Deep Learning (CS F425) 2024- 2025

Project Task 2

Flower Classification

Under supervision of: Dr. Bharat Richhariya

NISHANT SHUBHAM(2023H1120196P)
Ankit Kumar (2023H1030076P)
Abhay Tripathi (2023H1120190P)

Chapters

Introduction

Image classification is a very fundamental problem in real life. Thousands of images are generated every second, and classifying them is a big task.
In this project, we aim to classify various species of flower images using a CNN network. Flowers, with their diverse appearances in color, shape, and texture, offer a challenging yet rewarding domain for image classification.

Our dataset consists of thousands of labeled images of flowers of different categories representing different flower species. Using PyTorch, we train a deep neural network to identify flower types.

To monitor and optimize the training process, integrate weight and biases for experiment tracking hyperparameter tuning visualization of metrics.

Model Architecture

1) <u>Custom ResNet18 Model Architecture fully :</u>

Here full architecture of ResNet18l are used for training and validation of model accuracy

Architecture component

- Input Dimensions
- Convolutional Layers:
    - 7 x 7 kernels with stride 2 and padding 3.
    - 64 output channels.
- Batch Normalization is used.
- The ReLU Activation function is used to introduce non-linearity for feature extraction.
- Max Pooling.

This Architecture contains four main residual blocks for deeper feature extraction.

- Block 2: It maintains spatial dimensions at 56 x 56.
- Block 3: It reduces spatial dimension to 28 x 28 with a stride of 2 and increasing depth to 128
- Block 4: It further reduces dimensions to 14 x 14 and increases depth to 256.
- Block 5: The final block reduces dimensions to 7 x 7 and increases depth to 512.
    - Also, each block has two convolution layers with 3 x 3 kernels and batch normalization for each convolutional layer.

Final layers:

- Global Average pooling reduces the 7 x 7 x 512 output from the final block to a vector.
- Fully connected layers
    - A linear map of 512 features to 1000 intermediate neurons.
    - The final linear layer maps 1000 features to the number of output class(number of output class=60)

Input dataset features:

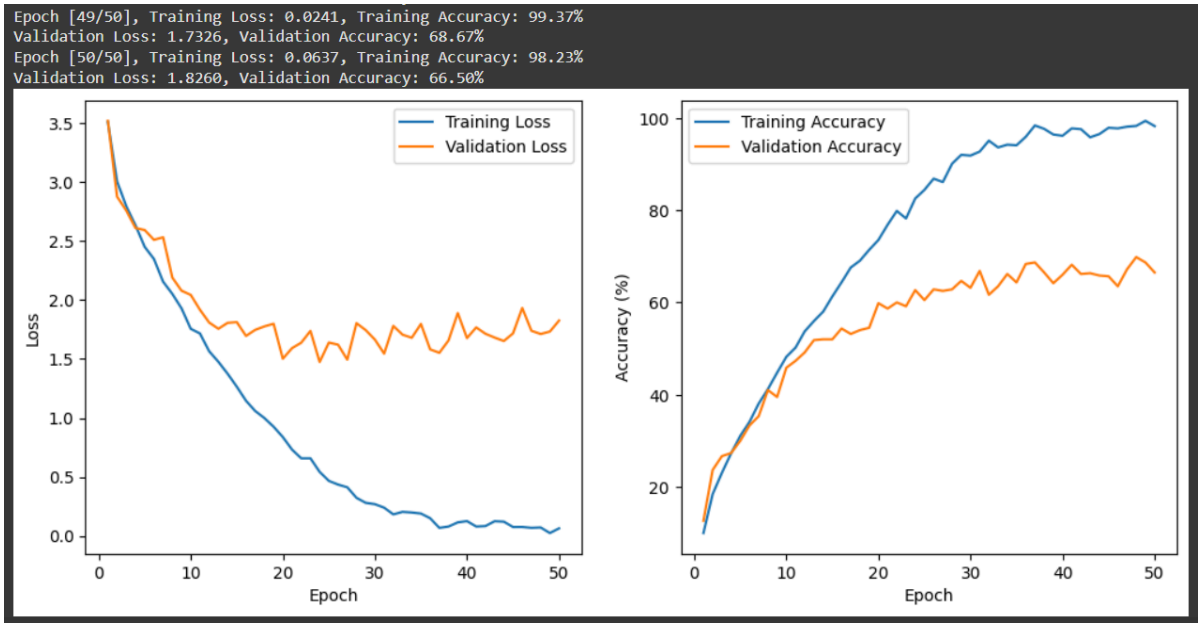
For training the model, 3000 images of 60 different class of size 256 x 256 x3 and corresponding labels are given in a .txt file.
For testing the model, 600 images of 60 different classes of size 256 x 256 x3 and corresponding labels are given in the .txt file.

