Assignment Report on Neural Networks

Y3919175

***Abstract*—The task was to construct and train a deep neural network classifier to categorize flower species using the Oxford Flowers-102 dataset. A deep convolutional neural network architecture was designed and implemented using PyTorch, incorporating various layers and a loss functions to optimise performance. The model was trained and fine-tuned using the training and validation sets respectively, and evaluated on the test set. Data augmentation techniques were applied to improve the generalizability of the model, and to reduce the model from overfitting to the training data. The achieved classification accuracy on the flowers-102 test set is 52.24%**

I. Introduction

THE task of this project is to develop a neural network classifier, which has solely been trained using the training set of the Flowers-102 dataset, to identify flower species from images.

1) Historical Approaches to Image Classification: Before convolutional neural networks (CNNs), image recognition models relied on manual feature extraction and subsequent machine learning algorithms. For example, "SIFT is particularly noted for its robustness against image transformations like scale, rotation, and changes in illumination, making it a foundation for earlier image recognition technologies. This algorithm extracts distinctive invariant features from images, which can be used to perform reliable matching between different views of an object or scene" [1]. SIFT is more computationally intensive than modern CNNs, especially with large or high-resolution datasets. Once trained, CNNs process new images more quickly.

2) Current Approaches to Image Classification: With the advent of deep learning, CNNs have become the dominant approach, achieving industry leading performance across various datasets. CNNs automatically learn hierarchical features from raw pixel data, eliminating the need for manual feature extraction. This shift has significantly improved the accuracy and robustness of image classification models. From this, using pretrained models such as ResNet, people have achieved accuracy scores of upwards of 96% [2].

3) Importance of the Problem: The flowers-102 dataset presents a specific challenge due to the high intraclass variability and the similarity between classes between different flower species. Addressing this challenge is crucial as accurate flower classification can have practical applications in many areas, such as educational, recreational, or conservation purposes—but also for professionals in fields such as botany, agriculture, and environmental science, where precise flower recognition can assist in biodiversity assessments, crop management, and ecological monitoring [3].

4) My Approach to the Problem: I have designed, and implemented a deep neural network using PyTorch to classify images into 102 different flower species. Simplified, the architecture leverages convolutional layers for feature extraction, followed by fully connected layers for classification.

II. Method

1) Network Architecture: CNNs are designed to automatically and adaptively learn spatial hierarchies of features through backpropagation by using multiple building blocks, such as convolution layers, pooling layers, and fully connected layers [4]. The architecture consists of four convolutional layers with ReLU activation functions to introduce non-linearity and enable the model to learn complex patterns. Followed by a max-pooling layer to reduce the spatial dimensions of the feature maps and to make the model invariant to small translations. The output is then flattened from a multi-dimensional array to a 1D tensor, as fully connected layers require inputs as single feature vectors. A dropout layer has a 0.5 probability of deactivating neurons to reduce overfitting. This means that each neuron, along with its connections, is temporarily "dropped", and does not participate in forward propagation or backpropagation. These steps progressively extract high-level features from the input images which will be used to classify the data, to output a final classification, the extracted features are then passed through the first fully connected layer, to a dropout layer again, and then to the second fully connected layer. This process outputs the probabilities for the flower classes.

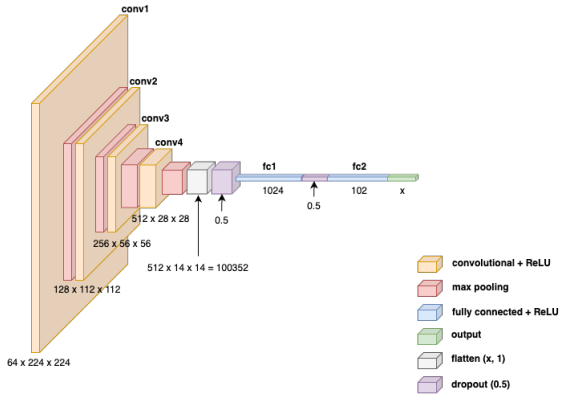

Fig. 1. Diagram of the convolution neural network architecture.

