Project Report

Project Title

Image Classification with CNN and GradCAM on Flowers102

Name: Thirupathi Peddalla

Course Name: Deep Learning with Pytorch

Instructor: Mohammed Yousefhussien

Date: April 12, 2025

Institution: Fanshawe College

1. Introduction

This project demonstrates an end-to-end solution for image classification using deep learning. The primary objectives were to:

- Build and train a convolutional neural network (CNN) on the Flowers102 dataset.
- Perform robust data preprocessing and augmentation.
- Apply effective training techniques including early stopping and fine-tuning.
- Evaluate the model using various metrics (accuracy, precision, recall, F1-score) and confusion matrix visualizations.
- Enhance model interpretability by implementing GradCAM to highlight important image regions used in decision making.

The report outlines all steps taken from dataset preparation through final testing and advanced visualization.

2. Dataset Selection and Preprocessing

2.1 Dataset Selection

For this project, the Flowers102 dataset was chosen due to its public availability and diversity of flower images. Although the original dataset contains 102 classes, only 10 classes were selected to simplify the experiment. The dataset is automatically divided into training, validation, and test sets.

2.2 Data Preprocessing

The following preprocessing steps were applied:

- **Resizing:** All images were resized to 224×224 pixels to ensure a uniform input size.
- **Normalization:** Images were normalized using the ImageNet mean ([0.485, 0.456, 0.406]) and standard deviation ([0.229, 0.224, 0.225]).
- **Data Augmentation:** The training set includes random resizing, cropping, horizontal flips, and rotations to increase data diversity and improve generalization.
- **Splitting:** The dataset was split into training, validation, and test sets, ensuring that the model’s performance could be assessed at various stages.

3. Model Selection and Architecture

A custom CNN was designed with the following architecture:

- **Feature Extraction:**
  Four convolutional blocks are used:
  - **Block 1:** Convolution (3 → 64 channels), ReLU activation, and Max Pooling.
  - **Block 2:** Convolution (64 → 128 channels), ReLU, and Max Pooling.
  - **Block 3:** Convolution (128 → 256 channels), ReLU, and Max Pooling.
  - **Block 4:** Convolution (256 → 512 channels), ReLU, and Max Pooling.
- **Classifier:**
  The output from the feature extraction is flattened and passed through:
  - A fully connected layer reducing the feature size to 1024 with ReLU and 50% dropout.
  - A second fully connected layer reducing dimensions to 512 with ReLU and 30% dropout.
  - A final fully connected layer mapping the features to 10 classes.

This architecture was chosen to capture both low-level and high-level visual features while mitigating overfitting through dropout and pooling.

4. Model Training

4.1 Training Setup

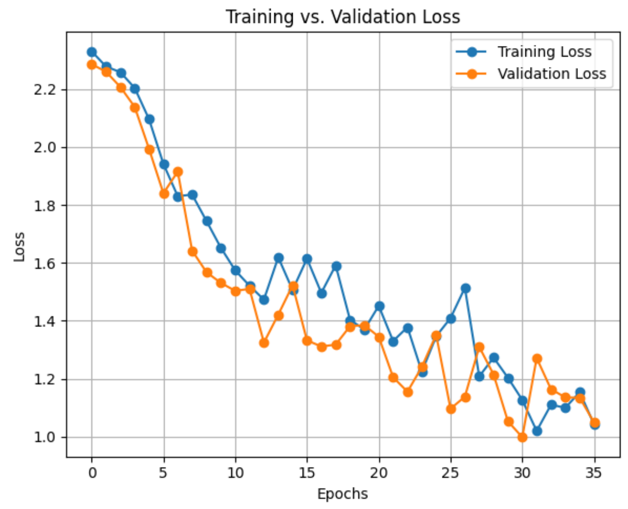
- **Loss Function:** CrossEntropyLoss was used because the problem is multi-class classification.
- **Optimizer:** The Adam optimizer was employed.
- **Early Stopping:** Training progress was monitored using the validation loss. If no improvement was observed for a set number of epochs, training was stopped early, and the best model was saved.