Learning rate=0.001

epoch=50

Output Result:

Validation accuracy:66.50%

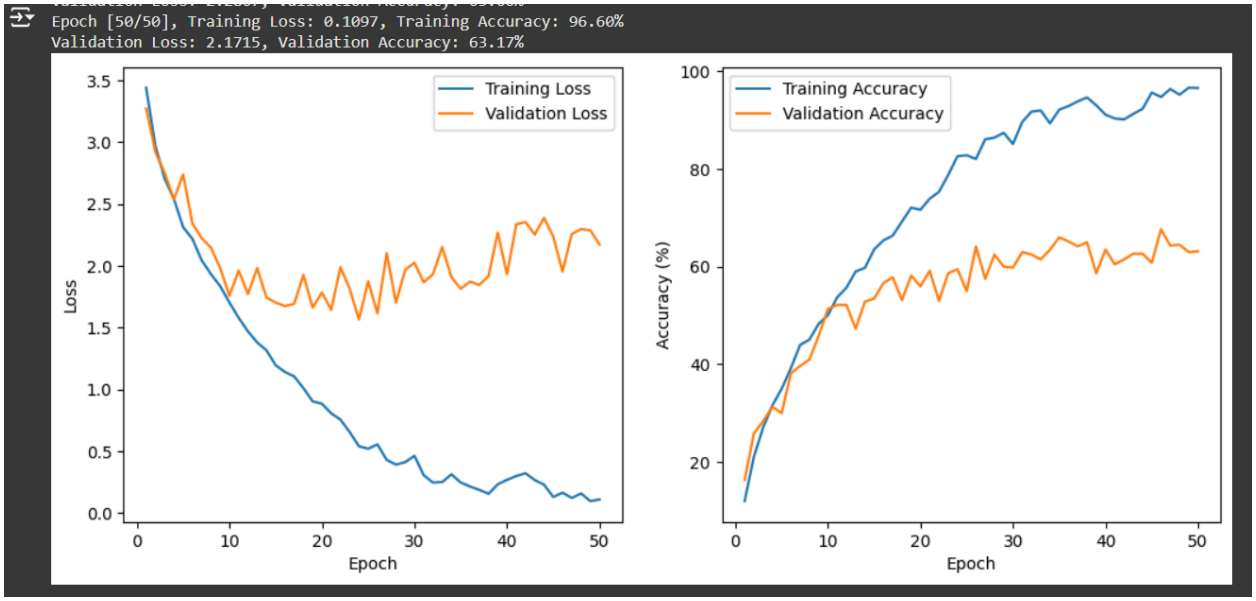



```
Epoch [1/50], Training Loss: 3.5087, Training Accuracy: 10.00%Validation
Loss: 3.5197, Validation Accuracy: 12.67%

Epoch [10/50], Training Loss: 1.7562, Training Accuracy: 48.20%Validation
Loss: 2.0435, Validation Accuracy: 45.83%

Epoch [20/50], Training Loss: 0.8381, Training Accuracy: 73.53%Validation
Loss: 1.5024, Validation Accuracy: 59.83%

Epoch [30/50], Training Loss: 0.2689, Training Accuracy: 91.83%Validation
Loss: 1.6644, Validation Accuracy: 63.17%

Epoch [40/50], Training Loss: 0.1254, Training Accuracy: 96.13%Validation
Loss: 1.6778, Validation Accuracy: 66.00%

Epoch [50/50], Training Loss: 0.0637, Training Accuracy: 98.23%Validation
Loss: 1.8260, Validation Accuracy: 66.50%
```

- After changing the activation function in the above ResNet18 Architecture from **ReLU** to **elu,** training the model on the same parameter

Result Output :

