IMLO Project Report

Y3912791

***Abstract*—The objective of this project is to classify flower species using the Oxford 102 Flower dataset, a well-known dataset for flower classification. A neural network classifier was constructed and trained from scratch using PyTorch [7], with a custom architecture that did not leverage any pre-trained models. The model was trained on the official train set, validated on the validation set, and tested on the test set of the flowers-102 dataset [4]. Data augmentation techniques were employed to enhance model robustness. The final model achieved a validation accuracy of 67.35% and a test accuracy of 64.22%, thereby demonstrating its effectiveness in flower species categorisation.**

I. Introduction

The image classification is a very fundamental task in computer vision, involving with classification of images into pre-defined classes. It has a wide range of using areas, including autonomous vehicles, medical imaging, security systems, and social media. Historically, various techniques have been employed to tackle image classification, such as manual feature extraction followed by traditional machine learning algorithms like Support Vector Machines (SVMs) and K-Nearest Neighbours (k-NN) [6].

The advent of deep learning has accelerated the rise of convolutional neural networks (CNNs) as the de facto standard for image classification tasks. Popularised by the AlexNet architecture in 2012 [1], CNNs have demonstrated remarkable performance through the automatic learning of hierarchical features from raw image data. Subsequent architectures such as VGGNet [2], ResNet [3] and Inception have further improved the limits of accuracy and efficiency in image classification.

The aim of this project is the construction and training of a convolutional neural network classifier to categories images of flowers from the Oxford 102 Flower dataset. This data set is a common benchmark in the field, containing images of 102 different flower species [4].A significant challenge in this project is the development of a model that is able to differentiate between the 102 different species of flower, taking into account various factors that influence flower appearance, such as lighting, the background and occlusion.

A deep learning approach utilising PyTorch [7] is employed to design and implement a bespoke neural network architecture. In contrast to pre-trained models, this network is constructed from the ground up, ensuring that it learns features specific to the Flowers-102 dataset [4]. The methodology incorporates data augmentation techniques [8] to enhance the model's robustness and performance. The network is trained, validated, and tested on the official splits of the dataset, and the results are analysed to evaluate its effectiveness in flower species classification.

II. Method

This section describes the neural network architecture employed for classifying the Oxford 102 Flower dataset, the loss function utilized, and the training procedure adopted.

A. Network Architecture

The proposed architecture is a Convolutional Neural Network (CNN) consisting of several convolutional layers, batch normalization, ReLU activation functions, pooling layers, and fully connected layers. The details of the architecture are illustrated in Figure 1:

- **Input Layer:** The network takes an RGB image of size $3 \times 224 \times 224$ as input.
- **Convolutional Layers:** The network includes seven convolutional layers with increasing depth, each followed by batch normalization and ReLU activation, with max-pooling layers to reduce spatial dimensions:
  - **Layer 1:** 16 filters, batch normalization, ReLU activation, max-pooling.
  - **Layer 2:** 32 filters, batch normalization, ReLU activation, max-pooling.
  - **Layer 3:** 64 filters, batch normalization, ReLU activation, max-pooling.
  - **Layer 4:** 128 filters, batch normalization, ReLU activation, max-pooling.
  - **Layer 5:** 256 filters, batch normalization, ReLU activation, max-pooling.
  - **Layer 6:** 512 filters, batch normalization, ReLU activation, max-pooling.
  - **Layer 7:** 1024 filters, batch normalization, ReLU activation, max-pooling.
- **Flatten Layer:** The output is flattened to a vector to feed into the fully connected layer.
- **Dropout Layer:** A dropout layer with a probability of 0.5 is used to prevent overfitting.
- **Fully Connected Layer:** The final layer consists of 102 output units, corresponding to the 102 flower classes.

B. Loss Function

The Cross-Entropy Loss function is employed to evaluate the model's performance. It is defined as:

$$L(y, \hat{y}) = -\sum_{i=1}^{N} y_i \log(\hat{y}_i)$$

where y represents the true label and $\hat{y}$ denotes the predicted probability for each class [5].

C. Optimizer

The Adam optimizer [5] is utilized for training the model. It is configured with a learning rate of 0.0001 and a weight decay of 1×10^{-5} for L2 regularization. To dynamically adjust the learning rate during training, the Cosine Annealing Learning Rate Scheduler is employed, which modifies the learning rate following a cosine curve.

D. Data Augmentation

To enhance the robustness and performance of the model, several data augmentation techniques are applied to the training data. These techniques include random resized cropping, horizontal flipping, random rotation, and color jittering. Such augmentations introduce variability in the training data, thereby improving the model's generalization to unseen data [8].

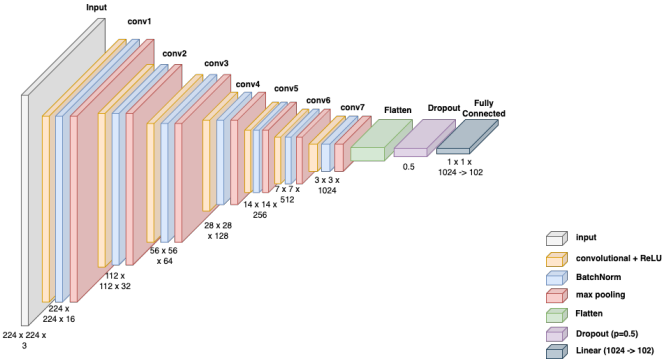

Fig. 1. The architecture of the Convolutional Neural Network (CNN) used for classifying the Oxford 102 Flower dataset.

III. Results & Evaluation

This section details the experiments conducted to train and evaluate the neural network model for the flower species classification task using the Oxford 102 Flower dataset [4]. The evaluation metrics, training setup, hyperparameters, and results are comprehensively presented.

