IMLO Report: Image Classification of the Oxford Flowers-102 Dataset

Y3920045

***Abstract*—The report contained information explaining the task, which was to train a machine learning model from the Flowers-102 dataset in order to classify an image into a set of pre-defined categories. Through my own knowledge learned and researched my aim was to create a deep learning neural network that could accurately identify and classify these images. In order to do this, I created Convolutional Neural Network which took in a split dataset of training data in order to learn patterns and fine details of these flowers. This CNN was actively updated throughout the process to reach an optimum accuracy of 57.77% when put against the testing dataset. This shows there is room for improvement that can be achieved with further training on other datasets and on larger scale.**

I. Introduction

THE task provided was to create a Neural Network from scratch to classify images within the Flowers-102 Dataset which consisted of 102 categories with each class consisting of 40 to 258 images of flowers. Machine learning has been utilised in all areas not just the Computer Science field. It is so integrated into our daily lives that many are unaware of its vast capabilities and the magnitude of its feats. It helps in many aspects such as face and speech recognition which can subsequently aid people with disabilities. The importance of image classification is not mentioned enough as its capabilities range across numerous technological and practical fields. Image Classification has almost limitless potential only limited by the complexity of the algorithms. In order for me to tackle this, I began by creating a Convolutional Neural Network which is often used for image classification due to layers being able to capture fine details and greater patterns whilst also being easily tuned to increase or decrease its complexity for the given dataset. This optimisation was achieved through the use of normalisation, generalisation and overfitting techniques such as dropouts, pooling and batch normalisation implemented within the PyTorch libraries [3].

II. Method

1) Loss Function and Optimisation: The network chosen for this project was a Convolutional Neural Network with my model containing six convolutional layers split into three blocks. This was then trained using the Cross Entropy Loss function for optimisation with Stochastic Gradient Descent (SGD).

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

Cross Entropy Loss was used as it is commonly used for determining the loss of multi-class classification tasks which is ideal for the Flowers-102 dataset. It measures the difference between the predicted values and the actual values.

2) Layering and Activation Function: By having six convolutional layers separated into three blocks it allows the input images to be fully learned whilst extracting the most amount of information from them. Within each block there is two convolutional layers increasing the size of the feature channels. The first layer starts at an input size of 3 and increasing to 512 in the last block. By having progressive layers, it allows the model to capture more features as it learns. This is due to different increasing input channels picking up on different aspects of the flowers. As the image is passed deeper into the network these aspects can be combined justifying the six convolutional layers as more complex patterns can be analysed. The early convolutional layers have a stride of 2 in comparison to later having a stride of 1. This allows the model to focus on a broader spatial dimensions of the image initially with general patterns and then being able to focus on finer details as the image is processed further in the CNN when the stride decreases. After the input data has been through the convolutional layer it is passed through the activation function which is Rectified Linear Unit (ReLU). ReLU is an activation function that allows for non-linearity of the data enabling the network to learn complicated patterns which is beneficial in Image Classification. The importance of this activation function is due to it defaulting any negative data points to 0 and using the maximum data point. Bharath Krishnamurthy states the importance of ReLU is due to it "helping mitigate the vanishing gradient problem during machine learning model training and enabling neural networks to learn more complex relationships in data"[1]

$$Relu(z) = max(0, z)$$

Having the activation function after each convolutional layer maintains consistency within the image data and provides faster computation during propagation of the model. However, CNN's with too deep layers can suffer from "Dying ReLU" problem.

3) Generalisation and Normalisation: After the activation function has been applied the model then performs a Batch Normalisation on the outputs. This is to regularise and prevent overfitting of the model on the training data which further improves the model's learning capabilities due to less generalisation on the data. At the end of each block, a Max Pooling layer is added which reduces the spatial dimensions of the input further reducing the chance of overfitting due to simplifying the image. When it is 'pooled' it allows the model to only focus on the most prominent features due to it condensing the input sizes after each block. The next convolutional block takes this as a new input. Having an Adaptive Average Pool at the end of the features it allows the

