Academic Year	Module	Report Number	Report Type
2025	AI and ML	01	Flower classification report

(AI and ML)

Flower Classification Using Deep Convolutional Neural Networks (CNNs)

Student Name: Anup Dangi

Student id: 2333319

Group: L6CG7

Module leader: Simon Giri

Tutor: Durga Pokharel

Submitted on: 5/16/2025

Abstract:

The goal of the research is to create a deep learning model that can identify various flower classifications from pictures. Images in the database are categorized according to five distinct flower species, totaling 5169. Daisy, Dandelion, Rose, Sunflower, and Tulip were used as training images. This project intended to create a system that relies on Convolutional Neural Networks (CNNs) to classify various flower types in images.

In this context, two components are examined with regard to the project's main focus. Initially, a CNN model with three convolutional and three fully connected layers was built from the ground up. The model was extended by adding dropout regularization to its depth and increasing the number of its layers. Two model variations were evaluated using criteria such F1-score, recall, accuracy, and precision. Results from experiments were obtained by testing the Adam and SGD optimizers to determine which one permitted the fastest training and convergence.

The VGG16 model, which was previously trained and allowed for transfer learning, was employed in the study's second phase. The upper layers were adjusted and adjusted to address the flower classification issue. The trials showed that the VGG16 model with Adam optimization converged more faster and generalized better than the deep custom CNN, notwithstanding the latter's success.

The results of this research demonstrate that, even in situations where data is scarce, the use of transfer learning in CNNs may guarantee strong picture classification performance.

Table of Contents

1. Introduction: 1
   1.1. Introduction to dataset used: 1
   1.2. Why Image classification task? 1
   1.3. Aims: 1
   1.4. Objective: 1
2. Dataset: 2
   2.1. Image resolution and pre-processing: 2
   2.2. Labeling and structure: 3
   2.3. Challenges faced: 3
3. Methodology: 3
   3.1. Baseline model: 3
   3.2. Deeper model: 4
   3.3. Loss function: 5
   3.4. Optimizer: 6
   3.5. Hyperparameter: 6
4. Experiments and result: 6
   4.1. Baseline vs Deeper Architecture: 7
   4.2. Computational efficiency: 7
   4.3. Training with different Optimizers: 8
   4.4. Challenges faced: 8
5. Fine-Tuning or Transfer Learning: 9
6. Conclusion: 10
7. References 11

Table of figures:

Figure 1 Image distribution per class..... 2
Figure 2 Baseline CNN model architecture 4
Figure 3 Deeper CNN model including dropout, batch normalization..... 5
Figure 4 Categorical cross-entropy formula 6
Figure 5 Comparison of training and validation loss..... 7
Figure 6 Training with different optimizer 8
Figure 7 Model summary of transfer learning..... 9
Figure 8 Training and validation accuracy..... 10

1. Introduction:

1.1. Introduction to dataset used:

Five different varieties of flowers are included in the data collection of flower photos used in the study. Sunflower, Tulip, Rose, Dandelion, and Daisy. The 5,169 photos in the collection are split equally between the two floral categories, with roughly 1,000 photos in each class. Every picture in the database includes a precise label that describes it according to the flower it depicts. This project's main objective is to create a model that can categorize flower photos into any of the five established flower types.

1.2. Why Image classification task?

One of the most common tasks in the field of computer vision is image classification. Text that describes what is depicted in the images is used to describe them. On the surface, this activity can appear simple, but in practice, it requires a great deal of focus on hundreds of small visual components in the image during analysis. Convolutional Neural Networks (CNNs) are a genuine professional at this kind of classification task since they are the best at learning and deciphering hierarchically structured visual patterns (Rawat, 2017). Type identification for plants, person identification, and condition analysis from medical scans are just a few examples of the advantages of image categorization.

1.3. Aims:

- To develop a CNN model capable of identifying various flower species from pictures.
- To train the model to accurately classify each bloom by using the flower dataset.
- To increase the model's accuracy by applying transfer learning to a pre-trained model, such as VGG16.

1.4. Objective:

- Train a CNN model to recognize what kind of a flower it is.
- Extend the model based on the experience of handling many examples of flowers.
- Evaluate the model to check how good it is at identifying flowers that it has not seen before.
- To determine which performs better, compare the performance of a pre-trained model and our pre-trained CNN model.

2. Dataset:

We used the manual flower image data set with five distinct classes for this research. Images of daisies, dandelion, roses, sunflowers, and tulips are included in the collection. The majority of the data was obtained from Kaggle, a well-known machine learning data lab. Despite being a decent place to start, the Kaggle Data Set had certain shortcomings, including watermarked or marked photos and a little class imbalance.

To address these issues, Hand-picked large photos with watermarks were removed from the collection. To assist balance the classes, additional pictures of each kind of flower were gathered from public websites (like Google Images). By entering the photographs into the appropriate class folders, we were able to compile a unique dataset that included both the Kaggle repository's content and the images we had obtained online.