Validation accuracy: 63.17



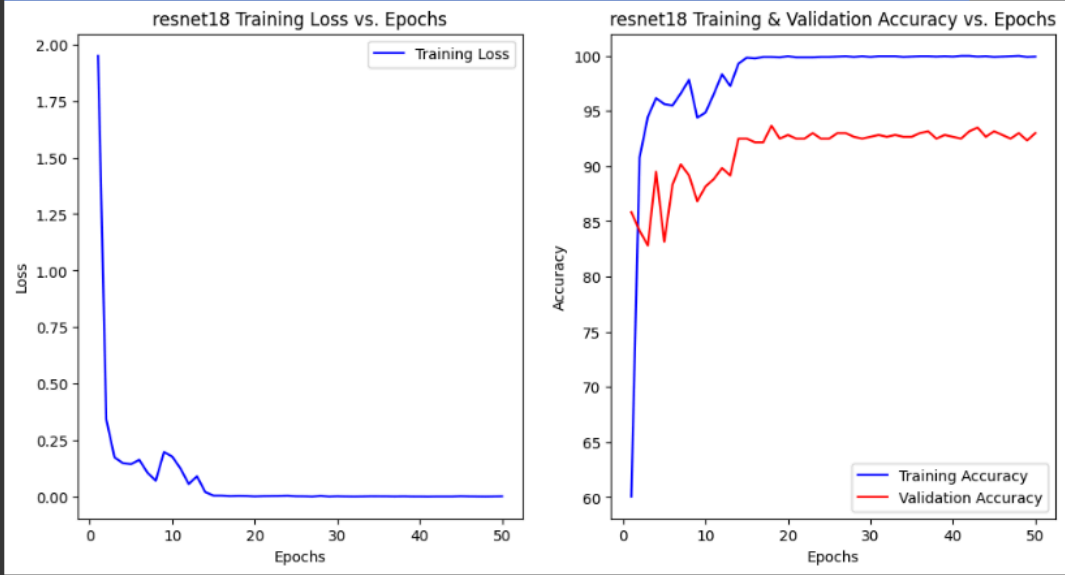
However, the validation accuracy result only improves a little.

2) ResNet18 pre-trained fine-tune model:

In this, the ResNet18 model is finetuned with an adaptive learning rate.

Data augmentation is used to improve validation accuracy.

```
Validation Accuracy: 92.33%
Current learning rate: 1.0000000000000004e-08
Epoch [50/50], Loss: 0.0032, Train Accuracy: 99.93%
Validation Accuracy: 93.00%
Current learning rate: 1.0000000000000004e-08
ResNet18 model weights saved to resnet18_with_augmentation_model_weights.pth
```

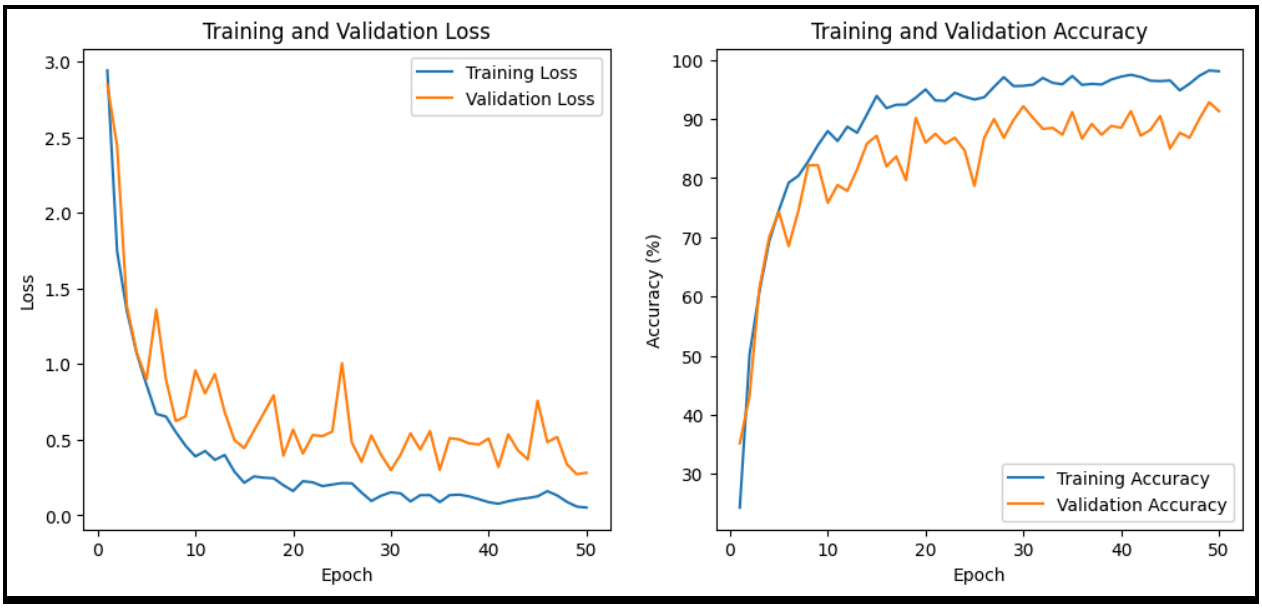



```
Epoch [1/50], Loss: 1.9489, Train Accuracy: 60.07%
Validation Accuracy: 85.83%
Current learning rate: 0.01
Epoch [2/50], Loss: 0.3433, Train Accuracy: 90.77%
Validation Accuracy: 84.17%
Current learning rate: 0.01
Epoch [3/50], Loss: 0.1751, Train Accuracy: 94.47%
Validation Accuracy: 82.83%
Current learning rate: 0.01
Epoch [4/50], Loss: 0.1500, Train Accuracy: 96.17%
Validation Accuracy: 89.50%
Current learning rate: 0.01
Epoch [5/50], Loss: 0.1454, Train Accuracy: 95.63%
Validation Accuracy: 83.17%
Current learning rate: 0.01
Epoch [15/50], Loss: 0.0064, Train Accuracy: 99.83%
Validation Accuracy: 92.50%
Current learning rate: 0.001
Epoch [30/50], Loss: 0.0035, Train Accuracy: 99.90%
Validation Accuracy: 92.67%
```

```
Current learning rate: 1e-05
Epoch [40/50], Loss: 0.0023, Train Accuracy: 99.93%
Validation Accuracy: 92.67%
Current learning rate: 1.0000000000000002e-06
Epoch [50/50], Loss: 0.0032, Train Accuracy: 99.93%
Validation Accuracy: 93.00%
Current learning rate: 1.0000000000000004e-08
```

