INT2 Group 14 Report

Group 14: Aous Abdulla, Ryan Doyle, Maddie Jones, Owen Lister, Sanjna Srinivasan

***Abstract*—Abstract — The task for this project was to construct and train a neural network which can classify images of flowers into one of 102 categories of species, given by the Oxford 102 Category Flower Dataset (Flowers-102). To complete the task, I used PyTorch to create a deep neural network that classifies the given flower images into the defined classes. This was achieved by first learning from tutorials via the PyTorch website, then adapting a basic network to my problem, and finally, fine tuning to improve accuracy. Overall, the network I designed achieved a classification accuracy of 69.9% on the flowers-102 test set.**

I. Introduction

THE Flowers-102 dataset comprises images of flowers from 102 different species, with each species represented by at least 40 images. The task was to develop a deep neural network to classify these images into their respective categories.

Before Convolutional Neural Networks (CNNs), computer vision heavily relied on manual feature extraction methods, like edge detection, texture analysis, and colour histograms; whilst useful, these techniques had limitations in capturing complex patterns and variations within images.

In 1989, LeNet-1 emerged as one of the first CNN models for image classification [1]. It utilized backpropagation to recognize handwritten digits efficiently. This concept evolved with increased hidden and fully connected layers, leading to LeNet-5, which achieved lower error rates on test data, and finally to Boosted LeNet-4 in 1998, with even smaller error rates [2].

AlexNet, introduced in 2012, significantly advanced image recognition by incorporating data augmentation. Techniques such as image translations, horizontal reflections, and RGB intensity alterations, expanded the training set and reduced overfitting [3].

More recently, region-based CNNs (R-CNNs) have enhanced image classification by using image segmentation to draw boxes around different objects within an image, followed by CNNs to classify these Regions of Interest (ROI). Mask R-CNN is among the most effective models for image classification today [4].

In computer vision, image classification is essential alongside object detection, semantic segmentation, and image segmentation; posing a challenging problem due to the processing power required to analyse image details down to individual pixels. The Flowers-102 dataset presents additional complexities; for example, two flowers of the same species may vary in colour or flowering stage, yet their images need to be classified into the same category. To address this, I created and trained a CNN to classify images, applying various transformations, into their respective categories.

II. Method

I created a Convolutional Neural Network (CNN) – a specialised neural network for image processing and classification – to classify images from the Flowers-102 dataset. I began by loading the official training, validation and testing splits. Due to restrictions on pre-trained networks, I then design a basic CNN network. After experimenting with various architectures, I decided that data augmentation was necessary given the dataset's relatively small size (8000 images total, with 1020 in the training split).

For the training data, I applied the following transformations: Random Resized Crop: Images resized to 250x250 pixels, randomly cropped between 50% and 100%, then resized back to 250x250, Random Rotation: Images randomly rotated up to 55 degrees, Color Jitter: Adjusted brightness and contrast by up to 0.5, and saturation by up to 0.2, Random Horizontal and Vertical Flip: Probability of 0.5. Gaussian Blur: Kernel size of 5. Conversion to Tensor. For validation and test data, images were resized to 250x250 pixels and converted to tensors. No additional pre-processing steps were performed, and normalisation was omitted due to no observed benefit.

The CNN architecture includes six convolutional layers with increasing filters (16 to 256) and a kernel size of 3x3, with stride 1 and padding 1. Each layer used Batch Normalization to stabilize and accelerate training, PReLU activation for flexible negative slopes, Max Pooling to reduce spatial dimensions, and Dropout (0.1 probability) in the first, fifth, and sixth layers to prevent overfitting.

After the convolutional layers, a flattening layer converted the multi-dimensional output into a one-dimensional vector for the fully connected layers, which had dimensions: (256 * 3 * 3, 512), (512, 256), and (256, 102), finally outputting for the 102 classes.

The model was trained for up to 250 epochs, with early stopping if no improvement in validation accuracy was observed for 25 consecutive epochs. This approach balanced training time and accuracy. The training cycle to determine the ideal optimizer and learning rate scheduler, achieving the following recorded results on the test set:

Scheduler	Adam	AdamW	RMSprop
ReduceLROnPlateau	54.8%	42.3%	51.4%
ConsineAnnealingLR	40.9%	46.6%	55.7%
LinearLR	56.5%	46.6%	53.5%

TABLE I
COMPARISON OF SCHEDULERS AND OPTIMIZERS

Following these performance metrics, the RMSprop optimizer with the CosineAnnealingLR scheduler was selected for its superior performance, and was used for all proceeding tests.

