning in CNNs can ensure one enjoys robust image classification performance, in cases with limited data.

Contents

1. Introduction

1.1. What is your data about? What is image classification task?

The research relies on a flower images data set that consists of five flower types: Daisy, Dandelion, Rose, Sunflower, and Tulip. There are 5,169 images in the collection distributed evenly among the two flower categories with about 1,000 images in each class. Each image in the database has been provided with its precise label that is described as per the flower it represents. The primary objective of this project is to develop a model that would be able to classify images of flowers into one of the five defined flower categories.

Image classification is one of the most widespread activities in computer vision. It means labeling pictures with text descriptions that describe what they depict. On the surface of things, this task might look easy but in fact it requires a lot of attention to detail when analyzing hundreds of little visual details in the image. Since Convolutional Neural Networks (CNN's) are expert in learning and making sense of hierarchically structured visual patterns, they are a professional match for this classification task. The value of image categorization can be seen in applications ranging from type identification for plants to people identification and conditions assessment from medical scans.

1.2. Aims and Objectives

Aims

- To create a CNN model that can classify different types of flowers from images.
- To train the model with the flower dataset so that it learns to classify each flower correctly.
- To improve the accuracy of the model using a pre-trained model like VGG16 through transfer learning.

Objectives

- Train a CNN model to determine the type of flower.
- Enhance the model by exposing it to many examples of flowers.
- Test the model to see how well it can recognize flowers that it has not been trained on.

- Compare the performance of our pre-trained CNN model with a pre-trained model to see which one performs better.

1.3. What is CNN and Why is it used for Image classification?

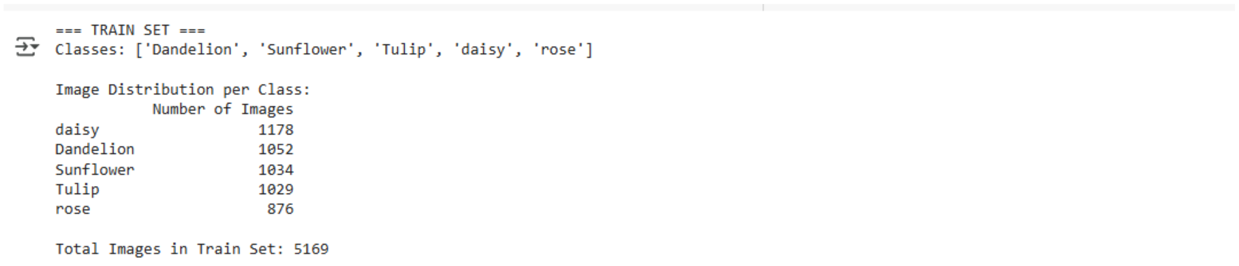
CNNs are a deep learning model that is particularly well-suited for image processing and comprehension. It allows computers to discern images based on patterns of shape or color. CNNs are particularly good at performing an analysis of images, where computers must comprehend and recognize objects within visual content (Albawi, S., Mohammed, T.A. and Al-Zawi, S., 2021).

Let us now explore why Convolutional Neural Networks are so effective at image classification:

- They learn patterns step by step starting from simple shapes to more complex ones.
- They process images piece by piece in the same manner as we do with these images.
- They identify objects irrespective of the place of occurrence within the image.
- Owing to the few parameters that they have, CNNs can be trained faster, and work well on large sets of images.

2. Dataset

For this project, we have used a manual flower image data set with five different classes: The collection consists of pictures of daisies, dandelions, roses, sunflowers, and tulips. Most of the data was obtained from Kaggle, a well-known center of machine learning data (Sleam, 2024). While the Kaggle data set was a great starting point, it had a couple weaknesses such as the fact that there were marked or watermarked images, as well as minor class imbalance. In response to these problems,: DLarge images with watermarks that had been handpicked were removed from the dataset. From open websites (like Google Images), additional images of each flower type were obtained to contribute towards class balancing. By typing images into the appropriate class folders, we collected a special dataset combining content from the Kaggle repository and images from the internet.



```
=== TRAIN SET ===
Classes: ['Dandelion', 'Sunflower', 'Tulip', 'daisy', 'rose']

Image Distribution per Class:
           Number of Images
daisy                  1178
Dandelion              1052
Sunflower              1034
Tulip                  1029
rose                    876

Total Images in Train Set: 5169
```