Figure 1 Image distribution per class

2.1. Image resolution and pre-processing:

- Resolution: All photos were rescaled to 180 × 180 pixels to provide uniformity in input for the CNN.
- Normalization: We improved training time while maintaining model stability by normalizing pixel values to fall between 0 and 1.

- Augmentation: Rotation, horizontal flipping, zooming, and brightness change are some of the data augmenting techniques used throughout the training process to improve data diversity and avoid overfitting.

2.2. Labeling and structure:

Because the images were arranged in distinct folders for every class, flow_from_directory() could automatically label them using directory names. Because of this folder-based structure, manual annotation tools were not required.

2.3. Challenges faced:

- Inequality of classes: Some classes, like Rose, lacked sufficient imagery to warrant further gathering and processing.
- Image noise: The majority of the photos needed to be manually selected since they included logos, text overlays, or were dimly lit.
- Resolution variance: The source photos had dimensionality variation, therefore maintaining the model's homogeneity required cropping and resizing.

3. Methodology:

In this study, five distinct flower photos were categorized using Convolutional Neural Networks (CNNs), which are a component of deep learning models. This work was based on experiments with two major models. A baseline, more straightforward model and a deeper, more intricate model were built.

3.1. Baseline model:

Three convolutional layers followed by pooling layers, which are used to extract features and reduce spatial dimensions, define the baseline model in relation to what is typically referred to in CNN studies. The model comprises three fully linked (Dense) layers prior to the output layer after feature extraction. Prior to evaluating deeper or more optimized models, the baseline model seeks to provide a simpler architecture for preliminary performance comparisons.

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 Baseline CNN model architecture

The deep CNN model that was used in the project is summarized in the image. To lower the image dimension while keeping important features, it starts with three consecutive Conv2D layers and then moves on to the MaxPooling2D layer. The model applies data to four dense layers after the final feature map is flattened into a one-dimensional array. The output layer has five units (quantities of various flower classes), after which there is a large hidden layer with 512 neurons and two additional layers with 256 and 128 neurons in succession. The model has over 26.47 million trainable parameters total, indicating its deep and intricate structure designed to capture intricate patterns in visual data.

3.2. Deeper model:

Between the baseline, individual choices that improve performance by adding additional convolutional layers, and regularization methods like Batch Normalization and Dropout, the deeper model performs the best. Because of the prolonged feature extraction and overfitting prevention, this helps to achieve improved accuracy, stability, and generalization.

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 Deeper CNN model including dropout, batch normalization

The CNN architecture is substantially deeper in this example of a model summary, with six convolutional layers followed by batch normalization and dropout after chosen layers. The model's 32 million trainable parameters demonstrate the network's complexity and tremendous capacity. By combining pooling with a convolutional block, spatial overload is avoided and dimensions are decreased. Before the final classification layer (5 output classes), the model applies two dense layers (512 and 256 units) with Dropout applied to them. This model is far more resilient than the baseline version since normalization and dropout are used across the network to improve training efficiency and prevent overfitting.

3.3. Loss function:

In this project, the categorical cross-entropy loss function was employed, and this is the default one for multi-class classification tasks. As the flower classification problem is one of predicting a single correct class from a number of possible flower categories, categorical cross-entropy is perfect as it will penalize errors made while predicting

classes by using the difference between the true label distribution (one-hot) and the distribution of predicted probabilities from a softmax output layer.

Mathematically, categorical cross-entropy is defined as:

$$\text{Loss} = -\sum_{i=1}^{\text{output size}} y_i \cdot \log \hat{y}_i$$

Figure 4 Categorical cross-entropy formula

3.4. Optimizer:

- SGD (Stochastic Gradient Descent): Although it operates somewhat slowly throughout its learning phase, choosing this optimizer results in steady updates.
- Adam: An optimizer that outperforms SGD thanks to automated learning rate tuning. Adam often dictates better performance than SGD in terms of both speed and performance.

3.5. Hyperparameter:

A learning rate of 0.001 was employed during model training in order to achieve consistent convergence without endangering the model's ability to learn. Because it strikes a fair balance between calculation efficiency and model performance, the batch size of 32 was selected to maximize memory utilization and expedite training. For both models, the number of epochs was set at 20, which allowed for enough iterations on the training data to support learning without running the danger of overfitting. These settings were carefully chosen to create a balance between the training speed and overall efficacy of the used models, taking into account the outcomes of preliminary testing.

4. Experiments and result:

The tests to evaluate the effectiveness of specific CNN models in flower classification are described in this part. Transfer learning was applied to a baseline network, a deeper network, and a pre-train VGG16 model in order to measure the efficacy of various models. Accuracy, loss, training duration, optimizer efficacy, and issues encountered during training were used to examine the link between model complexity and performance.