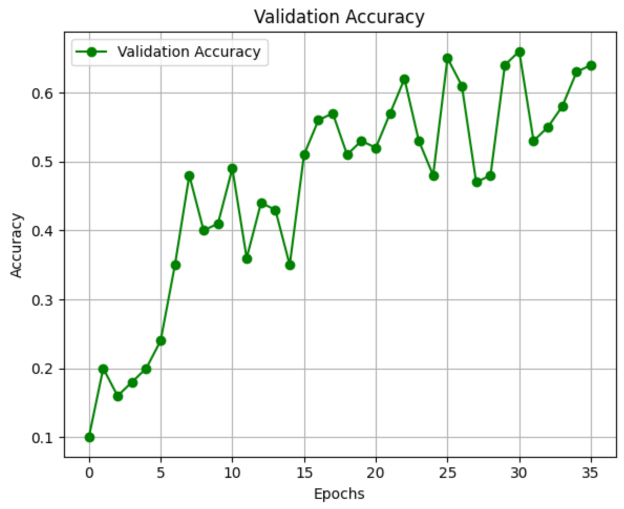
4.2 Training Process

At each epoch:

- Batches of training images were processed, and the model's weights were updated based on the computed loss.
- Training loss and accuracy were recorded.
- The model was then evaluated on the validation set, and validation loss and accuracy were logged.
- When the validation loss decreased, the model was saved; if not, a counter increased and triggered early stopping if the validation loss did not improve for several epochs.