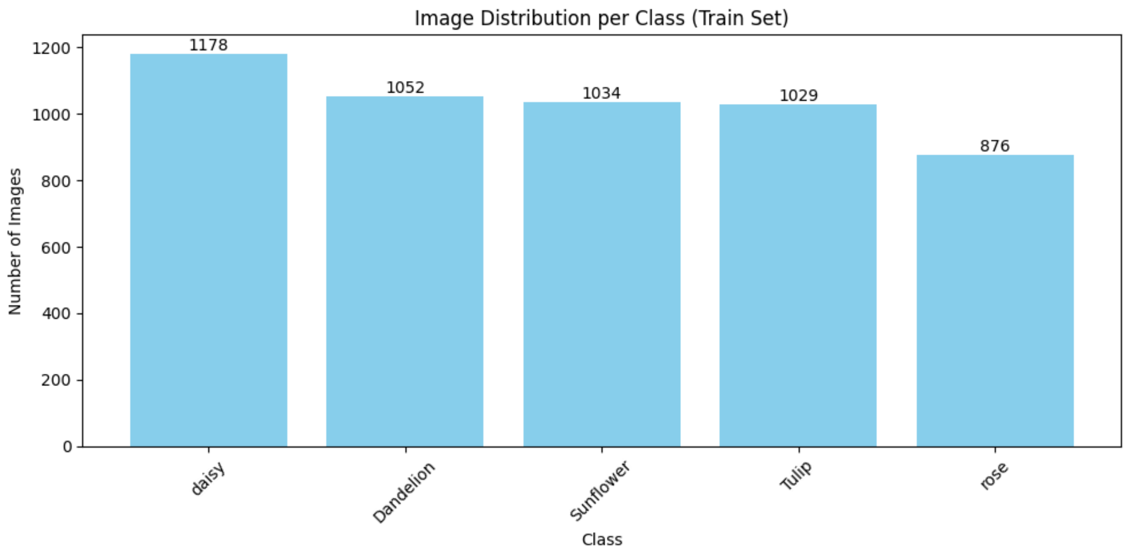


Figure 1: Number of Images per Class (Train)

Image Resolution and Preprocessing

- Resolution: For consistency in input for the CNN, all images were rescaled to 180 × 180 pixels.
- Normalization: By normalizing pixel values to be between 0 and 1, we increased training speed while keeping stability of the model.
- Augmentation: In order to increase the data diversity and prevent overfitting, some data augmenting strategies were applied as part of the training process which comprises, Rotation, Horizontal Flip, Zooming, Brightness adjustment.

Labelling and Structure

Images were organized in separate folders for each class, which allowed automated labeling using directory names with flow_from_directory(). Manual annotation tools were not needed due to this folder-based structure.

2.1. Challenges Faced

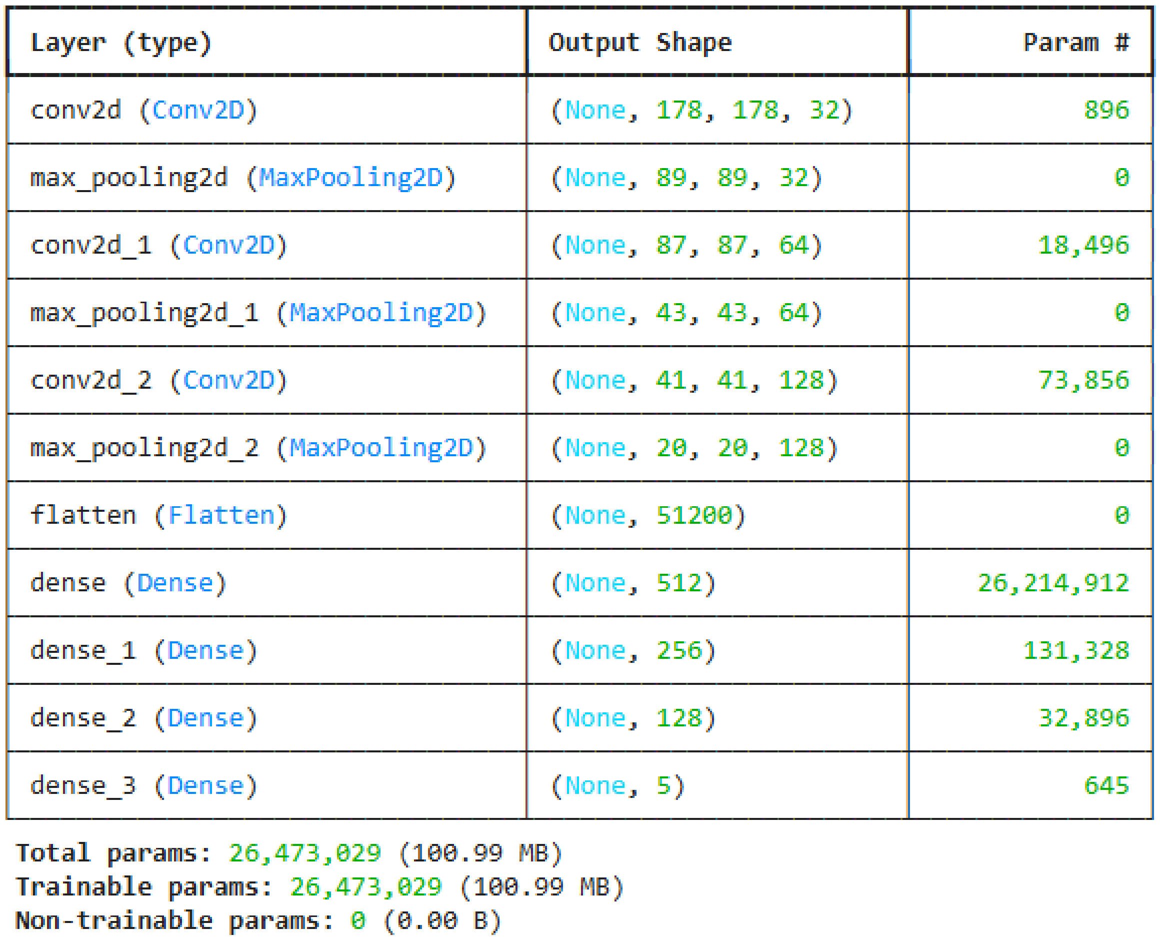
- Class imbalance: For some classes such as Rose, there was not enough imagery that called for additional collection and processing.
- Image noise: Most images held logos or overlaid text or were poorly lit, which required manual selection.
- Resolution variance: There was dimensionality variation in the original images, and uniformity of the model required resizing and cropping to be maintained.