3) <u>ResNet50 pre-trained model with data augmentation:</u>

Validation Accuracy: 91.33%

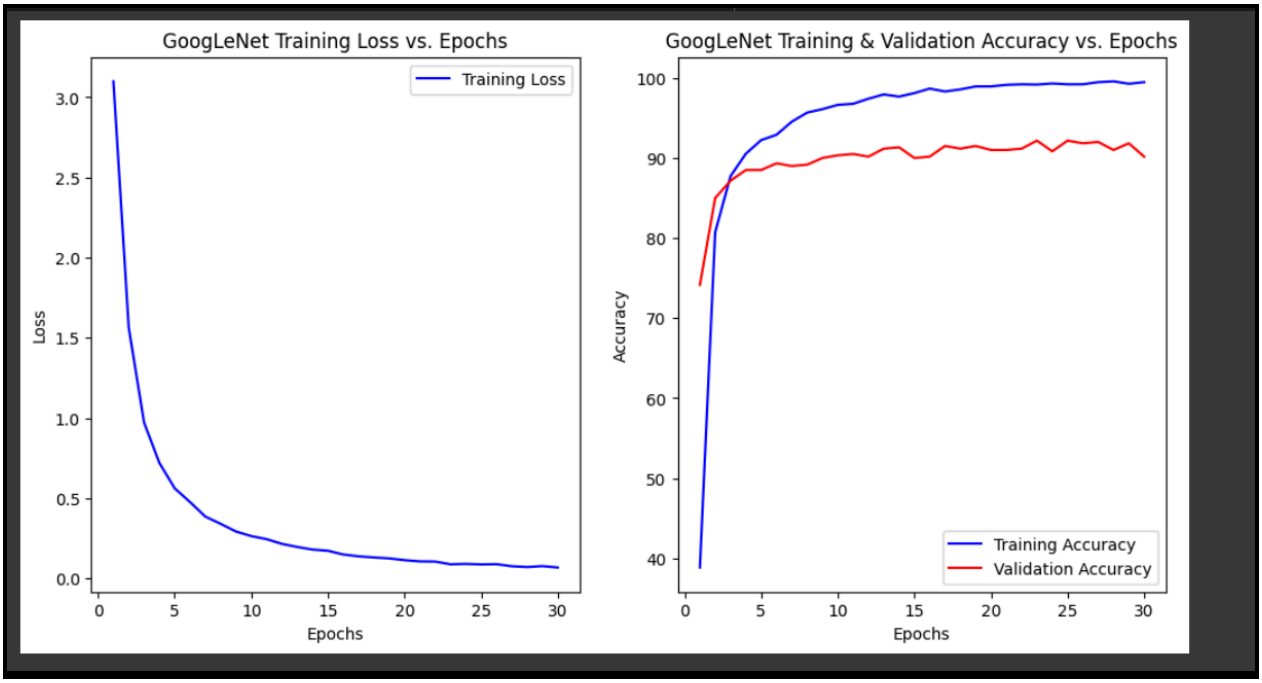



```
Downloading: "https://download.pytorch.org/models/resnet50-0676ba61.pth"
to /root/.cache/torch/hub/checkpoints/resnet50-0676ba61.pth
100%|██████████| 97.8M/97.8M [00:00<00:00, 106MB/s]
Epoch [1/50], Training Loss: 2.9414, Training Accuracy: 24.30%
Validation Loss: 2.8493, Validation Accuracy: 35.17%

Epoch [10/50], Training Loss: 0.3901, Training Accuracy: 87.97%
Validation Loss: 0.9587, Validation Accuracy: 75.83%

Epoch [20/50], Training Loss: 0.1613, Training Accuracy: 95.00%
Validation Loss: 0.5671, Validation Accuracy: 86.00%

Epoch [30/50], Training Loss: 0.1528, Training Accuracy: 95.60%
Validation Loss: 0.2997, Validation Accuracy: 92.17%

Epoch [40/50], Training Loss: 0.0875, Training Accuracy: 97.17%
```

```
Validation Loss: 0.5088, Validation Accuracy: 88.50%

Epoch [50/50], Training Loss: 0.0517, Training Accuracy: 98.07%
Validation Loss: 0.2816, Validation Accuracy: 91.33%
```