2) Loss Function: I employed PyTorch's cross-entropy loss function to measure the difference between predicted probabilities and true class labels. This suits multiclass classification, optimizing network parameters by minimizing prediction errors. CrossEntropyLoss heavily penalizes incorrect predictions, enhancing convergence during training.

$$l_n = -w_n \log \left(\frac{\exp(x_{n,y_n})}{\sum_{c=1}^{C} \exp(x_{n,c})} \right) \cdot 1_{\{y_n \neq \text{ignore\_index}\}}$$

Where: l_n The loss for the n-th example in the dataset, w_n The weight for the (n)-th sample, x_n The input logits for the (n)-th sample, $\exp(x)$ is the exponential function of x, C is the total number of classes., $1_{\{y_n \neq \text{ignore\_index}\}}$ is an indicator function that equals 1 when y_n is not equal to the ignore_index and 0 otherwise. [5]

3) Training Process: The model trains by iterating through epochs, learning as it goes. After each epoch, the model reports training loss, validation loss, and learning rate. Training loss measures how well the model fits to the training data, calculated by CrossEntropyLoss equation shown above. It allows for monitoring of how well the model is learning and improving over time. A decreasing training loss indicates that the model is effectively learning patterns in the training data. Likewise for validation loss, everything is the same except now the loss represents how well the model generalizes to new unseen data. This is used to see how well the model is likely to perform on the test set, as well as showing where the model starts overfitting to the training data, as at this point validation loss will stay the same or even increase. To combat this, the model makes use of PyTorch's ReduceLROnPlateau scheduler. If validation loss stalls for two epochs, the learning rate decreases by a factor of ten. Starting at a relatively high learning rate and letting it decrease over time allows for large steps initially to be made, which speeds up convergence. Reducing the learning rate helps the model escape local minima or saddle points in the loss landscape, where the model could otherwise get stuck and cease improving. As training continues, reducing the learning rate further allows the model to make smaller, more precise updates to the weights, which can lead to a higher test accuracy [6] as it allows for convergence to a deeper part of the loss surface. In order to stop the model at its best, each time a new lowest validation loss is achieved, the model dictionary is saved to be loaded again later. There is a patience system implemented. This causes the model to cease training if validation loss does not decrease for 5 epochs, which allows for it to be left to train without constant monitoring, as it will stop when no further improvements are being made. Technically training should last 100 epochs, but usually ends up stopping early around epoch 30 from this feature. After training, a graph is generated which shows how validation loss and training loss change over the range of epochs. This allows for the visualisation of overfitting, as where the lines start to diverge from one another shows the point at which overfitting started occurring.

III. Results & Evaluation

The models performance was evaluated on the official flowers-102 test set. The evaluation metric used was classification accuracy, which is the ratio of correctly classified images. Training for 90 minutes on my M1 Pro Macbook Pro with 16GB of memory (shared between CPU and GPU), setting the device in PyTorch to 'mps' (apple silicon equivalent of CUDA), I consistently achieve 51.78% accuracy on the test set using the Adam optimiser. Adam has shown to be more effective for image classification than other options, such as SGD [7], and was proven true though ,my testing of optimiser options. Shown below, I achieved varying results using different optimisers, but decided on Adam due to its consistency at giving a high accuracy. Hyperparameters such as learning rate, batch size, and the number of epochs were tuned based on the performance on the validation set. After lots of testing, I settled with an initial learning rate of 0.0005. This provided a steady decrease in loss without causing the model to become unstable, or miss out on finer patterns in the data. Furthermore, a relatively small batch size of 32 ensured that the model and its batch of images fit comfortably within the memory constrains of the GPU used to train. Each epoch with the above mentioned parameters made meaningful progress towards optimising the weights, without the learning being too noisy, as the case with smaller batches, or too averaged, as with larger batches. Other parameters used for training include train loader and validation loader, which loads the train data and validation data in batches respectively. Train loader is set to shuffle as to reduce the chances of learning patterns in the training data, which causes overfitting. Epochs are set to 200, however due to the patience system, which is set to stop the model after no improvements are made after five epochs, this was never reached. The choice of scheduler, specifically ReduceLROnPlateau, configured with mode='min, factor=0.1 and patience=2, decreases the learning rate when a plateau in the models performance is detected.