A. Experiments

The experiments were conducted on a MacBook Pro M2 with a 19-core GPU. The model training and evaluation processes were implemented using PyTorch [7], a widely-used deep learning framework. The focus was on designing and fine-tuning a Convolutional Neural Network (CNN) architecture tailored for the classification task.

B. Hyperparameters and Training Setup

The key hyperparameters and settings used in our experiments are as follows:

- **Optimizer:** Adam optimizer [5] with a learning rate of 0.0001 and weight decay of 1×10^{-5} for L2 regularization.
- **Learning Rate Scheduler:** Cosine Annealing Learning Rate Scheduler with T_{max} set to 100 [9].
- **Batch Size:** 32
- **Epochs:** Up to 600 epochs with early stopping.
- **Early Stopping:** Triggered if validation loss does not improve for 120 consecutive epochs.
- **Data Augmentation:** [8]
  - Random resized cropping to 224x224 pixels.
  - Random horizontal flipping.
  - Random rotation up to 10 degrees.
  - Color jittering with brightness, contrast, saturation, and hue adjustments.
  - Normalization with mean [0.485, 0.456, 0.406] and standard deviation [0.229, 0.224, 0.225].

C. Training Time

On the MacBook Pro M2 GPU, the model took approximately 10.5 seconds per epoch. Early stopping was typically triggered around 500 epochs, resulting in a total training time of approximately 1.45 hours.

D. Evaluation Metrics

The primary evaluation metrics used to assess the model's performance were:

- **Training Loss:** The average loss calculated on the training set.
- **Validation Loss:** The average loss calculated on the validation set.
- **Validation Accuracy:** The percentage of correctly classified samples in the validation set.
- **Test Accuracy:** The percentage of correctly classified samples in the test set.

E. Results

The table below summarizes the performance of our neural network model at different stages of training:

TABLE I
Performance Metrics at Different Stages of Training

Epoch	Train Loss	Validation Loss	Validation Accuracy (%)
1	4.8386	4.3833	4.22
10	3.6143	3.2143	23.04
25	2.9514	2.7180	33.04
50	2.1572	2.0850	47.55
100	1.6743	1.7818	55.10
200	1.3585	1.7303	57.45
400	0.7165	1.5867	63.14
500	0.4222	1.4164	67.23

Final Results:

- **Validation Accuracy:** 67.35%
- **Test Accuracy:** 64.22%

These results indicate that the model was able to achieve a reasonable level of accuracy in classifying flower species, validating the effectiveness of the architecture and training strategy employed.

IV. Conclusion & Further Work

This project successfully developed a Convolutional Neural Network (CNN) from scratch to classify images from the Oxford 102 Flower dataset. The final model achieved a validation accuracy of 67.35% and a test accuracy of 64.22%, demonstrating its effectiveness in categorising flower species.

The selected architecture, which includes multiple convolutional layers, batch normalization, ReLU activations, and dropout, proved to be a suitable choice for this task. The application of data augmentation techniques such as random cropping, horizontal flipping, rotation, and color jittering significantly enhanced the model's robustness and performance [8].

A. Successful Aspects

Several components of the approach were particularly effective:

- **Data Augmentation:** The introduction of variability in the training data through augmentation techniques enhanced the model's ability to generalize [8].
- **Batch Normalization:** This technique stabilized and accelerated the training process [5].
- **Dropout:** Overfitting was effectively reduced by preventing the model from relying too heavily on any single feature [5].
- **Cosine Annealing Scheduler:** The dynamic adjustment of the learning rate improved convergence [9].

B. Areas for Improvement

Despite the successes, some challenges and areas for improvement were identified:

- **Training Time:** The training process was relatively time-consuming, even with early stopping.
- **Model Complexity:** While the model performed well, further optimization of the architecture could lead to better performance and faster training.
- **Hyperparameter Tuning:** Additional experiments with different hyperparameter values might yield improved results.

C. Future Work

Future research could focus on several key areas to enhance the model's performance and efficiency:

- **Advanced Architectures:** Exploring more advanced neural network architectures, such as residual networks (ResNet) [3] or inception networks, could provide better performance.
- **Automated Hyperparameter Tuning:** Implementing automated hyperparameter tuning methods like grid search or Bayesian optimization to systematically find the optimal settings.
- **Transfer Learning:** Although not applied in this project, transfer learning techniques in other contexts could significantly improve performance and reduce training time.
- **Hardware Utilization:** Utilizing more powerful GPUs or distributed training methods could expedite the training process and allow for more extensive experimentation.

Overall, this project demonstrated the feasibility and effectiveness of using a custom-designed CNN for the task of flower species classification. The insights gained provide a solid foundation for further improvements and applications in related domains.

References

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," *Advances in Neural Information Processing Systems*, vol. 25, 2012.

[2] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[3] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[4] M.-E. Nilsback and A. Zisserman, "Automated Flower Classification over a Large Number of Classes," in *Proceedings of the Indian Conference on Computer Vision, Graphics and Image Processing*, 2008.

[5] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.

[6] C. M. Bishop, *Pattern Recognition and Machine Learning*. Springer, 2006.

[7] A. Paszke, S. Gross, F. Massa, et al., "PyTorch: An Imperative Style, High-Performance Deep Learning Library," *Advances in Neural Information Processing Systems*, vol. 32, 2019.
[8] C. Shorten and T. M. Khoshgoftaar, "A survey on Image Data Augmentation for Deep Learning," *Journal of Big Data*, vol. 6, no. 1, p. 60, 2019.
[9] I. Loshchilov and F. Hutter, "SGDR: Stochastic Gradient Descent with Warm Restarts," *arXiv preprint arXiv:1608.03983*, 2016.