22COC102 - Advanced Artificial Intelligent Systems

Student ID: F015011

Method: A systematic series of experiments to modify and evaluate one hyperparameter at a time while keeping all others constant in the given CNN model. This allows us to investigate its impact on the model's performance. After training, validating, and testing each model, we can compare their performance and recommend the optimal configuration. We will be experimenting with convolutional networks to classify the CIFAR-10 dataset, which consists of 60,000 32x32 pixel color images in 10 classes.

Tools: The project primarily uses *PyTorch* and *torchvision* for building and training neural networks, *matplotlib* and *numpy* for visualization and data manipulation, and *torch.optim* for optimizing the network's parameters. *SubsetRandomSampler* is used to randomly sample data from the dataset, *tqdm* is used to display a progress bar during training, and *sklearn.metrics* and *prettytable* are used to calculate and display various metrics and results.

Configurations

SimpleNet: Three convolutional layers and one fully connected layer. Each convolutional layer is followed by a max pooling layer and a Rectified Linear Unit (ReLU) activation function. The output of the third convolutional layer is flattened and then passed through a fully connected layer with ReLU activation. The torch.flatten function is used to flatten the output of the third convolutional layer before passing it through the fully connected layer. Adapted from [1].

LeNet5: Two sets of convolutional and max pooling layers, followed by two fully connected layers. The first set of convolutional and max pooling layers is followed by a second set, which is then followed by a third convolutional layer. The output of the third convolutional layer is flattened before being passed through the fully connected layers. The torch.view function is used to flatten the output of the third convolutional layer before passing it through the fully connected layers. Inspired from [2].

AlexNet: Five convolutional layers, three max pooling layers, and three fully connected layers. Each convolutional layer is followed by a ReLU activation function and a max pooling layer. The output of the last convolutional layer is flattened and then passed through the fully connected layers with dropout and ReLU activation functions. The AdaptiveAvgPool2d function is used to ensure that the output of the last convolutional layer has a fixed size before it is flattened. Inspired from [3].

References

[1] *Training a Classifier — PyTorch Tutorials 2.0.0+cu117 documentation* [online]. pytorch.org. (n.d.). [cited 19 March 2023]. Available at: https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html#training-a-classifier

[2] *Writing LeNet5 from Scratch in PyTorch* [online]. Paperspace Blog [cited 19 March 2023]. Available at: https://blog.paperspace.com/writing-lenet5-from-scratch-in-python/

[3] *Implementing Yann LeCun's LeNet-5 in PyTorch* [online]. Medium. [cited 19 March 2023]. Available at: https://towardsdatascience.com/implementing-yann-lecuns-lenet-5-in-pytorch-5e05a0911320

1 Experiment 1 (SimpleNet)

The following experiments seek to understand the impact of adjusting the learning rate and batch size on our SimpleNet model's performance.

1.1 Learning Rate Adjustment

The learning rate is a crucial hyperparameter in neural network training, as it determines the step size at which weights are updated during optimization. If the learning rate is too high, the model may diverge, while if it is too low, convergence may be slow. An optimal learning rate of 0.025 was found for SimpleNet, achieving the highest validation / test accuracy (64.33%) and converging reasonably quickly. When the learning rate is set too low, as in the case of 0.001, the optimizer appeared to make very small updates to the weights, which resulted in slow convergence during training. This is because the model takes longer to reach the optimal solution, and may have got stuck in a local minimum and thus had a testing accuracy of only 46.81%. On the other hand, when the learning rate was set to 0.05 and 0.1, the model was able to achieve a high accuracy early on in training, but it began to oscillate and plateau shortly after. This may be due to the learning rate being too large, causing the model to overshoot the optimal weights, achieving a test accuracy of 56.3% and 52.4% respecitvely.

Figure 1: lr=0.025

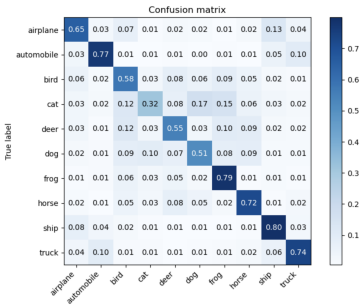

Figure 2: lr=0.025

1.2 Batch Size Adjustment

During the training process of our SimpleNet convolutional neural network, the model is trained on a given batch size of input data at a time. Smaller batch sizes (64 and 128) tended to improve training accuracy by allowing for more frequent, smaller updates to the model's weights. Despite converging more stably, this seemed to result in over fitting as presented by the decreased validation accuracy (56.59%) for this case. In contrast, larger batch sizes (512 and 1024) seemed to have more stable test / validation accuracy (62.5%) but took longer to converge to a good solution. A batch size of 256 formed a good balance between training efficiency and adaptability, yielding high testing accuracy (64%) while also having reasonable training time. The test accuracy was also higher for 256.

Figure 3: batchsize=512

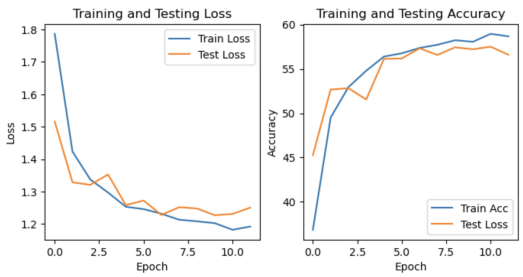

Figure 4: batchsize=64

2 Experiment 2 (LeNet5)

The following experiments seek to understand the impact of adjusting the optimizer algorithms and batch size on our more complex LeNet5 model's performance.

