IMLO report

Y3921636

***Abstract*—The task focused on in this report involves constructing and training a neural network classifier using the Flowers-102 dataset to perform flower species categorization. A Convolutional Neural Network (CNN) architecture has been developed comprising of convolutional and fully connected layers, along with batch normalisation, ReLU activation, max pooling and dropout. Stochastic Gradient Descent optimization along with data augmentation, learning rate scheduling and hyperparameter tuning optimised the model to allow for the network to be trained effectively. All methods and features combined led to a final accuracy on Flowers-102 test data to be 64.5%.**

I. Introduction

Image classification is the basis of computer vision, with the goal to automatically assign inputs to classes based on what is visible in the image. This problem holds significant importance within machine learning due to how useful it is in so many different practical applications. For example, image classification can aid in the diagnosis of diseases in biomedical imaging, analyse satellite images for environmental monitoring reasons and much more (1).

Research into deep learning has been long inspired by looking into how the human brain works, leading to artificial neural networks being created to tackle classification problems. In the 1990s the first Convolutional neural network (CNN) came to be, called LeNet, which involved the essential components of modern image classification networks. However, LeNet didn't leverage the full potential of CNNs, leading to advancements like AlexNet with ReLU activation and dropout, VGGNet going much deeper and ResNet including residual learning (2).

In this project, I'm tackling the task of flower species categorization. I will be approaching the problem by constructing a deep neural network architecture which will be trained, validated and tested all on the official splits given from the flowers-102 dataset. My aim for this project is to leverage the power of deep learning to develop a robust, accurate classifier capable of accurately discerning flower species from images.

II. Method

I have chosen to tackle the problem of classifying flower species using a Convolutional Neural Network (CNN) architecture. The CNN I've designed comprises several layers, namely convolutional, fully connected and others to extract features from input images and classify them into appropriate categories.

Convolutional layers are the main building blocks of a CNN, and are the simple application of a filter to an input that results in an activation. Repeatedly applying the same filter to an input results in a feature map, which is a map of activations. This feature map indicates the locations and strengths of detected features within the input, in our case, specifically representing edges, corners, shapes and textures within the input image (3).

My network includes four Convolutional layers, each followed by batch normalisation, rectified linear unit (ReLU) activation functions and max-pooling layers (4). I have employed batch normalisation to make the training for deep neural networks faster and more stable. ReLU is applied afterwards to introduce nonlinearity. I chose ReLU as it overcomes vanishing gradients, and works well with convolutional layers. After the activation function, a max-pooling layer is employed between the convolutional layers to reduce spatial dimensions and computational complexity. After all 4 sets previously described, a flatten layer then reshapes the output into a one dimensional tensor which is used in the fully connected layers.

My network includes three fully connected layers making every single neuron in one layer have a connection to every single neuron in the next layer, enabling a flow of information between each dimension, thus incorporating the entire image in the decision. These layers have customisable sizes making it possible to control the model's capacity and complexity.

Using both fully connected and convolutional layers means the relationship between the learned features (found in convolutional layers) and the sample classes is understood.

To help prevent the network becoming too closely fitted to the training data, called overfitting, a dropout layer is included after all batch normalisation layers to reduce variance shift which improves performance (5; 6). Overfitting can become a serious problem, preventing the model from making accurate predictions.

A loss function is fundamental to creating a suitable network for image classification. It measures how well the neural network models the training data. Without this, the network can't tell if certain weights improve the performance of the network or not. As this problem requires a classification neural network I am using a Categorical Cross-Entropy Loss function (7). This employs the concept of Entropy, which measures the uncertainty of an event, where a lower entropy implies the model is confident. Cross-Entropy loss functions quantifies the dissimilarity between the predicted probability distribution (the output of the model) and the actual distribution (the true categorical label). The formula is shown below:

$$-\sum_{c=1}^{M} y_{o,c} \log(p_{o,c})$$

By minimising loss, the model learns to prioritise assigning higher probabilities for correct classes and simultaneously diminishing the probabilities for incorrect classes. This improves the model's accuracy, refining its ability to make precise class assignments.

III. Results & Evaluation

In deep learning an optimizer is used to find model parameters that minimise the loss function, resulting in better predictions.

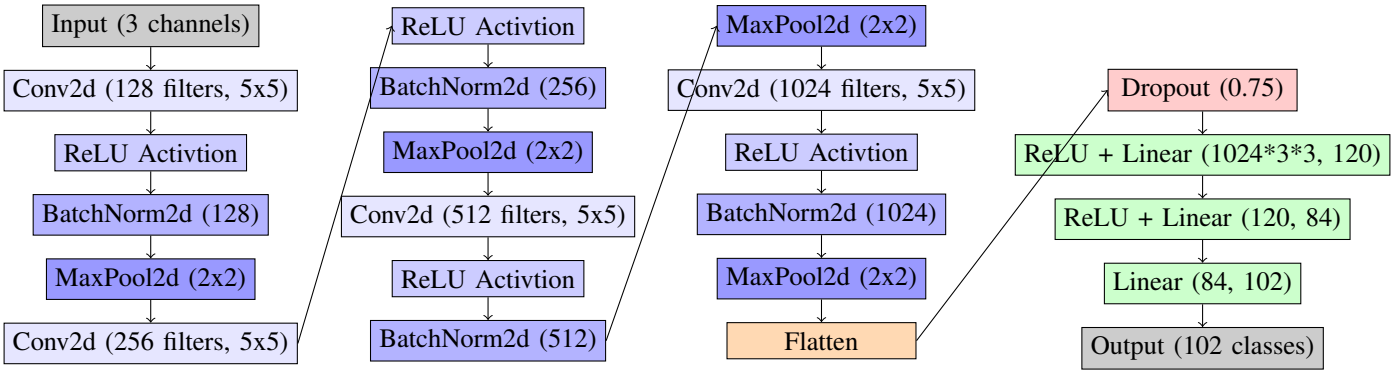

Fig. 1. Diagram of Network Architecture

To train my network I chose to use the Stochastic Gradient Descent (SGD) optimization algorithm, as research highlighted it as a strong algortihm compared to others (8), and in personal experiments it performed the best.