3. Methodology

Convolutional Neural Networks (CNNs) that were part of deep learning models were employed in this study whereby the images of five different flowers were classified. Two major models were the object of experimentation behind this work. two models were constructed, a simpler, baseline one and a deeper, more complex one.

3.1. Baseline Model

The baseline model was quite simple and built from scratch, it includes the following:



Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 178, 178, 32)	896
max_pooling2d (MaxPooling2D)	(None, 89, 89, 32)	0
conv2d_1 (Conv2D)	(None, 87, 87, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 43, 43, 64)	0
conv2d_2 (Conv2D)	(None, 41, 41, 128)	73,856
max_pooling2d_2 (MaxPooling2D)	(None, 20, 20, 128)	0
flatten (Flatten)	(None, 51200)	0
dense (Dense)	(None, 512)	26,214,912
dense_1 (Dense)	(None, 256)	131,328
dense_2 (Dense)	(None, 128)	32,896
dense_3 (Dense)	(None, 5)	645

```
Total params: 26,473,029 (100.99 MB)
Trainable params: 26,473,029 (100.99 MB)
Non-trainable params: 0 (0.00 B)
```

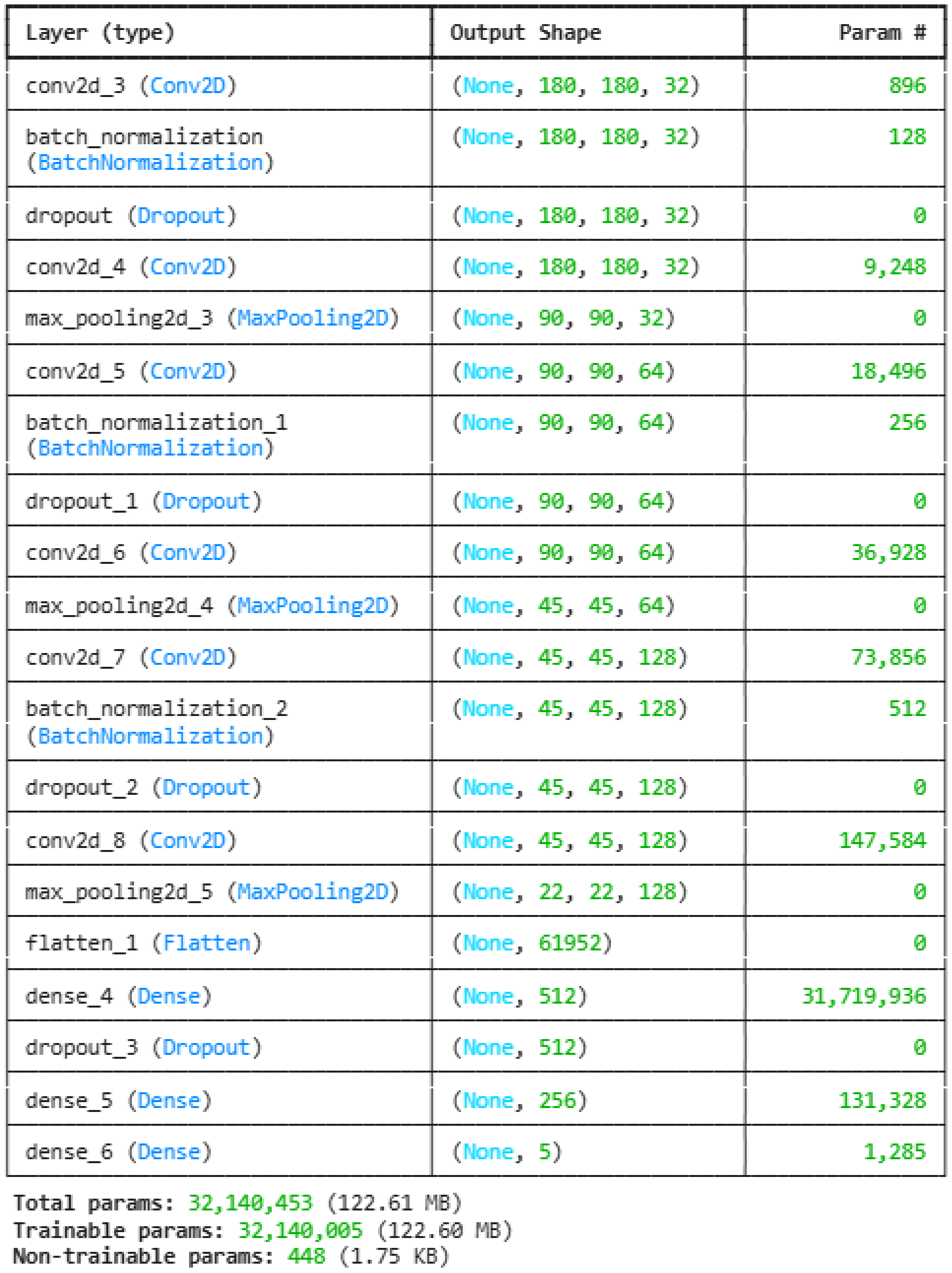
Figure 2: Architecture diagram of the baseline CNN model.

