Report of my Flowers102 Neural Network

James Leney, Y3876738

***Abstract*—This is a report on a convolutional neural network created to classify the images in the Flowers 102 Dataset. The network is four convolutional layers and then two fully connected layers, trained over 2000 epochs with a batch size of 128. It achieved a classification accuracy of 44 percent.**

I. Introduction

THIS assessment focuses on the creation of a neural network with the goal of classifying images of flowers accurately by species. There have been many different approaches to image classification in the past, but I think the most important steps came from Yan Le Cun and Krizhevsky [1], introducing convolutions, to reduce the dimensions of the input and the ReLU activation, for better non-linearity, both of which I have used. Currently, Image Classification is used for many things, such as face recognition or object detection in most phones, but arguably the most important thing it is used for is in medical image anaylsis, in which it can identify diseases or cancers [2].

II. Method

I started off by transforming the images to 64 x 64 to reduce the training time while I was building and testing the network, I also normalised the data using the mean and standard deviation which I calculated from the total dataset and applied a random horizontal flip for better generalisation [3]. I then began with a very basic convolutional network, this was because a convolutional neural network is able to automatically learn the spacial hierarchies of features, like edges and shapes, which is very important for recognising objects in images. It does this by sliding a kernel (3 x 3 in my case) over the image and performing element-wise multiplication and summation [4]. It consisted of 2 convolutional layers each followed by a ReLU activation, to introduce non-linearity, I used ReLU because of its efficiency and its ability to mitigate gradient-related issues[5]. And a pooling layer, to reduce the spacial dimensions of the output from the convolutional layer, this reduces computational load and number of parameters. Then a fully connected linear layer as the output layer with a softamax activation to convert the output to a probability distribution across the class labels which is used for prediction. Finally a batch size of 32 while using the Stochastic Gradient Decent (SGD) optimiser, this yielded results of around 12 percent classification test accuracy from 20 epochs, which was not ideal. After this I switched to using Adam as it is more robust and faster [6], because I used Adam, I used a learning rate of 0.001. The learning rate is the hyperperameter that controls the rate that the model's parameters are updated based on the gradients computed during backpropagation. After switching to Adam, I increased the amount of convolutional layers to 3, and then 4, by increasing the number of layers, I increased the modules capacity for finding relationships and ability to capture finer details, hopefully increasing accuracy. Further, I added a padding of 1 to avoid shrinking and ensured the stride was a default of 1, each new layer was followed by a ReLU activation and a pooling layer. I then added one more linear layer to for additional feature learning. To the linear layers, I added dropout layers to help prevent overfitting. I also increased the number of Epochs to 50 and then to 2000, allowing more time for my model to converge, and increased the batch size to 64 and then 128 for increased efficiency. Also during this I added sequential containers for both the feature extraction and classification sections of my model in an effort to make it more clear. After all these changes I changed the image size back to the full 256 x 256 but introduced a random resize crop to 224 pixels to the training data as it would increase variance in the data to help the model learn features in different positions, and cropped the validation and test images to 224 pixels to the center, as many images have the main focus in the center so it removes non-essential features. As I made these incremental changes the classification test accuracy also increased slowly increased. I trained the model by setting the model to train mode, iterating through batches of training, data passing the input images through the network to get predictions, calculating the loss between the predicted outputs and the actual labels using the loss function, then the gradients of the loss were computed using backpropagation, finally the parameters are updated using the Adam optimisation to minimise the loss. The loss is then added to the running loss. My final neural network is set out like this:

Convolutional Layer 1: 32 Filters, 3 x 3 Kernel size, A stride of 1, A padding of 1, then a ReLU activation and 2 x 2 max pooling.

Convolutional Layer 2: 64 Filters, 3 x 3 Kernel size, A stride of 1, A padding of 1, then a ReLU activation and 2 x 2 max pooling.

Convolutional Layer 3: 128 Filters, 3 x 3 Kernel size, A stride of 1, A padding of 1, then a ReLU activation and 2 x 2 max pooling.

Convolutional Layer 4: 256 Filters, 3 x 3 Kernel size, A stride of 1, A padding of 1, then a ReLU activation and 2 x 2 max pooling.

Fully Connected Layer 1: 1024 neurons, Dropout regularization and ReLU activation.

Output Layer: 102 neurons, one for each class, Dropout regularization then softmax activation.

I trained the model using the Cross-Entropy Loss function, due to its robustness and compatibility

H(P, Q) = — sum x in X P(x) * log(Q(x))
Where H() is the cross-entropy function, P may be the target distribution and Q is the approximation of the target distribution. [7]

with a learning rate of 0.001, which is suited for Adam.