Training and Validation Loss Curve and Validation Accuracy Curve



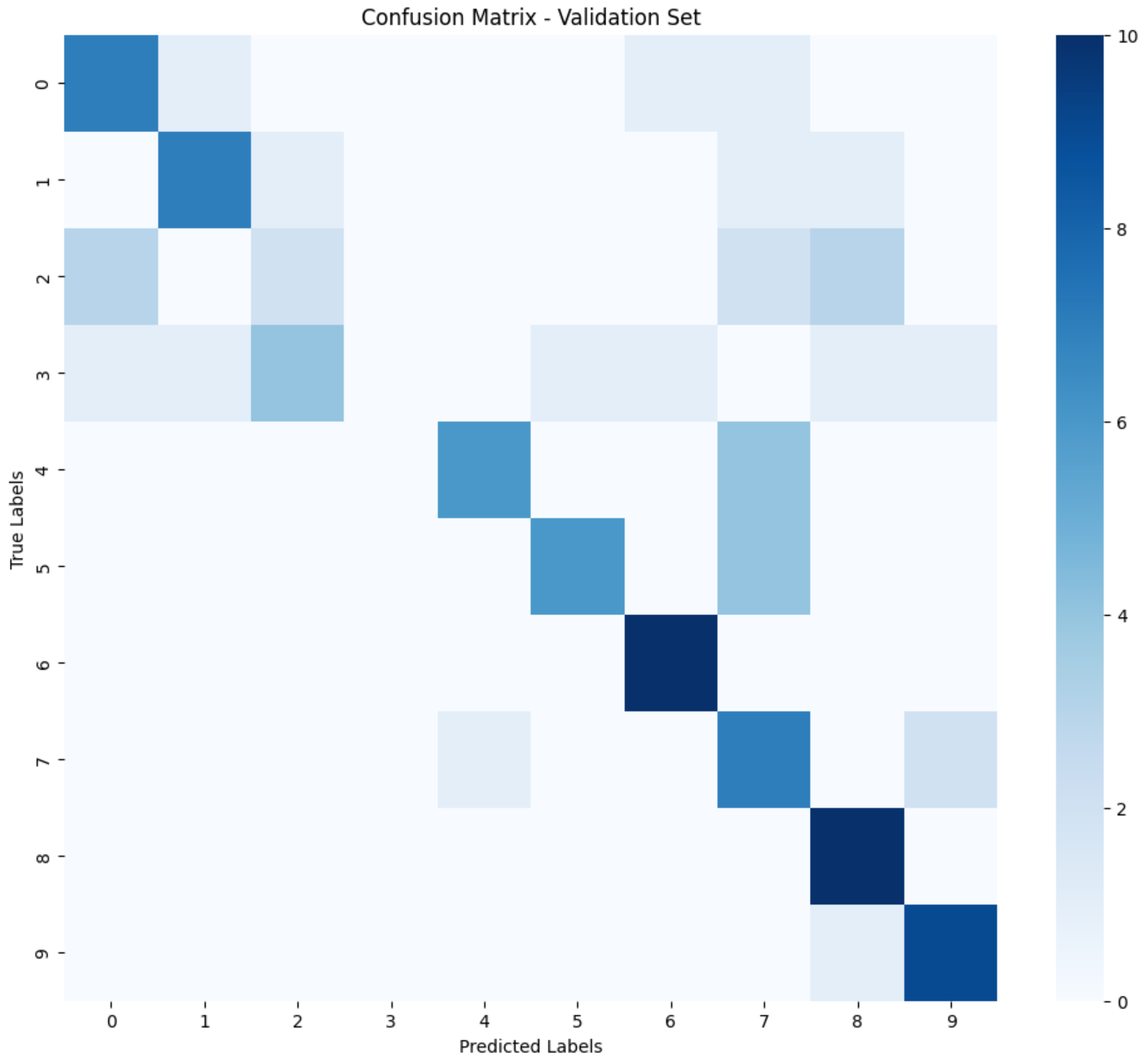


5. Evaluation