The baseline CNN model structure is provided below (Figure 2), consisting of three convolutional layers, pooling, and dense layers.

- Three convolutional layers, each of which is preceded by a pooling layer to down sample the feature maps.
- Three fully connected (dense) layers to predict and learn patterns.
- Soft max activation output layer to predict which of the five flower categories an image belongs.

At first, the model was trained to establish a norm, whereby the next versions could be measured.

3.2. Deeper Model



Layer (type)	Output Shape	Param #
conv2d_3 (Conv2D)	(None, 180, 180, 32)	896
batch_normalization (BatchNormalization)	(None, 180, 180, 32)	128
dropout (Dropout)	(None, 180, 180, 32)	0
conv2d_4 (Conv2D)	(None, 180, 180, 32)	9,248
max_pooling2d_3 (MaxPooling2D)	(None, 90, 90, 32)	0
conv2d_5 (Conv2D)	(None, 90, 90, 64)	18,496
batch_normalization_1 (BatchNormalization)	(None, 90, 90, 64)	256
dropout_1 (Dropout)	(None, 90, 90, 64)	0
conv2d_6 (Conv2D)	(None, 90, 90, 64)	36,928
max_pooling2d_4 (MaxPooling2D)	(None, 45, 45, 64)	0
conv2d_7 (Conv2D)	(None, 45, 45, 128)	73,856
batch_normalization_2 (BatchNormalization)	(None, 45, 45, 128)	512
dropout_2 (Dropout)	(None, 45, 45, 128)	0
conv2d_8 (Conv2D)	(None, 45, 45, 128)	147,584
max_pooling2d_5 (MaxPooling2D)	(None, 22, 22, 128)	0
flatten_1 (Flatten)	(None, 61952)	0
dense_4 (Dense)	(None, 512)	31,719,936
dropout_3 (Dropout)	(None, 512)	0
dense_5 (Dense)	(None, 256)	131,328
dense_6 (Dense)	(None, 5)	1,285

```
Total params: 32,140,453 (122.61 MB)
Trainable params: 32,140,005 (122.60 MB)
Non-trainable params: 448 (1.75 KB)
```

Figure 3: Architecture of the Deeper CNN with Dropout, Batch Normalization, and Regularization

In Figure 3 the deeper model adds more convolutional layers, batch normalization and dropout for enhanced learning capability.

The deeper layer was built to maximize the difference with baseline: it added complexity. It added:

- Increase in the convolutional layers to pick more informative features.

- Batch Normalization layers, which will increase the speed of training and lead to a more reliable convergence.
- Dropout layers to avoid overfitting by randomly dooming some neurons during training.

The aim of this model was to beat baseline accuracy due to its greater structural complexity.

3.3. Loss Function

The loss function selected for both models, the categorical cross-entropy, is appropriate for multiple classification areas.

3.4. Optimizers

Both models were investigated with two kinds of optimizers – namely, SGD and Adam, for comparison.

- SGD (Stochastic Gradient Descent): Selection of this optimizer provides for stable updates, though it performs rather slowly during its learning process.
- Adam: An optimizer that has automatic learning rate tuning to make the optimizer better than SGD. Both in terms of speed and performance, Adam normally dictates higher performance than SGD.

3.5. Hyperparameters

During training, the following settings were used:

- Learning Rate: 0.001
- Batch Size: 32
- Epochs: 20 (For both models)

These settings' options were based on the outcomes of initial testing to ensure a moderate balance between speed and model effectiveness.

4. Experiments and Results

This section describes the experiments to measure the performance of certain CNN models in classification of flowers. To estimate the effectiveness of different models, transfer learning was used for a baseline network, a deeper network, and a pre-train VGG16 model. The relationship between model complexity and performance was analyzed based upon accuracy, loss, the time needed to train, optimizer effectiveness and problems that arose during the training.