4) <u>GoogLeNet model fine-tuned with early stopping:</u>

Learning rate 0.001
number_epoch=50
Validation Accuracy: 90.0%
Early stopping occurs at epoch 30




```
------------ Googenet fine-tuned with early stopping
Epoch [1/50], Loss: 3.0993, Train Accuracy: 38.87%
Validation Accuracy: 74.17%
Epoch [2/50], Loss: 1.5645, Train Accuracy: 80.73%
Validation Accuracy: 85.00%
Epoch [3/50], Loss: 0.9719, Train Accuracy: 87.73%
Validation Accuracy: 87.17%
Epoch [4/50], Loss: 0.7197, Train Accuracy: 90.53%
Validation Accuracy: 88.50%
Epoch [5/50], Loss: 0.5613, Train Accuracy: 92.23%
Validation Accuracy: 88.50%
Epoch [15/50], Loss: 0.1718, Train Accuracy: 98.10%
```

```
Validation Accuracy: 90.00%
Epoch [20/50], Loss: 0.1133, Train Accuracy: 98.93%
Validation Accuracy: 91.00%
Epoch [30/50], Loss: 0.0674, Train Accuracy: 99.47%
Validation Accuracy: 90.17%
Early stopping at epoch 30
```

Design Choices

The final architecture we used was Fine Tuned ResNet18 with data augmentation and ReduceLROnPlateau as a Learning Rate Scheduler, which gave a validation accuracy of 93%. We tried other pre-trained models like VGGNet, GoogLeNet, EfficientNet, ResNet-50, and ResNet18 and, based on accuracy, went ahead with ResNet18. We started with a constant learning rate and tried various values. We went ahead with the Learning Rate Scheduler, ReduceLROnPlateau, and StepLR and finally decided on ReduceLROnPlateau based on performance.

We also augmented the image dataset and saw an increase in accuracy. We tried various Regularization techniques like Weight Decay and Dropout, but the results were unfavorable. We also trained a Custom CNN from scratch with architecture similar to ResNet18 and tried various combinations, like modifying the loss function, but the accuracy could have been better.

Result Accuracy

	Model Architecture type	Learning_rate	Batch_size	Number of Epoch	Training loss	Training accuracy	Validation accuracy
1	Custom ResNet18 architecture	0.001	32	50	0.0637	98.23%	66.50 %
2	Custom ResNet18 architecture with ELU	0.001	32	50	0.1097	96.6%	63.17%
3	RestNet18 pretrained finetune	0.001	32	50	0.0032	99.93%	93.00%
4	ResNet50 pre-trained	0.001	32	50	0.057	98.07%	91.33%
5	GoogLeNet model fine-tuned with early stopping	0.001	32	50	0.067	99.47%	90%

Conclusion

In this flower classification task, multiple experiments with different architectures for image classification, including **ResNet18**, **ResNet50**, **GoogLeNet,** and ResNet18 with fine-tuning and data augmentation, consistently delivered better performance than others.

Fine-tuning helped adapt the pre-trained model to the specific dataset and enhanced its generalization ability. Data augmentation further contributes by artificially creating more datasets and making the model more robust.