Individual Project Report

Y3919217

***Abstract*—This report details how I designed and constructed a neural network to classify images of flowers-102 dataset correctly. To complete this I created a convolutional neural network, training and validating each model iteration to improve accuracy then finally testing the model. I achieved an accuracy score of 75.2% on the flowers-102 test set.**

I. Introduction

In this project, I aimed to construct a neural network classifier that would take an image of a flower as an input, from the flowers-102 dataset and correctly output the classification of that flower. This was accomplished by training the classifier on the training data, validating after each epoch on the validation set, and testing the model against the test set to determine its accuracy. Historically this task has been undertaken using single-layered neural networks until multi-layered neural networks were available via back-propagation algorithms. When convolutional neural networks were developed, and combined with deep learning algorithms they provided the next major development for image classification. From here specific networks began to develop providing innovations and improvements [5].

Image classification is an important machine learning problem due to its wide range of applications. For instance, in healthcare image classification can help diagnose treatments and diseases in medical scans. Vehicle automation heavily relies upon image classification to correctly detect objects in real-time to provide navigation. To tackle this image classification problem I constructed a convolutional neural network and adapted it to reduce overfitting and increase accuracy.

II. Method

Initially, I followed a PyTorch tutorial [6] giving myself an understanding of the processes and methodology required to complete this task. I created a convolutional neural network as they have many different filters which process information quickly and can learn to detect features of an image without including manual feature extraction. I began with a 3-layered convolutional neural network and for each layer, an activation function, a max pooling layer followed, and a single fully connected layer. By training the model and adding extra layers, I reached a model with 5 convolutional layers going from 3 input to 512 output features but still one fully connected layer. This is where my model began overfitting so I added batch normalisation after each layer and dropout after the final layer to improve regularisation. For my optimiser I initially chose SGD, after some research I changed this to Adam which provided more accurate results[4].

I added multiple linear layers to my fully connected layers, with an activation function after each layer, with dropout at the end of the layers enabling the model to learn more complex patterns and reduce overfitting. Resulting in 2 linear layers stepping from 2048 features down to the 102 flower classes. Once my model had been set up, I began tuning the hyperparameters of the network to improve accuracy and generalisation[2]. Altering my dropout values until I found a more optimal value, settling on 0.5 which provided more accurate results and reduced overfitting. I adapted the learning rate to improve performance, discovering that a slower learning rate improved the accuracy of my model. However, a greater number of epochs were required to reach the same accuracy score. To combat this I implemented an adjustable learning rate which slowed down throughout the training process if the validation score didn't increase for a chosen number of epochs. However, this reduced the test accuracy so I slowed the learning rate after a specified number of epochs. For my loss function, I chose to use Cross-Entropy Loss, as it is popular for multi-class classification tasks, the formula for which is $\Sigma_C^N y_c log(p_c)$ where N is the number of classes, y_c is the probability distribution of the classes and p_c is the predicted probability distribution output by the model.

Having tuned the hyperparameters of the model, I started to augment the training data to improve test accuracy, reduce overfitting and keep the validation accuracy as close to the training accuracy as possible. The transforms I added attempted to increase the size of the training data set which would provide more accurate classifications[1].

This all culminated in a 10-layered convolutional neural network, the layers were paired, where the first layer increased in size and had batch normalisation, an activation function and a max pooling layer after it, and the second layer in the pair remained at a constant size and had batch normalisation and an activation function following it. Following all the convolutional layers dropout was added. The fully connected layers consisted of flattening the convolutional layers and then 3 linear layers with the first layer having batch normalisation and an activation function, the second layer having dropout and the third layer having batch normalisation, dropout and an activation function.

III. Neural Network Architecture

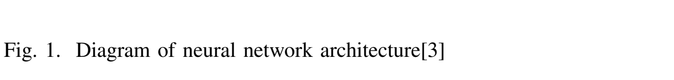

Fig. 1. Diagram of neural network architecture[3]

IV. Results and Evaluation

To evaluate my model I used the default train/validation/test split, after training each epoch the accuracy, total loss and epoch loss are calculated for the training data, then the epoch is evaluated against the validation data, the metrics for accuracy and the average loss is calculated on the validation set. As the model is being trained the validation accuracy is used to show the model performance until all epochs have been completed, the model then evaluates against the test dataset which is the final metric that determines the quality of the model. Having collected all this data a graph of training vs validation loss and accuracy was plotted to help visualise the training progression.

My experiments were all performed within a Jupyter notebook on a local machine, with the hardware used for each experiment being an AMD Ryzen 5 3600 CPU, NVIDIA Geforce GTX 1050Ti, and 16GB of RAM. The model is trained on the training set of data, for each epoch the model is then validated against the validation data. Once the model has finished training, it is tested against the test data to find the accuracy and the average loss. My initial results from the basic convolutional neural network with SGD as my optimizer were not very accurate and simply changing the optimizer to Adam resulted in a greater increase in accuracy. Running further training, the results gathered demonstrated that my model was greatly overfitting as my training accuracy would be reaching 100% while the validation accuracy was only reaching around 20%. Upon slowing my learning rate down and seeing an improvement in accuracy I concluded that taking more time to learn the features of a flower would provide better results.