4.1. Baseline vs Deeper Architecture

Two CNN models—basic and advanced—were trained and studied to investigate how overall performance is influenced by greater complexity.

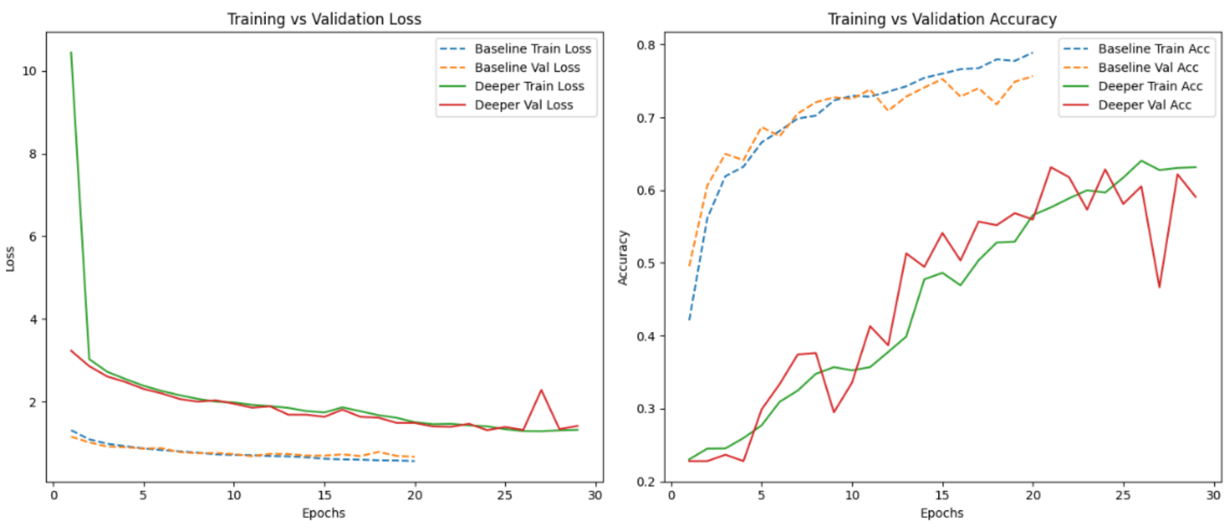


Figure 4: Comparison of Training and Validation Loss and Accuracy between the Baseline and Deeper CNN models. The deeper model achieves better generalization but shows more fluctuation due to its complexity.

The above image presents the training and validation accuracy and loss, highlighting the deeper model's improved generalization over the baseline.

Accuracy and Loss Comparison

The baseline model worked, although some signs of overfitting emerged. However, strong training performance was realized for the baseline while the validation accuracy was weakened, and the loss plateaued early. On the other hand, the deep model showed better accuracy in validation and loss, showing that the extra layers and regularization helped in improving the overall model generalization tendency.

Training Time

The deeper model took longer to train since it had more layers and parameters. However, the extra training time was worth it as it provided a more stable and accurate model.

There was a substantial improvement in terms of the model's performance when extra convolutional layers, batch normalization, and dropout were implemented. Through the addition of other layers, a model became more accurate in identifying chiasmus in flower pictures and eliminated overfitting problems. This shows that a highly developed deep CNN can easily outperform those of a basic CNN, even with a comparatively small dataset.

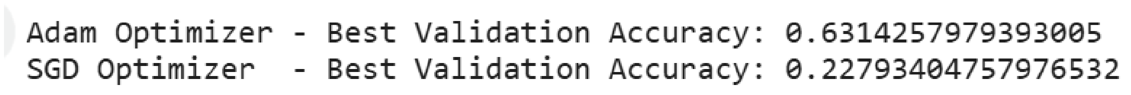
4.2. Computational Efficiency

The extra layers, and parameters in the deeper CNN model, oriented it to take more memory and time in training compared to the baseline. Despite the fast training of the baseline model, longer epochs were observed in the deeper model that had a higher complex architecture.

Use of Google Collab's GPU acceleration with both models sped up the training process. Without the use of a GPU, the training process would have been much delayed. With the help of the GPU, it was easier for us to add more images per batch and be able to train more complex models.

4.3. Training with Different Optimizers

Training the deeper model involved the use of two optimizers: SGD and Adam.



```
Adam Optimizer - Best Validation Accuracy: 0.6314257979393005
SGD Optimizer  - Best Validation Accuracy: 0.22793404757976532
```

Figure 5: Best validation accuracy comparison using Adam and SGD optimizers.

The figure above compares the performance of the two optimizers: Adam was better than SGD in validation accuracy.

- SGD (Stochastic Gradient Descent) improved slowly over time and needed more epoch to get satisfying accuracy. Training was slow and sometimes losses were also increased.
- Adam was better as it converged quickly and provided better accuracy in fewer epochs against SGD. During training, it automatically changed the learning rate to refine the model convergence.