III. Architecture Diagram

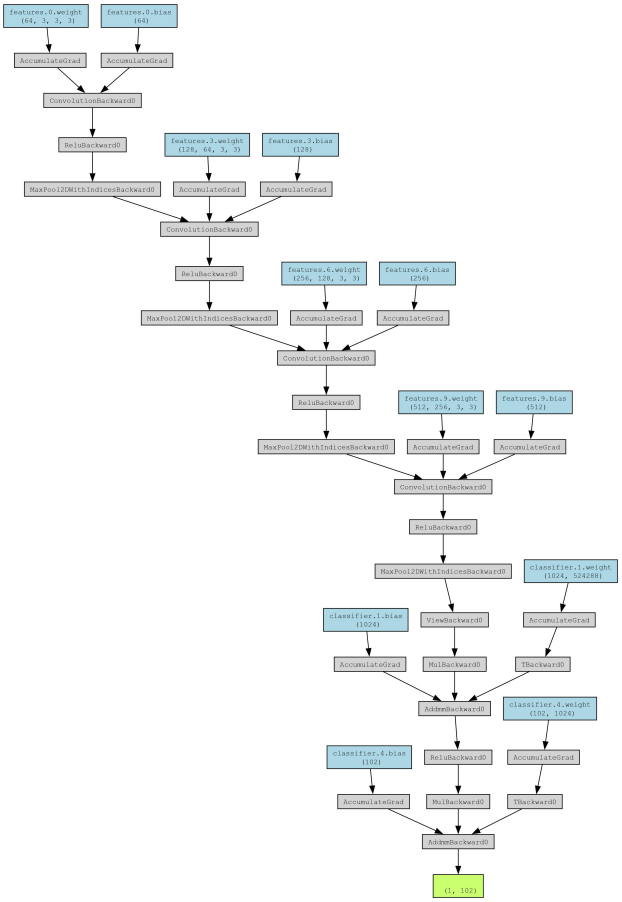

Fig. 1. Architecture Diagram.

IV. Results and Evaluation

As stated above, I tried various different experiments and iterations, such as increasing the amount of layers, increasing the batch size and number of epochs, and transitioning from an SGD optimiser to and Adam optimiser, the reasoning for all are stated above. The final Test accuracy was 44 percent after these changes, I computed this switching the model to evaluation mode after each epoch, iterating through batches of validation data without gradient computation, performing a forward pass to obtain predictions, then calculating the validation accuracy by dividing the correct number of predictions to the total number of labels. After each epoch, the epoch number, running loss, validation accuracy and time taken are printed to enable monitoring. If the most recent iteration provided the highest accuracy, the model parameters are saved. After all epochs are completed, the model parameters with the highest accuracy that were saved before are loaded, and much like before the model is set to evaluation mode, it then iterates through batches of test data instead of validation data, performs a forward pass to obtain predictions, and then once again calculates the accuracy by dividing the correct number of predictions to the total number of labels. Finally, the final test accuracy is displayed, giving the final result of 44 percent.

Hyperparameters used:
Epochs = 2000
Batch size = 128
Learning rate = 0.001

Hardware and environment:
GPU: NVIDIA GeForce RTX 4070ti SUPER
CPU: Intel Core i5-14600k
Operating System: Windows 11
IDE: Visual Studio Code
Framework: Pytorch 2.3.0

Training time:
Approximately 430 Minutes

V. Conclusion

The Neural Network I created only achieved a test accuracy of 44 percent. This is a low success rate and leads me to believe I could have changed a multitude of things. Future work could include creating a more complex Neural Network, with more layers, different optimisation strategies and possibly better data augmentation. I could also strive to include new techniques, such as transfer learning, which was not included in my original network due to me not being entirely familiar with it. The creation of the architecture was a strong point as it was understandable and worked as implemented, the challenges came when I struggled to manipulate it to increase the results effectively. Despite this I firmly believe that using a Convolutional Neural network was the correct choice as it still has the best attributes as stated in the Method section, I would just need to make the changes stated above.

REFERENCES

[1] A. Karpathy, "The Evolution of Image Classification Explained," Stanford University, Stanford, CA, USA, 2018. [Online]. Available: https://stanford.edu/ shervine/blog/evolution-image-classification-explained. [Accessed: May 26, 2024].

[2] "Image Classification," Viso AI, 2022. [Online]. Available: https://viso.ai/computer-vision/image-classification/. [Accessed: May 26, 2024]. J. Brownlee, "Cross-Entropy for Machine Learning," Machine Learning Mastery, 2019. [Online]. Available: https://machinelearningmastery.com/cross-entropy-for-machine-learning/. [Accessed: May 26, 2024].

[3] S. Rashid, "What is Image Normalization," Medium, October 2019. [Online]. Available: https://medium.com/@shoaibrashid/what-is-image-normalization-d8305bf328c0. [Accessed: May 26, 2024].

[4] S. Toshniwal, "Why Are Convolutional Neural Networks Good for Image Classification?" Data Driven Investor, Medium, 2020. [Online]. Available: https://medium.datadriveninvestor.com/why-are-convolutional-neural-networks-good-for-image-classification-146ec6e865e8. [Accessed: May 26, 2024].

[5] Bharath Krishnamurthy, "ReLU Activation Function," Built In. [Online]. Available: https://builtin.com/machine-learning/relu-activation-function. [Accessed: May 26, 2024].

[6] Rahul Agarwal, "Adam Optimization," Built In. September 2023. [Online]. Available: https://builtin.com/machine-learning/adam-optimization. [Accessed: May 26, 2024].

[7] UnpackAI, "Cross-Entropy Loss in ML," Medium, 2021. [Online]. Available: https://medium.com/unpackai/cross-entropy-loss-in-ml-d9f22fc11fe0. [Accessed: May 26, 2024].