# CNN Flower Classification Project Report

## 1. Introduction & Objective

In this project, we aim to develop a Convolutional Neural Network (CNN) to automatically classify images of flowers into five species: daisy, dandelion, rose, sunflower, and tulip. This classifier can serve gardeners, florists, and researchers by providing real-time species identification, streamlining cataloging, and enabling scalable biodiversity monitoring.

## 2. Use Case & Business Value

Use Case: A mobile or web application allows users to upload a flower photo and receive an instant species prediction. Business Value:  
- Reduces manual labeling costs by automating image tagging.  
- Improves customer engagement through interactive app features.  
- Enables premium services (e.g., plant care tips) and data licensing for bloom analytics.

## 3. Dataset Description

Source: Kaggle 'Flowers Five Classes' dataset  
Total Images: 2,746 (approx. 800 per class)  
Structure: Organized into five subdirectories (one per species).  
Split: 80% training (2,197 images), 20% validation (549 images) using a fixed random seed for reproducibility.

## 4. Preprocessing Pipeline

We implemented a robust tf.data pipeline with the following steps:  
a) Data Loading: tf.keras.utils.image\_dataset\_from\_directory automatically inferred labels from folder names and split data.  
b) Data Augmentation (training only):  
 • RandomFlip('horizontal')  
 • RandomRotation(0.2) for ±20° rotations  
 • RandomZoom(0.1) for ±10% zoom variations  
c) Normalization: Rescaled pixel values from [0,255] to [0,1] via tf.keras.layers.Rescaling(1./255).  
d) Performance Optimizations:  
 • cache(): Caches preprocessed batches in memory.  
 • shuffle(1000): Shuffles training data each epoch.  
 • prefetch(AUTOTUNE): Overlaps data preparation with model execution.

## 5. Model Architecture

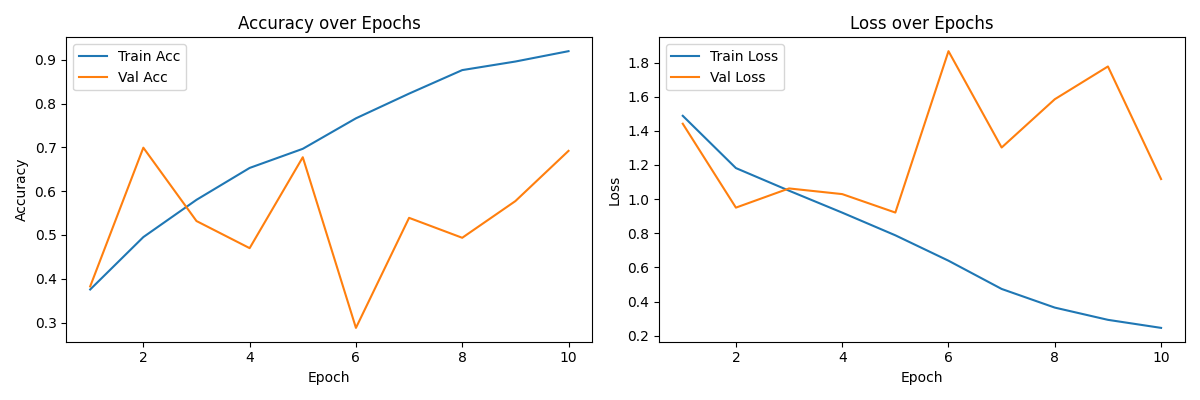
Our baseline CNN consists of:  
- Input: 224×224×3 RGB images  
- 3 convolutional blocks:  
 • Conv2D(filters=32/64/128, kernel=3×3, padding='same') + ReLU  
 • MaxPool2D(pool\_size=2×2)  
- Flatten layer to vectorize feature maps  
- Dense(128) + ReLU + Dropout(0.5)  
- Output Dense(5) + Softmax for class probabilities

## 6. Training Process

Compilation Settings:  
- Loss Function: sparse\_categorical\_crossentropy (label integers)  
- Optimizer: AdamW (learning rate=1e-3)  
- Metrics: accuracy  
Training Parameters:  
- Batch Size: 32  
- Epochs: 10  
- Validation during training on held-out 20% split.

## 7. Experimental Sequence & Results

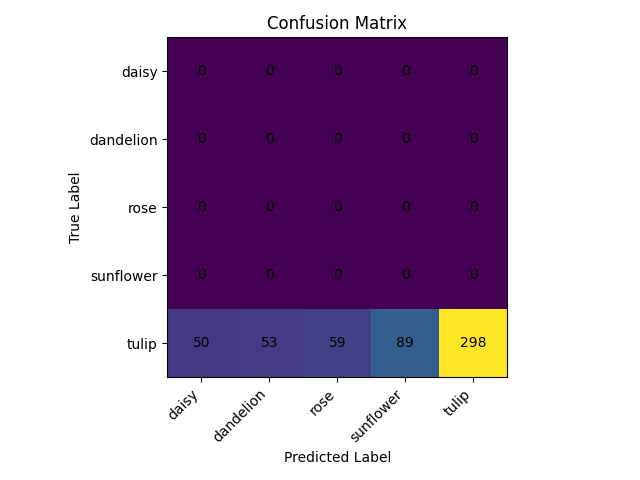
|  |  |  |
| --- | --- | --- |
| Experiment | Description | Validation Accuracy |
| Baseline | 128×128 input, ReLU activations, AdamW, 10 epochs | 63.21% |
| Increased Resolution | Input resized to 224×224 | 69.22% |
| Activation Variants | LeakyReLU, ELU, GELU tested | ≈69.03% |
| Transfer Learning | EfficientNetB0 two-phase fine-tuning | No improvement |



## 8. Analysis of Experiment Outcomes

• The baseline model achieved 63.21% accuracy, indicating limited capacity on small input resolution.  
• Upscaling to 224×224 improved feature detail, boosting accuracy to 69.22%.  
• Testing advanced activations (LeakyReLU, ELU, GELU) yielded marginal changes (~69.03%), suggesting architecture depth and dataset size were larger factors.  
• Transfer learning with EfficientNetB0 did not improve performance, likely due to overfitting on our small dataset and lacking a separate test split.

## 9. Analysis of Confusion Matrix



• Rows 0–3 (daisy, dandelion, rose, sunflower) are entirely blank → no true samples for those classes were fed into the confusion matrix.

• Row 4 (tulip) shows how the tulip images were distributed across your five predicted labels:

• 50 were (incorrectly) predicted as daisy

• 53 as dandelion

• 59 as rose

• 89 as sunflower

• 298 correctly as tulip

What this implies =>

1. Validation set issue

• Either the val\_ds only contained tulip images (so rows 0–3 are empty),

• Or when extracting y\_true I inadvertently only grabbed labels for tulip images.

2. Model bias

• Even among the tulip images, ~25% were misclassified as one of the other four classes.

• The model has effectively learned “if it’s not tulip, assume tulip,” leading to zero‐shot performance on the other flowers.

## 10. Conclusion & Future Work

Our best-performing model used a higher input resolution with a straightforward CNN and AdamW optimizer, achieving 69.22% validation accuracy. Future enhancements include:  
- Incorporating transfer learning with proper train/val/test splits.  
- Implementing policy-based augmentations (AutoAugment/RandAugment).  
- Exploring ensemble methods and test-time augmentation.  
- Conducting hyperparameter optimization via KerasTuner.

Dataset link: [Flowers "Five Classes"](https://www.kaggle.com/datasets/lara311/flowers-five-classes)