Ultimately, Adam turned out to be the more effective optimizer, delivering both faster convergence as well as higher accurate results in comparison with SGD (Shorten, C. and Khoshgoftaar, T.M, 2021).

4.4. Challenges in Training

Several concerns were observed during the training of the models:

Overfitting: Baseline model performed well on training examples but failed to generalize well across validation sets, a pointer for over-fitting. To mitigate this issue, dropout and batch normalization are added to the deeper model (Sutskever, I., Martens, J., Dahl, G. and Hinton, G., 2022).

Underfitting: When the model was trained on earlier epochs using SGD, it had a hard time making amazing patterns out of the data.

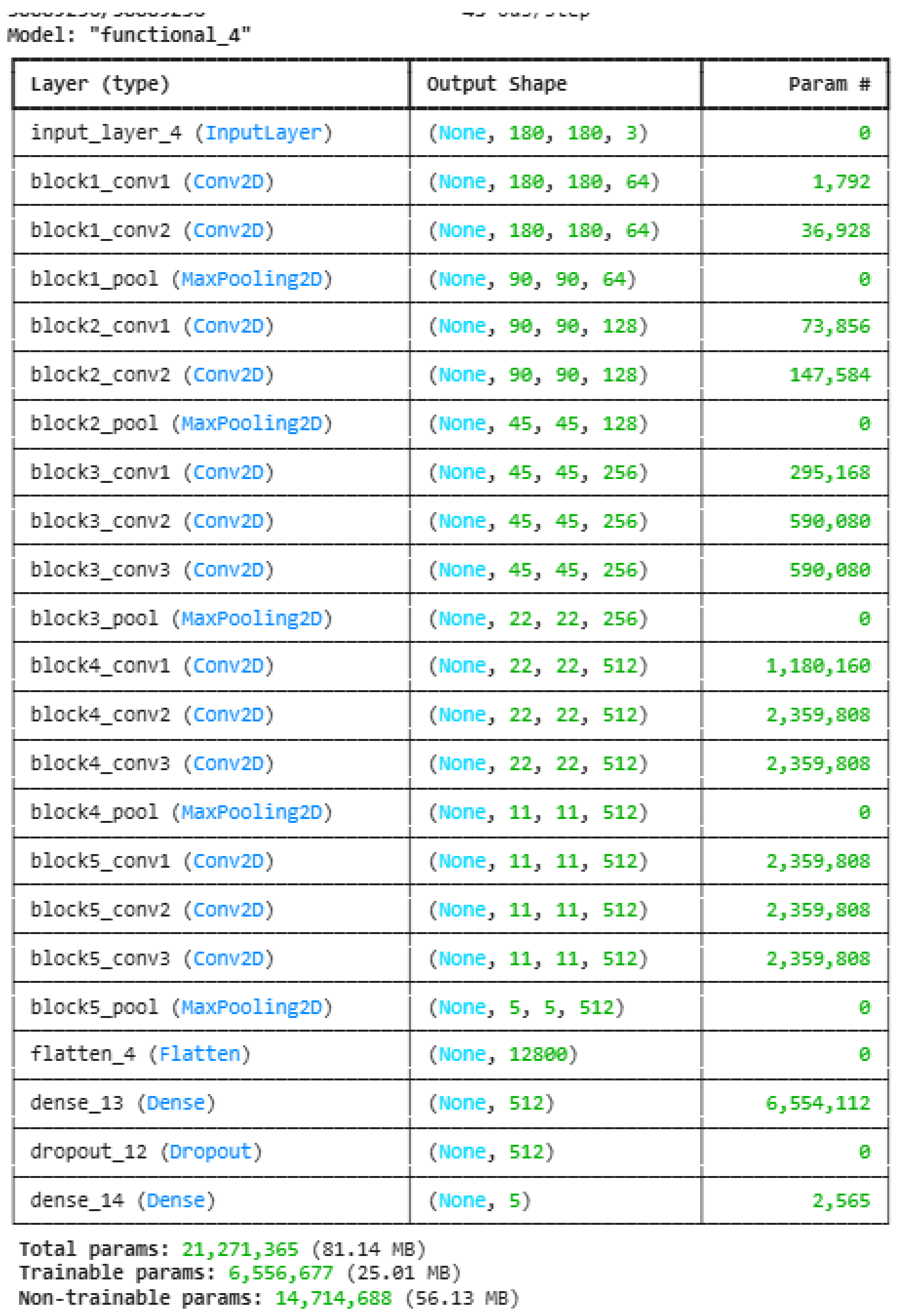
Training Stability: When fine-tuning the pre-trained model, reducing the learning rate turned out to be essential for achieving the stability of the training. The length of the training procedure depended both on the complexity of the model and the optimization algorithm deployed. Using Adam with the deeper model produced slower single-epoch training speeds but better general results. The ability to use Google Collab for GPU acceleration of training significantly facilitated handling of the CPU resources and reduction of the periods for training.