TABLE I
RESULT COMPARISON BETWEEN DIFFERENT MODELS

	Validation Accuracy			**Test Accuracy**
Epoch	20	50	100	100
Model 1	22.3	28.6	32.1	26.9
Model 2	42.8	49.3	51.7	46.3
Model 3	33.4	41.3	47.2	43.4
Model 4	31.6	40.5	49.8	48.1

Model 1 consisted of 3 convolutional layers, each with an increasing number of features starting from 3 and ending at 128 features. It also had a single linear layer. Model 2 had 5 convolutional layers, with each layer also increasing in features and doubling the number of features resulting in 512 features. The number of linear layers increased to 2 hidden layers and a final output layer stepped down from 2048 to 1024 features before dropping to the number of classes. Model 3 had 8 layers which was the first model that utilized a paired layer configuration. In this configuration the number of features increased every other layer, resulting in 256 features, additionally, this model also added another hidden layer dropping down to 512 features before continuing down to the number of classes. Model 4 contained 10 total layers in the same pair configuration which resulted in 512 total features, this model also added batch normalisation and dropout to the hidden layers.

Having fine-tuned the hyperparameters of the model I began to augment the training data to improve the results the following table shows the results for different augmentations I tested.

TABLE II
RESULT COMPARISON BETWEEN DIFFERENT DATA AUGMENTS

	Validation Accuracy			**Test Accuracy**
Epoch	100	200	500	500
Augmentation 1	35.4	47.8	54.7	48.6
Augmentation 2	30.2	50.3	58.3	55.8
Augmentation 3	36.7	49.2	61.4	59.3
Augmentation 4	37.9	56.6	70.8	66.4
Augmentation 5	27.5	47.2	67.9	62.1

There were four different augmentations used in this process. Augmentation 1 included the following transforms: a conversion to tensor; normalization with mean=[0.485, 0.456, 0.406] and std=[0.229, 0.224, 0.225]; resize to 256 x 256; random rotation; and a random vertical flip. Augmentation 2 included the previous transforms and added a random horizontal flip, random adjust sharpness and colour jitter transforms. Augmentation 3 included the previous transforms adding random auto contrast and changing the value of the random rotation transform from 15 to 180 degrees. Augmentation 4, added the random affine transform. Finally, Augmentation 5 added the random erasing and random perspective transforms.

I achieved an accuracy of 75.2% in my final model after training 1000 epochs for 416 minutes (6 hrs 54 minutes). I used Adam as my optimization algorithm with a learning rate of 0.0001 and a weight decay of 0.0001. The batch size for training data was 8, while for validation and test data, it was 32. I implemented a scheduler that decreased the learning rate by 0.99 after 450 epochs. I used several data transforms for the training data including resize to 256 x 256; colour jitter with values of brightness=0.2, contrast=0.2, saturation=0.2, hue=0.1; random affine with values degrees=15, translate=(0.1, 0.1), scale=(0.8, 1.2); random rotation up to 180 degrees; random auto contrast; random adjust sharpness with values of (1.5, 0.5); random horizontal flip; random vertical flip; to tensor; random erasing; random perspective with values of 0.2 for the distortion scale and 0.5 for p; normalization with mean = [0.485, 0.456, 0.406] and std = [0.229, 0.224, 0.225]. For the validation and test datasets, I used the same normalization, resize and to tensor augmentations as used for training data, but without any of the other augmentations.

V. Conclusion and Further work

The architecture I chose was a good choice as it generalises well, does not overfit to the training data and the difference between the training accuracy and validation accuracy remains close throughout the training. This meant I achieved good performance with my model. Ensuring the difference between training and validation accuracy remained as small as possible greatly improved the overall accuracy. However, this meant I had to wait for a while for training to be completed and see whether an improvement was made. The addition of plotting a graph helped project potential results without having to run a larger amount of epochs. To further increase my classification accuracy I should further research and utilise pre-trained models and tune the hyperparameters as required.

References

[1] PyTorch. “Illustration of transforms.” (2017), [Online]. Available: https://pytorch.org/vision/main/auto_examples/transforms/plot_transforms_illustrations.html (visited on 05/10/2024).

[2] S. Ramesh. “A guide to an efficient way to build neural network architectures- part ii: Hyper-parameter selection and tuning for convolutional neural networks using hyperas on fashion-mnist.” (2018), [Online]. Available: https://towardsdatascience.com/a-guide-to-an-efficient-way-to-build-neural-network-architectures-part-ii-hyper-parameter-42efca01e5d7 (visited on 05/03/2024).

[3] A. Lenail. “Nn-svg.” (2019), [Online]. Available: https://alexlenail.me/NN-SVG/AlexNet.html (visited on 05/16/2024).

[4] D. Giordano. “7 tips to choose the best optimizer.” (2020), [Online]. Available: https://towardsdatascience.com/7-tips-to-choose-the-best-optimizer-47bb9c1219e (visited on 05/02/2024).

[5] N. Kushwaha. “A brief history of the evolution of image classification.” (2023), [Online]. Available: https://python.plainenglish.io/a-brief-history-of-the-evolution-of-image-classification-402c63baf50 (visited on 05/01/2024).

[6] PyTorch. “Pytorch tutorial: Building your model (beginner).” (2024), [Online]. Available: https://pytorch.org/tutorials/beginner/basics/buildmodel_tutorial.html (visited on 04/30/2024).