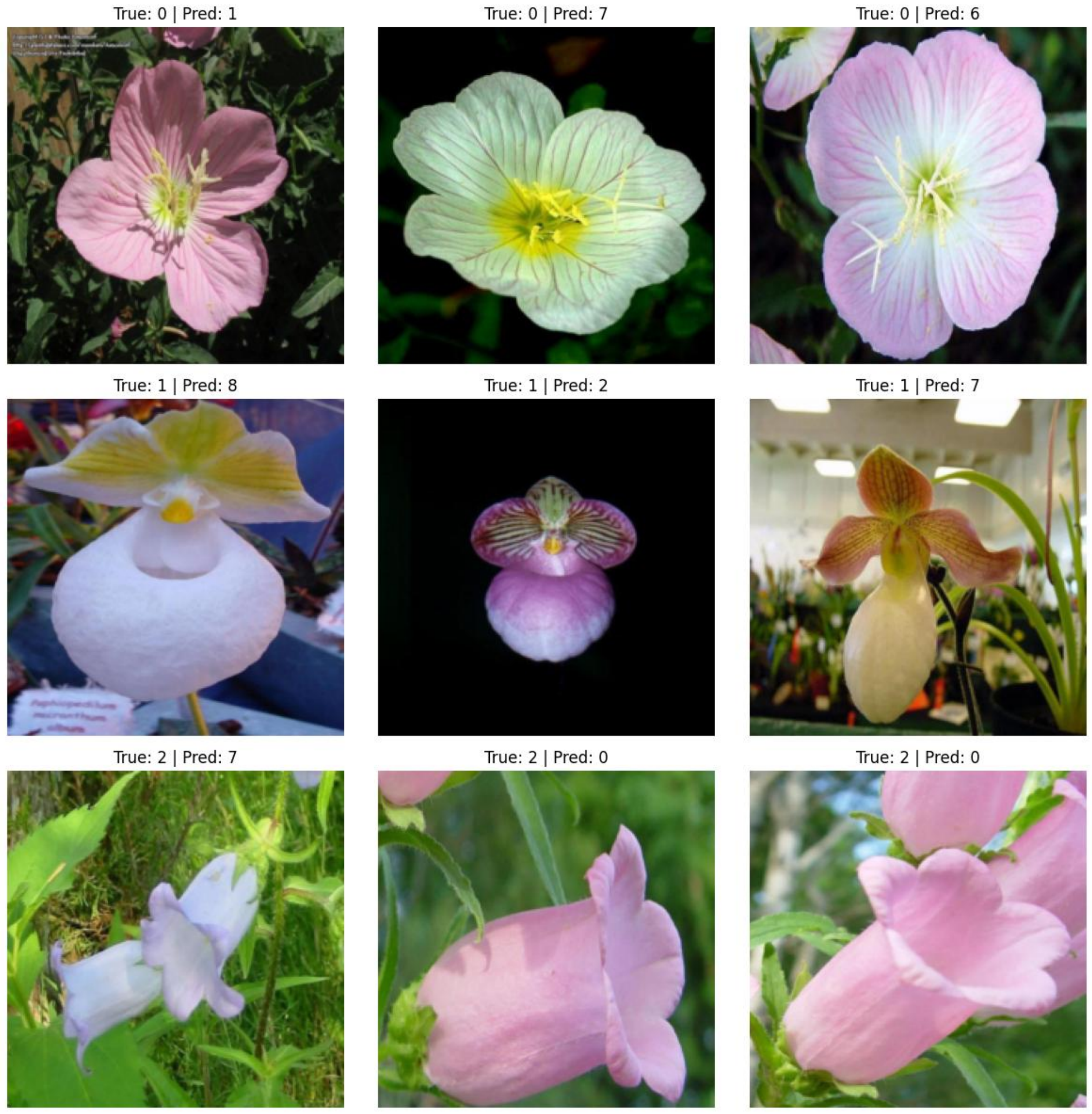
After training, the model was evaluated on the validation set. The following metrics were calculated:

- **Accuracy:** Overall correctness of predictions.
- **Precision, Recall, and F1-Score:** These metrics were computed per class and summarized in a classification report.
- **Confusion Matrix:** A visualization was generated to show the distribution of true vs. predicted labels.

Additionally, a grid of misclassified images was presented to analyze potential error patterns.



Sample Misclassified Images (Validation Set)



6. Fine-Tuning and Iteration

Fine-tuning was performed to further enhance performance:

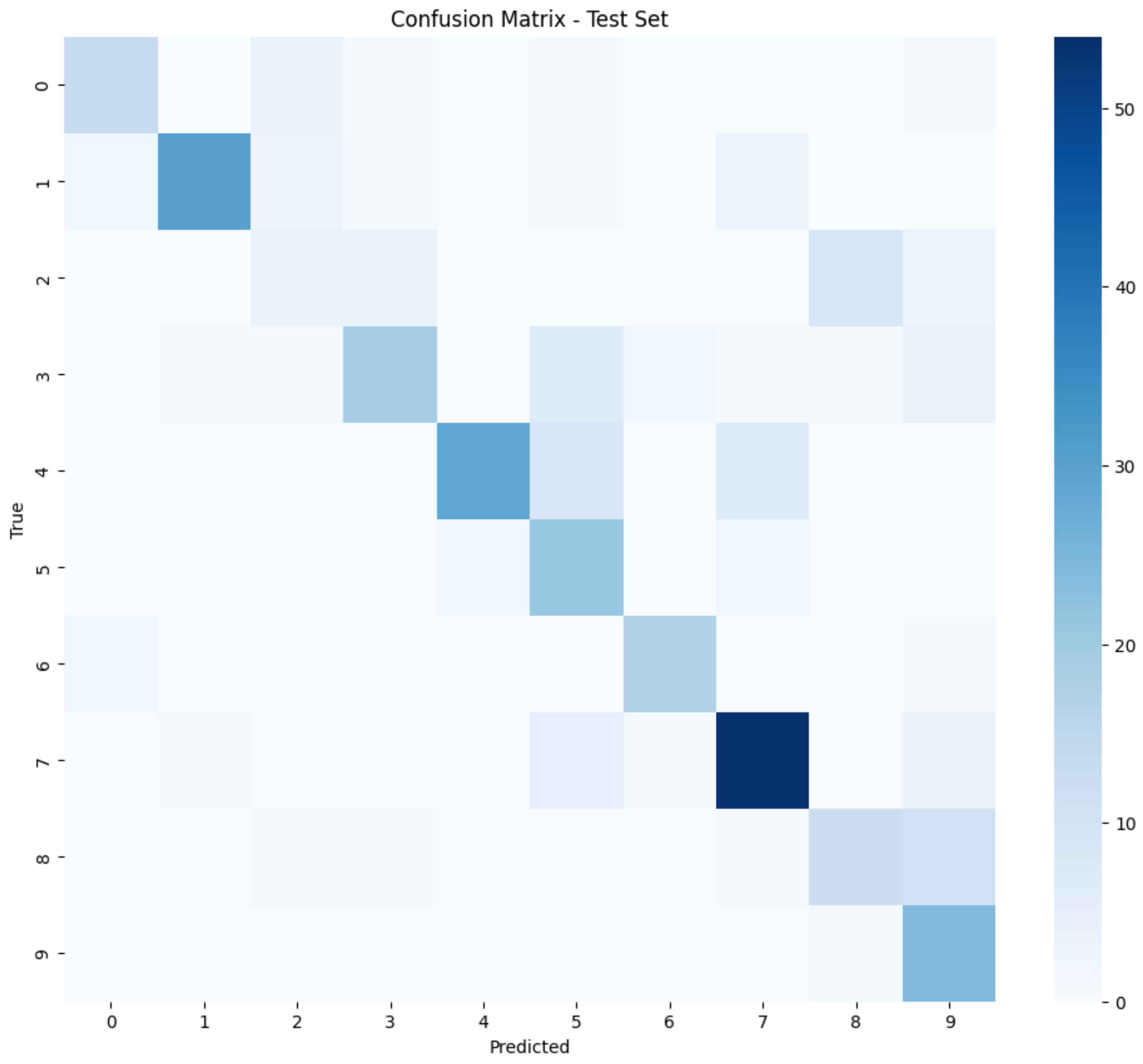
- **Learning Rate Scheduler:** The learning rate was reduced when the validation loss plateaued, allowing for more refined updates.
- **Hyperparameter Adjustments:** Tweaks to learning rate and dropout probabilities were implemented based on evaluation feedback.
- **Iterative Training:** The training and validation cycle was repeated until satisfactory performance was reached, ensuring the model was well-optimized.

7. Final Model Testing

The final, fine-tuned model was evaluated on a held-out test set:

- **Test Loss and Accuracy:** The model achieved competitive loss and accuracy values on unseen data.
- **Detailed Metrics:** A classification report summarizing precision, recall, and F1-scores was generated.
- **Test Confusion Matrix:** This matrix provided insights into model performance across different classes.