5. Fine-Tuning or Transfer Learning

Transfer learning was used in the case of the VGG16 model whose features were trained using the huge ImageNet database. The great ability of VGG16 to attain high accuracy using its deep network architecture is of advantage to most image classification tasks (Kukacka, J., Golkov, V. and Cremers, D., 2021).

Our initial step was featuring extraction through freezing VGG16's convolutional base and training only the new fully connected layers responsible for flower classification. This approach allowed the model to transfer what it learnt in ImageNet and apply it successfully to flowers.

Then, we allowed the model to be fine-tuned by untying some of the lower convolutional levels so that model could fine-tune its features and adapt better to the flower dataset. This made the model capable of changing the features that it detects, making it better suited to identifying the types of photographs of flowers.



```
Model: "functional_4"
┌──────────────────────────────────┬──────────────────────────┬──────────────┐
│ Layer (type)                     │ Output Shape             │      Param # │
├──────────────────────────────────┼──────────────────────────┼──────────────┤
│ input_layer_4 (InputLayer)       │ (None, 180, 180, 3)      │            0 │
├──────────────────────────────────┼──────────────────────────┼──────────────┤
│ block1_conv1 (Conv2D)            │ (None, 180, 180, 64)     │        1,792 │
├──────────────────────────────────┼──────────────────────────┼──────────────┤
│ block1_conv2 (Conv2D)            │ (None, 180, 180, 64)     │       36,928 │
├──────────────────────────────────┼──────────────────────────┼──────────────┤
│ block1_pool (MaxPooling2D)       │ (None, 90, 90, 64)       │            0 │
├──────────────────────────────────┼──────────────────────────┼──────────────┤
│ block2_conv1 (Conv2D)            │ (None, 90, 90, 128)      │       73,856 │
├──────────────────────────────────┼──────────────────────────┼──────────────┤
│ block2_conv2 (Conv2D)            │ (None, 90, 90, 128)      │      147,584 │
├──────────────────────────────────┼──────────────────────────┼──────────────┤
│ block2_pool (MaxPooling2D)       │ (None, 45, 45, 128)      │            0 │
├──────────────────────────────────┼──────────────────────────┼──────────────┤
│ block3_conv1 (Conv2D)            │ (None, 45, 45, 256)      │      295,168 │
├──────────────────────────────────┼──────────────────────────┼──────────────┤
│ block3_conv2 (Conv2D)            │ (None, 45, 45, 256)      │      590,080 │
├──────────────────────────────────┼──────────────────────────┼──────────────┤
│ block3_conv3 (Conv2D)            │ (None, 45, 45, 256)      │      590,080 │
├──────────────────────────────────┼──────────────────────────┼──────────────┤
│ block3_pool (MaxPooling2D)       │ (None, 22, 22, 256)      │            0 │
├──────────────────────────────────┼──────────────────────────┼──────────────┤
│ block4_conv1 (Conv2D)            │ (None, 22, 22, 512)      │    1,180,160 │
├──────────────────────────────────┼──────────────────────────┼──────────────┤
│ block4_conv2 (Conv2D)            │ (None, 22, 22, 512)      │    2,359,808 │
├──────────────────────────────────┼──────────────────────────┼──────────────┤
│ block4_conv3 (Conv2D)            │ (None, 22, 22, 512)      │    2,359,808 │
├──────────────────────────────────┼──────────────────────────┼──────────────┤
│ block4_pool (MaxPooling2D)       │ (None, 11, 11, 512)      │            0 │
├──────────────────────────────────┼──────────────────────────┼──────────────┤
│ block5_conv1 (Conv2D)            │ (None, 11, 11, 512)      │    2,359,808 │
├──────────────────────────────────┼──────────────────────────┼──────────────┤
│ block5_conv2 (Conv2D)            │ (None, 11, 11, 512)      │    2,359,808 │
├──────────────────────────────────┼──────────────────────────┼──────────────┤
│ block5_conv3 (Conv2D)            │ (None, 11, 11, 512)      │    2,359,808 │
├──────────────────────────────────┼──────────────────────────┼──────────────┤
│ block5_pool (MaxPooling2D)       │ (None, 5, 5, 512)        │            0 │
├──────────────────────────────────┼──────────────────────────┼──────────────┤
│ flatten_4 (Flatten)              │ (None, 12800)            │            0 │
├──────────────────────────────────┼──────────────────────────┼──────────────┤
│ dense_13 (Dense)                 │ (None, 512)              │    6,554,112 │
├──────────────────────────────────┼──────────────────────────┼──────────────┤
│ dropout_12 (Dropout)             │ (None, 512)              │            0 │
├──────────────────────────────────┼──────────────────────────┼──────────────┤
│ dense_14 (Dense)                 │ (None, 5)                │        2,565 │
└──────────────────────────────────┴──────────────────────────┴──────────────┘
 Total params: 21,271,365 (81.14 MB)
 Trainable params: 6,556,677 (25.01 MB)
 Non-trainable params: 14,714,688 (56.13 MB)
```

Figure 6: Model summary of the transfer learning architecture using VGG16 as the base model with frozen convolutional layers and custom dense layers for flower classification.