I implemented a 'reduce learning rate on plateau' scheduler after noticing overfitting and to help with the limitation of the SGD optimizer with the fluctuations not allowing convergence into local minima. This reduces learning rate when validation loss stops improving.

I applied multiple different transformations onto the training data during development. Upon getting stuck around 50% accuracy with heavy overfitting I conducted data augmentation to get a larger pool of training data, which resulted in a three times greater training set containing: the original training images, randomly rotated/flipped images and images with colour changes. More data to learn from, especially when it's a 20% training, 80% testing split, ultimately resulted in network accuracy improvements, pushing it to 60+%.

Throughout development, I ran multiple different sessions of hyper parameter tuning. Initially for parameters like learning rate, weight decay and the fully connected layer parameters, then later for the convolutional layer filters. These experiments were performed by training the data for 20 epochs using specified parameters and using the validation accuracy to determine the best choices.

Accuracy	L1	L2
21.06%	200	350
24.56%	250	350
24.17%	350	350
24.46%	200	400
21.01%	250	400
26.17%	350	400

Accuracy	Weight Decay
12.53%	0.1
33.53%	0.01
34.54%	0.001

Shown above are the results from experimenting with fully connected layer hyper parameters (left) and weight decay values (right). For weight decay, it was determined that 0.01 would be a better fit than 0.001 as the accuracy was very similar but a larger weight decay tackles overfitting better, for the fully connected layers the best accuracy was chosen.

Tests were done using a ParameterGrid from sklearn.model_selection, which could were iterated through to test every combination of the supplied parameters.

I also found the dropout layer could be optimised at around 60% accuracy. This involved experimenting with multiple dropout layers but ultimately having one layer with a higher dropout rate of 0.75 produced the best results.

The optimal values that resulted in the highest accuracy for each parameter from the testing performed are listed: Learning Rate: 0.001, weight decay: 0.01, L1/L2: 350/400 (used in fully connected layers), filter1: 128, filter2: 256, filter3: 512, filter4: 1024. Dropout rate: 0.75, Patience: 8 (in SGD optimiser)

The parameters above are used in my final network model and optimizer. I conducted my final training on the official training splits of the Flowers-102 dataset, augmented with transforms, run on visual studio code on a Windows PC with a RTX 3070 GPU and 16GB RAM.

Using 200 epochs, the final training execution took 2 hours and 17 minutes, and when the model was run on the official test split in the Flowers-102 dataset it achieved a final accuracy of 64.5 %

IV. Conclusion

With my first instance of the model getting only 8% testing accuracy, the development of the current model has gone through many experiments, some yielding improvement, others not so much. Experimenting with more complex models that include a large number of filters, doing extensive transformations onto the training data, and experimenting with extreme values for hyper parameters have shown to be the worst of my choices for the model. However I found using data augmentation to increase the training data set, performing hyper parameter tuning and including dropout to the model have been the most influential, positive additions in aiding with the classification of images. At its current state, the worst part of the model is certainly the overfitting with my model being too dependent on the training data, limiting how well it can identify images within the test/validation sets.

In conclusion, although it has a simple architecture, my network has managed to achieve 64.5% in a short amount of development time. I find this result proves the architecture has a solid foundation for classifying images to a fair degree of confidence. With minor improvements, namely fixing the issues with overfitting, the model can certainly be pushed to a much higher degree of accuracy in classifying images to the flower species present in the Flowers-102 dataset.

REFERENCES

[1] A. Parti, "Understanding image classification: A comprehensive guide," *Pareto*, Apr 2024. [Online]. Available: https://pareto.ai/blog/image-classification

[2] W. Wang, Y. Yang, X. Wang, W. Wang, and J. Li, "Development of convolutional neural network and its application in image classification: a survey," *Optical Engineering*, vol. 58, no. 4, pp. 040 901–040 901, 2019.

[3] J. Brownlee, "How do convolutional layers work in deep learning neural networks?" Apr 2020. [Online]. Available: https://machinelearningmastery.com/convolutional-layers-for-deep-learning-neural-networks/

[4] K. O'shea and R. Nash, "An introduction to convolutional neural networks," *arXiv preprint arXiv:1511.08458*, 2015.

[5] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[6] X. Li, S. Chen, X. Hu, and J. Yang, "Understanding the disharmony between dropout and batch normalization by variance shift," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 2682–2690.

[7] S. Oladele, "An introduction to cross-entropy loss functions," *Machine Learning Cross-Entropy Loss Functions*, 2023. [Online]. Available: https://encord.com/blog/an-introduction-to-cross-entropy-loss-functions/

[8] A. C. Wilson, R. Roelofs, M. Stern, N. Srebro, and B. Recht, "The marginal value of adaptive gradient methods in machine learning," *Advances in neural information processing systems*, vol. 30, 2017.