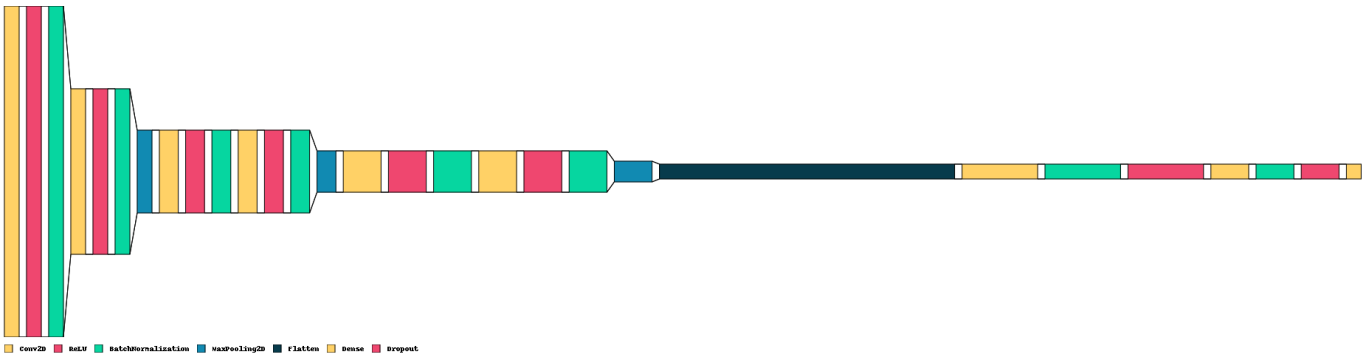

Fig. 1. CNN Architecture

model to be consistent when it is being fed into the Forward function. In this function, small amounts of Dropout are added to further reduce chances of overfitting and generalisation.

4) Structure: Initially the Neural Network's architecture consisted of having the features such as pooling, activation function and convolution being apart of the Forward function. Later this was taken away in order to improve the models modularity and simplicity by making it sequential. It allowed the model to be more thoroughly tweaked in order to be more efficient.

III. Results & Evaluation

The model was ran on a T4 GPU provided by Google Colab[2] environment which comes with integrated PyTorch libraries [3] and Python. This allowed the model to run at a time of approximately 1 hour to run 200 epochs achieving an accuracy of 57.77%. This CNN was trained using the Stochastic Gradient Descent learning algorithm with a Learning rate of 1e-3, Momentum of 0.9 and L2 Regularisation of Weight Decay at 1e-4. These Hyper Parameters were picked carefully in order to maximise the models efficiency when training it on the dataset. Having a small learning rate of 1e-3 results in better precision in the model as it can increment and learn slower not missing any crucial details. Introducing a weight decay into the learning algorithm allowed for the implementation of a second layer of Regularisation. Applying this further reduces the chance of the model overfitting and learning the training dataset too well improving generalisation to new and unseen data. Before the learning algorithm takes place the training data initially goes through a large array of transformations in order to reduce generalisation. These transformations occur randomly per epoch as it brought in through the trainloader so the data can't memorise the images it is training on, reducing overfitting. This results in an increase in the models classification accuracy as it learns more patterns and details, as it is training, rather than memorising the actual images. However, this didn't come without its challenges. Adding these transformation was a risk as some benefit the model whereas others hindered it, increasing factors such as 'RandomErasing' by marginal scales like 0.01 resulted in decreasing the models accuracy significantly from 50% to 32% due to key areas of the image being removed. This resulted in large amounts of trial and error until optimal rates were found. Along with this, there were a few choices between what learning algorithm to use which got narrowed down to two choices, Adam or SGD. I ended up using SGD as the learning algorithm as it yielded the best results despite being slower. SGD also seemed to be more consistent when ran multiple times, often Adam's results were varied from per run whereas SGD kept consistency throughout. Therefore, although Adam is often preferred in modern Neural Networks due to its increased speed and adaptive learning rates the chosen learning algorithm was SGD due to it having better generalisation on new untrained datasets. Originally I started with a batch size of 32 which was later increased to 64 after further research as it helped reduce generalisation along with stabilising the results making them more reliable. When training the model, it was put into training mode using PyTorch libraries [3]which allowed Dropout and BatchNorm to work efficiently compared to when it is in Evaluation mode when these instances are paused.

IV. Conclusion & Further Work

Overall, the model was successful achieving an accuracy of 57.77% showing that it is capable of taking in an input of images, learning and recognising patterns within to then accurately classify the images after. However, there is no shying away from the fact the model can be improved in terms of accuracy, learning times and overall efficiency. Despite extensive hyper-tuning furthering this to other parameters or even swapping loss functions and learning algorithms to ones which were not considered could see an improvement in the models accuracy. On more powerful hardware an increase in learning times can be improved and not being limited to T4 GPU on Google Colab, this can allow for more layers to be added while still allowing it to be ran in the allotted time whilst having the model pick up potential patterns it would not see otherwise. Along with this, enabling the model to be ran on other flower datasets outside the project will help it learn patterns resulting in better model accuracy. Employing pre-trained models that have been trained on more expansive datasets will see an increase in the accuracy of image classification on the flowers-102 dataset. This approach to furthering the model would allow it to become more refined reaching higher accuracy's in potentially decreased time on the dataset.

REFERENCES

[1] *B. Krishnamurthy, "Understanding ReLU, the Most Commonly Used Activation Function," Built In. [Online]. Available: https://builtin.com/machine-learning/relu-activation-function [Accessed 08 May 2024]*

[2] *Google, "Google Colab", google.com [Online]. Available: https://colab.google/. [Accessed 09 May 2024]*

[3] PyTorch, "PyTorch documentation — PyTorch master documentation," *Pytorch.org*, 2019. https://pytorch.org/docs/stable/index.html