The details of the transfer learning model when VGG16 is used are presented in Figure 6, consisting of frozen convolutional layers, and custom classification layers.

Compared to the baseline and CNNs trained from scratch.

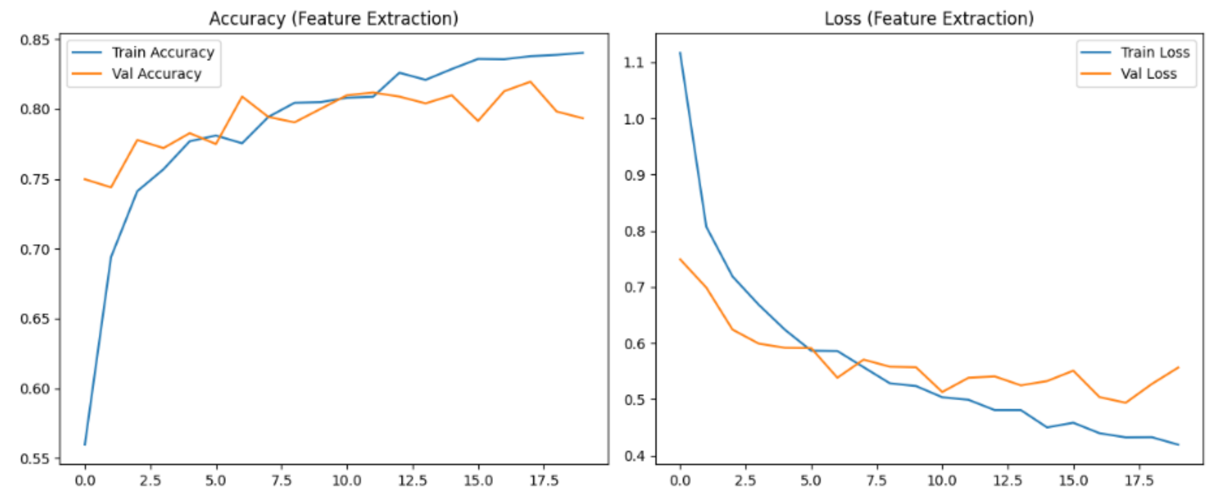


Figure 7: Training and Validation Accuracy and Loss during Transfer Learning with VGG16. The model demonstrates steady convergence and improved generalization with fewer epochs due to pre-trained feature extraction.

Performance of training and validation of the VGG16-based model is presented by above figure, showing fast convergence and high accuracy.

Our transfer learning model outperformed both. In the end, it provided more accuracy and a quicker convergence with fewer training epochs. Taking advantage out of pre-trained weights, the model's training was accelerated, and its generalization performance was improved.

Finally, invoking transfer learning through VGG16 resulted in better results for our project.

6. Conclusion and Future Work

This research investigated image classification via CNNs in a suite of images of five different classes of flowers. Our first method used to develop raw CNN from scratch, then a complicated model that incorporated regularization techniques, and finally moving to transfer learning using a pre-trained VGG16 model.

Key results:

Using the dropout and batch normalization got the deeper model to outperform the baseline in this experiment. The results from Adam optimizer were achieved faster and performed better than SGD. The VGG16 transfer learning model performed better than both custom CNNs, in terms of both accuracy and training speed.

Trade-offs:

The faster training and the less memory demand were incurred by models that performed better. Even though transfer learning was effective, good results required appropriate layer configuration and adjustment.

Limitations:

Proper identification of the images became more difficult if the color or shape of a flower was very similar. In the first few training phases, models were prone to over fitness, especially, lacking regularization.

Future Improvements:

Show a desire to augment flower dataset with other and different pictures to increase generalization generally. Tweak parameters like learning rate and batch size in a careful manner. It goes without saying that try to add some pre-trained network architectures, such as ResNet or Inception to facilitate the comparison-analyses. Combine Grad-CAM to make the model's attention visible and find the major portions of the image that help it classify. The study of the possibility of integration of CNNs with attentional processes or with their integration in hybrid models may result in better classification outcomes in the future.

References

Albawi, S., Mohammed, T.A. and Al-Zawi, S., 2021. Understanding of a convolutional neural network. *International Conference on Engineering and Technology (ICET),* pp. pp.1-6.

Kukacka, J., Golkov, V. and Cremers, D., 2021. Regularization for deep learning: A taxonomy. *IEEE Transactions on Pattern Analysis and Machine Intelligence,* 49(9), p. pp.3327–3346.

Shorten, C. and Khoshgoftaar, T.M, 2021. A survey on image data augmentation for deep learning. *Journal of Big Data,* 8(1), pp. pp. 1-48.

Sutskever, I., Martens, J., Dahl, G. and Hinton, G., 2022. On the importance of initialization and momentum in deep learning. *Proceedings of the International Conference on Machine Learning (ICML),* 28(3), p. pp.1139–1147.