CNNs for Image Classification

Group 7: Zack Tyler-Kyle, Sam Savery, Hamza Salman, Megan Miles and Niamh Hanratty

***Abstract*—Our task is to design, train and test a neural network for the classification of images of flowers. We made a convolutional neural network (CNN) that classifies images. We used PyTorch to build the CNN architecture collaboratively and used a Stochastic Gradient Descent (SGD) algorithm to improve the accuracy of the CNN. We achieved the classification accuracy of 32.23% on the flowers-102 test set.**

I. Introduction

THE task we sought to achieve is to create a neural network that takes a 2D input image of a flower, classifies this as a category of flower and outputs the result. We used the Oxford 102 Category Flower Dataset to train and test our neural network. Image classification is relevant in various fields, including the medical image classification in clinical practice [1]. This machine learning (ML) problem is closely related to the emergence of Big Data, providing opportunities [2] for more efficient training of neural network models as image classifiers.

Historically, image processing was completed using traditional Machine Learning techniques such as Support Vector Machines (SVMs) and Random Forests, however these methods heavily relied on manual intervention to generate features. Therefore making it difficult to scale the algorithm to work with larger datasets.

In modern day image classification, Convolutional Neural Networks (CNNs) have become the obvious choice. This is due to the use of filters and pooling that mimics human behaviour by dividing the image into sub-sections. However, when CNNs were first introduced in the 1980s by Yann LeCun they weren't feasible since it required a lot training data and computational power that didn't exist in the 1980s [3]. It wasn't until 2012, where AlexNet revolutionised the world of CNNs [4]. AlexNet introduced the idea of more layers in the network which leads to a better accuracy and using ReLU as the activation function instead of sigmod which vastly changed the non-linearity.

Over the years, there have been several additions to the CNN architecture, including different types of layers like the inception layer which was introduced in the inceptionNet CNN [3]. The inception layer proposed the concept of multiple filter of different sizes in the same layer to improve feature extraction. However, this layer drastically increases the computational needs of the network.

It is clear from research that there is a fine balance between computational power and accuracy of the model. Moving to create our own Neural Network we will have to experiment with the number of layers, the types and the activation function to decipher the best way to train the data.

II. Method

We created a CNN as this architecture is most specialised for image recognition. The convolutional layers are used for pattern recognition in images as features and storing these patterns to teach the model, for high accuracy and fast training times.

A. Network structure

The network we've created in Fig. 1.

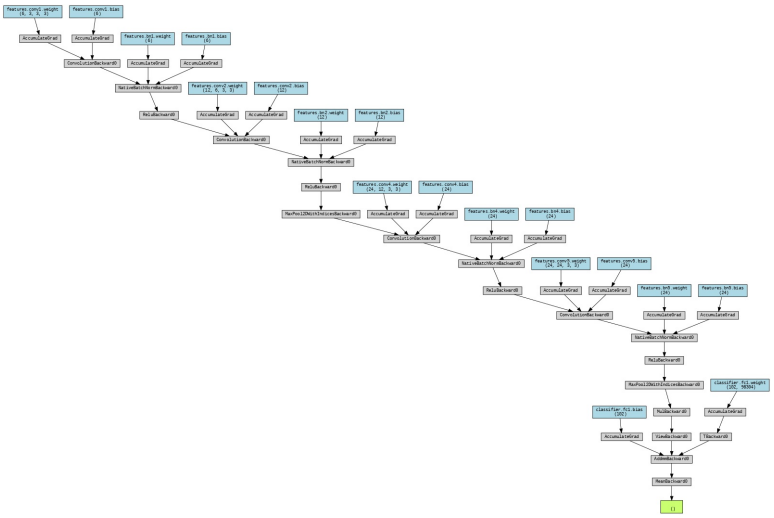

Fig. 1. Graph of network architecture.

Our loss function is the Cross Entropy Loss function, which "helps determine the accuracy of our model in numerical values". [5] Loss function is given in Equation 1.

$$L(y,t) = -\sum_{k=0} t_i \ln y_i \tag{1}$$

Termination conditions when the model is training include: 1) training time 2) number of epochs 3) chance of improvement.

1) Data Pre-processing: To begin training our model, we had to normalise the training dataset. This involved transforming images to 160x160 pixels; centre cropping; rotate images (only in training, not testing); transform images into tensors and make everything the same "brightness", using specific standard values across ML (reference). Then, loading and defining data.

2) Layers: Once pre-processing was completed, we followed a tutorial on YouTube which detailed how to create a CNN in PyTorch and a potential architecture that we could follow [6]. The video details a cycle of Conv layer, Activation Layer, Conv Layer, Activation layer and then pooling repeated several times over until the classification layer. Although we

found this useful we decided to expand on it and include a batch normalisation layer in the cycles. We found that the batch layer helps reduce the training time and generally improve the accuracy of the model. Although it increases the computational time, the benefits outweigh the negatives.

To start the network, we included a conv-batch-activation-conv-batch-activation-pool cycle. The first convolutional Layer will take in three channels, one for each RGB value and then expand this with 12 out-channels. We decided on 12 filters for this project as we needed a fine balance between a Neural Network that isn't computationally expensive and one that has an appropriate level of complexity and accuracy. We found that 12 was the best number.

For our activation layer, upon completing research [7] we found that the best function to use was ReLU. The activation layer is important to the overall structure of the network since it since it introduces non-linearity into the network so the model can learn complex relationships between the input and the output.