III. Architecture

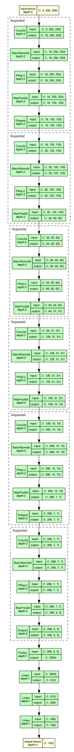

Fig. 1. Architecture Diagram of my CNN (zoom in for clearer text)

IV. Results & Evaluation

I conducted several experiments to achieve a 69.9% prediction accuracy on the testing split. This indicates that the model's predictions are significantly better than random guessing, demonstrating that the network has effectively learned from the dataset. To evaluate the neural network's performance, I used validation and test set accuracies. During training, I printed the loss and training accuracy at each epoch, as well as the loss and prediction accuracy on the validation set. This allowed me to monitor the network's performance and make adjustments as needed. Once training was completed, I loaded the best validation accuracy model state and tested it on the test dataset. I then graphed the testing accuracy against the validation accuracy, and the testing loss against the validation loss. This helped identify where the model plateaued and whether it was overfitting. The addition of dropout layers reduced overfitting, although it was still present at up to 30% towards the end of training. For the final architecture described previously, I ran the model for 250 epochs (approximately 1.5 hours) with various hyperparameters. The resulting accuracies after 250 epochs (or 25 early stopping epochs) are shown in Table II.

Learning Rate	Resulting Accuracy	Batch Size	Resulting Accuracy	Weight Decay	Resulting Accuracy
0.001	46%	8	53.6%	0.1	43%
0.0005	48%	16	50.8%	0.01	45.8%
0.0001	46.1%	32	48.3%	0.001	46.8%
0.00005	42.3%	64	47.7%	0.0001	47.6%

TABLE II
Comparison of Learning Rates, Batch Sizes, and Weight Decays

From these results, I chose a learning rate of 0.0005, a batch size of 8, and a weight decay of 0.0001. I then ran the model with these parameters for 500 epochs, taking approximately 3 hours and achieving a 62.1% accuracy. Observing continued progress, I extended the training to 2000 epochs, resulting in a 69.9% accuracy after 7.5 hours. The tests were performed on an M3 MacBook Pro with 36GB of Unified RAM and a 16-core GPU.

V. Conclusion & Further Work

Overall, a good result was achieved with a final accuracy of 69.9%. Initially, I aimed for an accuracy above 75%, but this proved more challenging than anticipated due to the restriction on using pre-trained models. After achieving only 30% accuracy in the first two weeks, I set a more realistic goal of 55%, which was eventually surpassed.

The chosen architecture was appropriate for this project, considering the level of complexity and the imposed restrictions. For further work, I aim to investigate the overfitting observed in the model and understand why the accuracy plateaued at 60-70

This project has significantly enhanced my knowledge and interest in building convolutional neural networks. I now have a strong understanding of CNNs and their intricacies.

While the resulting 69.9% test accuracy may not be sufficient for real-world applications, it is a commendable achievement given the project's constraints and available resources. The architecture can be further improved by exploring fine-tuning techniques such as varying the types and numbers of layers, activation functions, learning rates, batch sizes, optimizers, and loss functions. Future work could involve experimenting with more advanced architectures, such as ResNet, known for its robustness and capability with the utilisation of residual layers [5].

The project progressed well, with significant improvements in accuracy over time, reflecting my growing expertise in neural networks. I learned a great deal by testing different parameters, optimizers, and layers; this knowledge will be invaluable in future projects.

References

[1] Y. LeCun et al., "Backpropagation Applied to Handwritten Zip Code Recognition," Neural Computation, vol. 1, no. 4, pp. 541-551, Dec. 1989, doi: 10.1162/neco.1989.1.4.541.

[2] Y. Lecun et al., "Gradient-based learning applied to document recognition," Proceedings of the IEEE, vol. 86, no. 11, pp. 2278-2324, Nov. 1998, doi: 10.1109/5.726791.

[3] A. Krizhevsky et al., "ImageNet classification with deep convolutional neural networks," Commun. ACM, vol. 60, no. 6, pp. 84–90, Jun. 2017, doi: 10.1145/3065386.

[4] E. Odemakinde, "Everything about mask R-CNN: A beginner's guide," Viso.ai. Accessed May 9, 2023.

[5] K. He, et al., "Deep Residual Learning for Image Recognition," 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 770-778, doi: 10.1109/CVPR.2016.90.