4.1. Baseline vs Deeper Architecture:

To find out how more complexity affects overall performance, two CNN models—basic and advanced—were trained and examined.

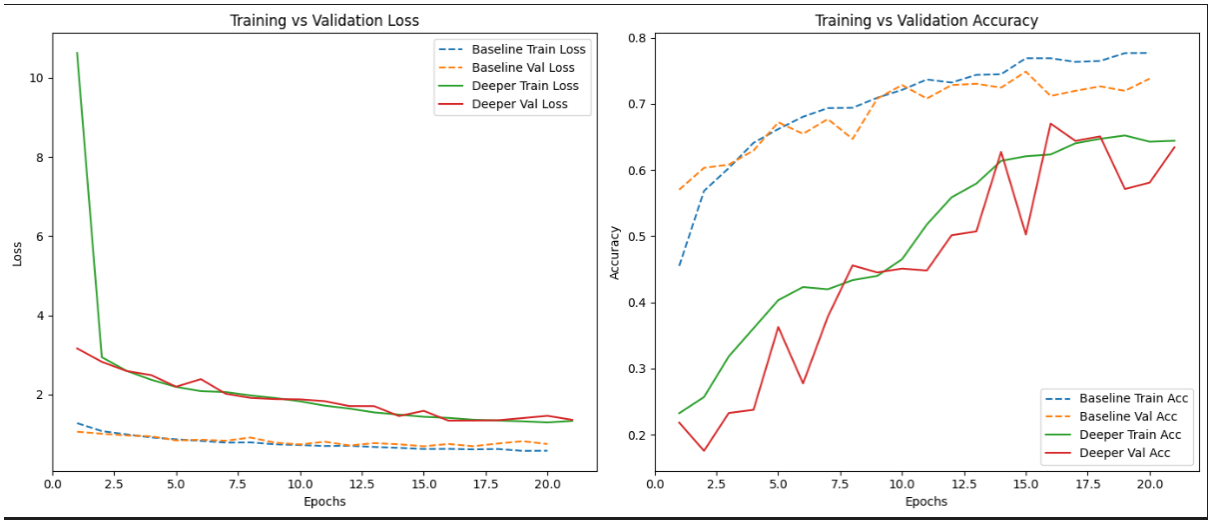

Figure 5 Comparison of training and validation loss

A baseline model and a deeper model's training and validation results over 21 epochs are displayed in the diagram. The deep model starts with a greater loss that gradually drops (i.e., with more variability in validation loss), which suggests potential instability or overfitting, despite the baseline model providing lower and more stable training and validation loss, according to the loss curves on the left. With smoother and more accurate curves, the accuracy plot on the right demonstrates that the baseline model consistently performs better than the deeper model in terms of both training and validation accuracy. Despite the higher capacity, the deeper model shows a wide variance in validation accuracy and modest development, indicating poorer generalization.

4.2. Computational efficiency:

The deeper CNN model's additional layers and parameters were designed to make it require more memory and training time than the baseline. Even while the baseline model trained quickly, the deeper model with a more sophisticated architecture showed longer epochs.

Training both models was accelerated by using Google Collab's GPU acceleration. The training procedure would have taken a lot longer without the usage of a GPU. Our

ability to train more complicated models and add more photos per batch was made easy by the GPU.

4.3. Training with different Optimizers:

The use of two optimizers that was used during the training of deeper model are Stochastic Gradient Descent (SGD) and Adam (Adaptive Moment Estimation).

```
print("Adam Optimizer - Best Validation Accuracy:", max(history_deeper.history['val_accuracy']))
print("SGD Optimizer  - Best Validation Accuracy:", max(history_sgd.history['val_accuracy']))
```

```
Adam Optimizer - Best Validation Accuracy: 0.67022305727005
SGD Optimizer  - Best Validation Accuracy: 0.22793404757976532
```

Figure 6 Training with different optimizer

Stochastic Gradient Descent (SGD) is an optimization algorithm that updates model weights using a single sample at a time, while Adam (Adaptive Moment Estimation) combines the benefits of both AdaGrad and RMSProp by using adaptive learning rates and momentum.

The image shows the best validation accuracy achieved by models using two different optimizers: Adam and SGD. The Adam optimizer significantly outperformed SGD, achieving a best validation accuracy of approximately 67%, compared to only 22% with SGD. This suggests that Adam's adaptive learning rate and momentum-based updates enabled faster and more effective convergence during training, whereas the SGD optimizer struggled to reach a comparable level of generalization on the validation set.