TABLE I
COMPARISON BETWEEN OPTIMISERS.

Optimiser	Average Test Accuracy
Adam	52.24%
SGD	47.45%
ASGD	48.34%

IV. Conclusion & Further Work

The project successfully implemented a CNN for classifying images from the Oxford Flowers-102 dataset. While the achieved accuracy of 52.24% demonstrates the model's ability to distinguish among a diverse set of flower species, it is evident that there is significant room for improvement compared to state-of-the-art models, which can achieve much higher accuracy. The architecture, which included convolutional, max-pooling, and fully connected layers, played a crucial role in learning hierarchical feature representations from the diverse flower images in the dataset. This foundational structure provided a robust framework for feature extraction and classification, showcasing the model's capability to process and analyze complex visual data effectively. Aspects of the project went well was the effective use of the PyTorch framework and the implementation of a custom training loop that adaptively adjusted learning rates. Despite efforts with data augmentation and dropout layers, challenges like overfitting, indicated by validation loss plateaus, persisted. Classifying unseen data into 102 different classes from only 1020 training images does not provide much training data, as there is only around 10 images per class of flower for the model to learn from.

REFERENCES

[1] R. Yamashita, M. Nishio, R. K. G. Do, and K. Togashi, "Convolutional neural networks: An overview and application in radiology," *Insights into Imaging*, vol. 9, no. 4, pp. 611–629, Jun. 2018. DOI: 10.1007/s13244-018-0639-9. [Online]. Available: https://doi.org/10.1007/s13244-018-0639-9.

[2] A. Kolesnikov, L. Beyer, X. Zhai, *et al.*, "Big transfer (bit): General visual representation learning," May 2020, [Online]. Available: arXiv:1912.11370v3 [cs.CV].

[3] Y. Li, Y. Lv, Y. Ding, H. Zhu, H. Gao, and L. Zheng, "Research on a flower recognition method based on masked autoencoders," *Horticulturae*, vol. 10, no. 5, May 2024. DOI: 10.3390/horticulturae10050517. [Online]. Available: https://doi.org/10.3390/horticulturae10050517.

[4] L. Chen, W. Yi, L. Zhang, X. Wang, H. Tang, and J. Zhang, "Smart grid image recognition based on neural network and sift algorithm," in *2023 International Conference on Networking, Informatics and Computing (ICNETIC)*, Palermo, Italy, 2023, pp. 1–5. DOI: 10.1109/ICNETIC59568.2023.00071.

[5] *Torch.nn.crossentropyloss — pytorch 1.10.0 documentation*, https://pytorch.org/docs/stable/generated/torch.nn.CrossEntropyLoss.html, Accessed: 26-May-2024, PyTorch.

[6] S. Uppal, S. Raheja, and N. R. Das, "Fire detection alarm system using deep learning," in *2023 13th International Conference on Cloud Computing, Data Science & Engineering (Confluence)*, Noida, India, 2023, pp. 54–58. DOI: 10.1109/Confluence56041.2023.10048842.

[7] S. Mehta, C. Paunwala, and B. Vaidya, "Cnn based traffic sign classification using adam optimizer," in *2019 International Conference on Intelligent Computing and Control Systems (ICCS)*, Madurai, India, 2019, pp. 1293–1298. DOI: 10.1109/ICCS45141.2019.9065537.