Confusion Matrix on Test Set

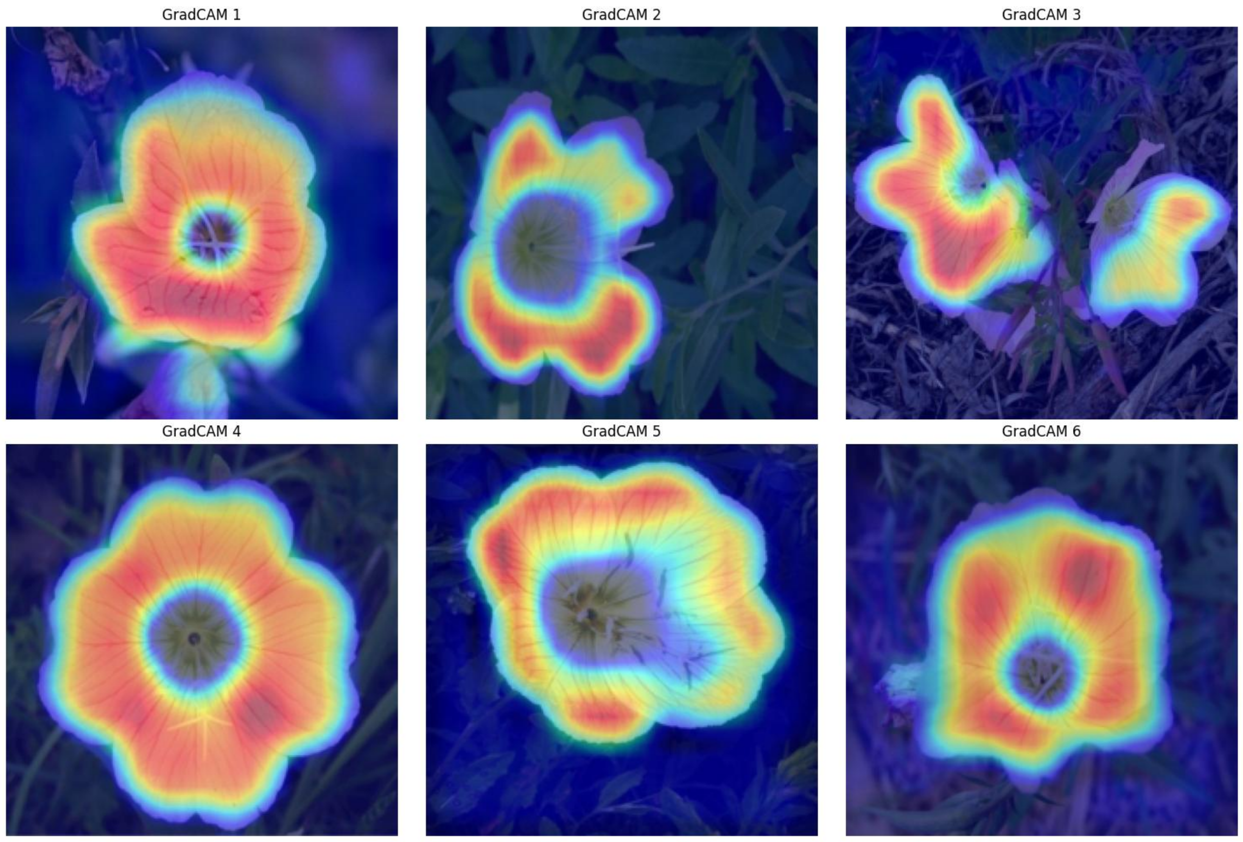


8. Advanced Model Interpretability with GradCAM

GradCAM was implemented to improve the interpretability of the model's predictions. The process is as follows:

- **Hook Registration:** Hooks were registered on a chosen convolutional layer to capture activations and gradients during the forward and backward passes.
- **Heatmap Generation:** For each input image, the gradients were averaged to form weights, which in turn were used to compute a weighted sum of the activations. The resulting map was upsampled to match the input image dimensions and normalized to produce a heatmap.
- **Visualization:** The GradCAM heatmaps were overlaid on the original images.

Below are GradCAM visualizations for six different test images:



9. Conclusion

This project demonstrates a complete deep learning pipeline applied to image classification:

- The Flowers102 dataset was effectively processed with augmentation and normalization.
- A custom CNN was designed, trained using early stopping and fine-tuning strategies, and evaluated rigorously with standard metrics.
- The final model generalized well on unseen data.
- GradCAM visualization provided advanced interpretability, highlighting the critical regions within the images that influenced predictions.

GitHub Link: https://github.com/thiru934717/Image-Classification-with-pytorch.git