4.4. Challenges faced:

- Overfitting: The baseline model performed well to training examples but not generalizing well for validation sets- a sign of over-fitting. To alleviate this problem, dropout as well as batch normalization is added to the deeper model.
- Underfitting: When the model was training on earlier epochs with SGD, it gave it a problem for producing amazing patterns to the data.
- Training Stability: When fine-tuning the pre-trained model, it turned out to be crucial to reduce the learning rate to achieve the stability of training. The training procedure duration depended on the model's complexity and the applied algorithm of optimization. With the deeper model, singe-epoch training speed was slower

with Adam, but the general results were better. The ability to use Google Collab for GPU acceleration of the training made it much more effortless to handle CPU resources and lower the periods required for training.

5. Fine-Tuning or Transfer Learning:

The VGG16 model, whose features were trained using the massive ImageNet database, employed transfer learning. The majority of picture classification jobs benefit from VGG16's exceptional capacity to achieve high accuracy through its deep network architecture.

Our first step was to feature extraction by training only the new fully connected layers that were in charge of flower categorization and freezing the convolutional basis of VGG16. This method made it possible for the model to successfully apply what it had learned in ImageNet to flowers. In order for the model to improve its features and better fit the flower dataset, we then untied some of the lower convolutional levels to enable for fine-tuning. As a result, the model was able to modify the characteristics it recognizes, which improved its ability to recognize different kinds of floral photos.

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 7 Model summary of transfer learning

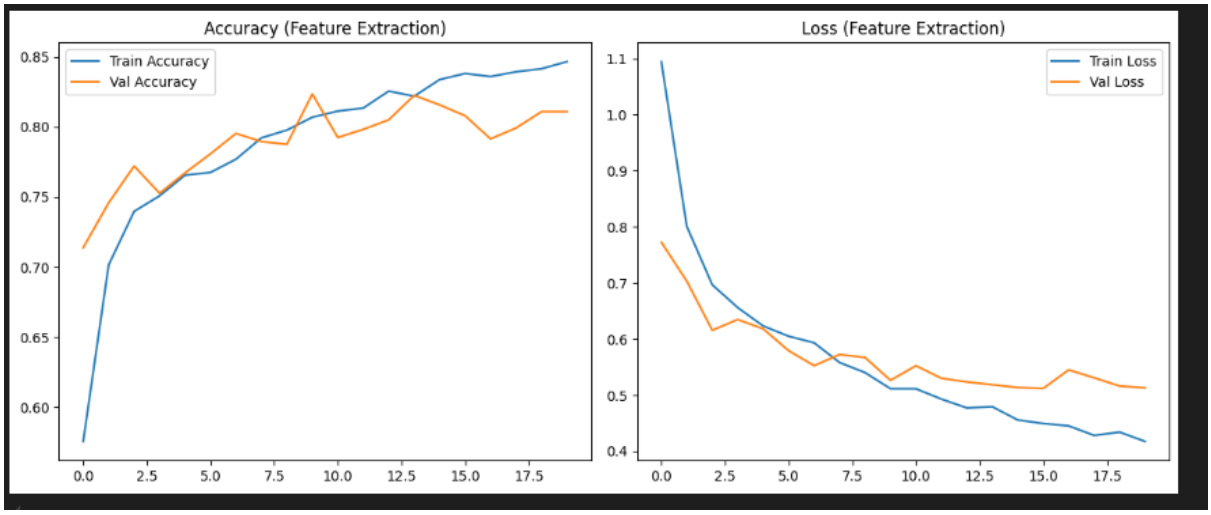

Figure 8 Training and validation accuracy

The first image depicts the architecture of a convolutional neural network (CNN) for image classification; it is probably over the VGG16 model; input size is equal to 180x180x3. It consists of several blocks of convolutional layers ending with the max pooling followed by fully connected (dense) layers. The model has roughly 21.27 million parameters in total where only around 6.56 million are trainable – usage of feature extraction (freezing of base layers). The second image shows the training and validation accuracy and loss for 20 epochs. The rate of the training accuracy continues to rise above 0.85, whereas the validation one stabilizes near 0.83 with small variations, thus, indicating slight overfitting. On the same note, training loss gets gradually smaller whereas, validation loss remains stable with certain lapses reinforcing the lack of severe overfitting but an effective generalization performance.

6. Conclusion:

Using a custom dataset, this experiment successfully showed that flower photos could be classified using a CNN. Particularly after Batch Normalization and Dropout were added to improve performance and prevent overfitting, the deeper CNN model performed admirably. When compared to SGD, the Adam optimizer showed better training behavior, and the trials were more effective because to Colab's GPU acceleration.

The dataset was also small and generated manually and it can limit the generalization of the model. In future work, we can consider incorporating proper baseline for comparison, data augmentation, and optimization of Hyperparameters with

automation tools, and transfer learning with VGG, ResNet, Inception, to enhance performance even further.

7. References

Rawat, W. a. W. Z., 2017. Deep Convolutional Neural Networks for Image Classification: A Comprehensive Review. *Neural Computation,* Volume 29, pp. 2352-2449.