In the 7th layer, we included a pooling layer which we decided to use max pooling instead of average pooling. As a result, the model produces sharper edges and produces robust representations of the image, which means small changes in images won't make huge changes in the network as a whole. Therefore using Max pooling improves localisation and generalisation.

Next, to ensure that the network has the best opportunity to learn more about features and therefore improving the overall accuracy of the model we added another conv-batch-activation-conv-batch-activation-pool cycle.

In the 15th layer, there was a drop-out layer which randomly zeros a percentage of neurons in the layer. This is done so that the neurons do not depend on each other and start to learn more independently. However, later on in the project we decided to remove this layer since it was failing make decent progress in the learning. As a result the it was a lot faster to train the data and produced a better accuracy.

Finally, the classification layer which is responsible for predicting the label of the input image using the information gathered in the previous layers.

III. Results & Evaluation

The evaluation metrics used to judge the performance of our CNN are the training time and test accuracy as these are comprehensible metrics that can be used to compare results between other similar CNNs.

We used the Stochastic Gradient Descent optimiser, which finds the model parameters that correspond to the best fit between predicted and actual outputs, [8] because this popular optimisation algorithm produced better results for our model than other popular optimisation algorithm Adams. The improvement in accuracy between epochs was greater with SGD than Adams. Due to the dataset's small size and the limitations of our neural network's layers, the SGD was more suitable for our project.

A. Training parameters

Using the hyperparameters: BATCH_SIZE=16, LEARN_RATE=0.001, NUM_EPOCHS=100 with the parameters: WEIGHT_DECAY=0.0001, NUM_OF_CLASSES=102, with the CROP_SIZE=500, RESIZE_SIZE=256, our model produced the training and validation accuracy in Fig. 2. This shows the training accuracy increasing as expected, however the diverging training accuracy from the validation accuracy suggests overfitting in our model. We trained the model (with the supplied dataset), for 16 pochs before this graph, hence the accuracy not starting at 0. The training time for the final model was 1 hour and 45 minutes, with a final test accuracy of 32.23%.

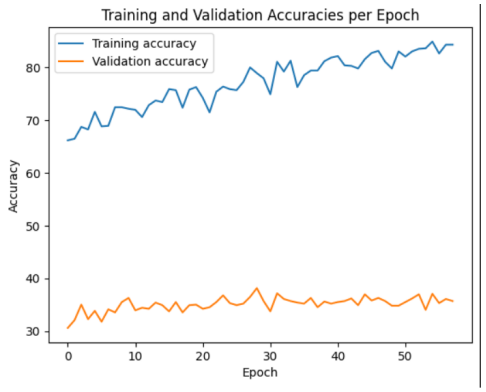

Fig. 2. Accuracy graph

The environment used for training the data was the Google Colab environment, with accelerated GPU, with the GPU type set to T4. Using this for training meant the hardware and environment was consistent throughout experiments, meaning changes in time to train and test accuracy was significant.

Initial conclusions from our results suggest our CNN does learn, improving the accuracy between epochs, but is not efficient enough to produce high test accuracy with such a small training set.

IV. Conclusion

In conclusion, our model's performance was poor in comparison to other neural networks with the same goal. Not enough training time. However, overfitting occurred, meaning the model was finding the best fit for the training data rather than becoming the best predictive model.

The CNN architecture used was a good choice because it is specialised for the task of data classification. Adding more convolutional layers may have improved the test accuracy.

Areas of the project that were successful were: the preprocessing of the data, in which the training set was maximised and the search area for the model was minimised. Areas for improvement include: overfitting, avoiding changing parameters during training in order to run several different experiments and compiling results for easier analysis of what improves accuracy in our model.

Further work on this CNN could include: further experimentation with layer configuration (i.e. use average pooling) to reduce overfitting.

REFERENCES

[1] J. G. L. Cai and D. Zhao, "A review of the application of deep learning in medical image classification and segmentation," *Annals of Translational Medicine*, vol. 8, no. 11, pp. 713–713, 2020.

[2] J. W. L. Zhou, S. Pan and A. V. Vasilakos, "Machine learning on big data: Opportunities and challenges," *Neurocomputing*, vol. 237, pp. 350–361, 2017.

[3] A. Amidi and S. Amidi, "The evolution of image classification explained," *Stanford Blogs*, 2019.

[4] I. S. A. Krizhevsky and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," Available at https://papers.nips.cc/paper/2012/hash/c399862d3b9d6b76c8436e924a68c45b-Abstract.html (01/01/1970).

[5] T. . Team, "What is cross-entropy loss?" Available at https://365datascience.com/tutorials/machine-learning-tutorials/cross-entropy-loss/.

[6] P. Loeber, "Pytorch tutorial 14 - convolutional neural network (cnn)," Available at https://www.youtube.com/watch?v=pDdP0TFzsoQ (07/02/2020).

[7] A.Thakur, "Relu vs. sigmoid function in deep neural networks," Available at https://wandb.ai/ayush-thakur/dl-question-bank/reports/ReLU-vs-Sigmoid-Function-in-Deep-Neural-Networks–VmlldzoyMDk0MzI (11/05/2022).

[8] M. Stojiljković, "Stochastic gradient descent algorithm with python and numpy," Available at https://realpython.com/gradient-descent-algorithm-python/.