2.1 Optimizer Algorithms

For SGD, decreasing losses and increasing accuracies indicate improving and generalizing performance, but lagging validation accuracy suggests potential over fitting due to the optimiser's susceptibility to noisy gradients. Despite this limitation, SGD is the most optimal optimizer for the training and validation test sets among the three (68.79%). Adam's rapid decrease in losses and increase in accuracies in the first few epochs indicate quick attainment of optimal solutions. The close tracking of validation accuracy with training accuracy suggests Adam's ability to handle noisy gradients well, preventing overfitting. Despite this, the training plateaued, achieving an accuracy score of 46.98%. Similarly, Adagrad's quick decrease in losses and increase in accuracies followed by a plateau may indicate difficulty in finding further improvements. The increased validation accuracy early on suggests better generalisation but later potential of over fitting due to Adagrad's tendency to decrease learning rate too quickly for some parameters, resulting in a rudimentary model unable to capture all the patterns in the data.

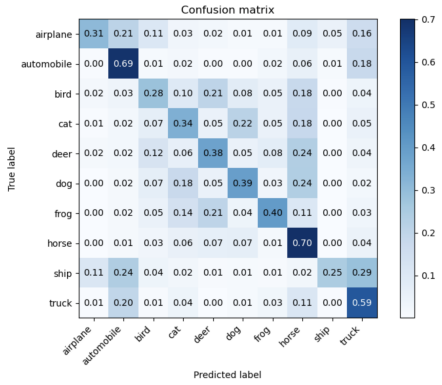

Figure 5: momentum=0.0

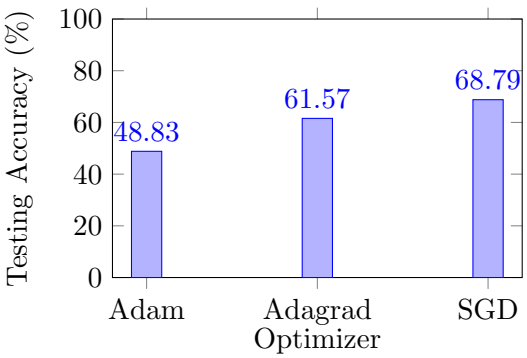

Figure 6: optimizer testing accuracy

2.2 Momentum

Momentum accelerates training and slows convergence; the momentum rate, ranging from 0 to 1, determines the degree of acceleration. At a momentum value of 0.0, we see very slow convergence, with the validation accuracy increasing only marginally over 12 epochs. This is because momentum helps the network avoid getting stuck in local minima by pushing it towards the global minimum, and a momentum value of 0.0 means that there is no such push; the resulting confusion matrix (Figure 5) demonstrates the poor test accuracy (43.44%) as a result. At a momentum value of 0.5, we see a noticeable improvement in the training process; the validation accuracy starts to increase much faster, showing increase convergence without overshooting the optimal solution. At a momentum value of 0.9, we see the best results. The validation accuracy improves much more rapidly than with either of the other momentum values, and our testing accuracy follows suit achieving 68.79%. This is because a high momentum value allows the network to take larger steps towards the minimum, which can help it avoid getting stuck in local minima. At 0.99, we see a very erratic and worse performing version of our LeNet5 model, demonstrating a final test accuracy of 52.36%.

3 Experiment 3 (AlexNet)

These experiments examine how adjusting the dropout rate and epochs affects the performance of the complex AlexNet model. Although accuracy is improved, this model requires significantly longer training times than previous models due to its complexity.

3.1 Dropout Rate

Dropout is a regularization technique used to prevent overfitting during training by randomly dropping out a percentage of neurons. Lower dropout rates can result in overfitting while higher rates can lead to underfitting due to the loss of information within AlexNet. At a 0.1 dropout rate, training accuracy is highest (83.56%) and validation accuracy is slightly lower (69.33%), still indicating reasonable generalization ability but certainly exhibits over training. A 0.5 dropout rate results in lower training accuracy (82.38%) but higher validation and test accuracy (87.76%), suggesting less overfitting and increased adaptability of the model. A 0.99 dropout rate results in very low training accuracy (19.50%), and even worse validation and test accuracy (10.01% meaning our model is guessing out of our 10 classes), indicating significant information loss during training and thus considerable underfitting. Therefore, 0.1 and 0.5 can be considered optimal dropout rates to balance underfitting and overfitting during AlexNet training.

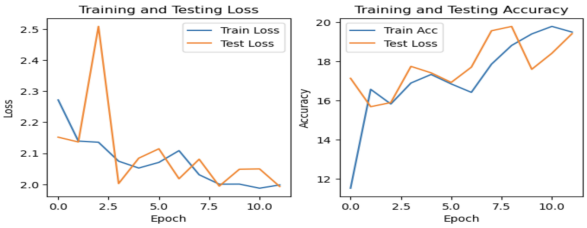

Figure 7: 99% Dropout Rate Training

3.2 Number of Epochs

AlexNet can have a larger training time due to its deeper architecture and higher number of parameters compared to our earlier models, and so the number of epochs we train it for can be crucial to efficiency when training. Unsurprisingly, as the number of epochs increases, both the training and validation accuracy show improvement, while the training and validation loss decrease, indicating the model's progress and ability to learn. However, there is a point where the accuracy improvement becomes less significant, and the loss decrease plateaus, revealing the onset of overfitting. In this case, when training for 24 epochs, our model achieves a training accuracy of 97.63% which can indicate overfitting but also presented a test accuracy of 93.35%, albeit at the cost of the longest training time (1hr 45m). In contrast, three epochs results in a low test accuracy of 39.69%, ascending to 62.49% after 6 epochs. After careful consideration, we found that training for 12 epochs offers an optimal trade-off between training time and accuracy, yielding an impressive testing accuracy of 87.76% in a shorter time (53 minutes).

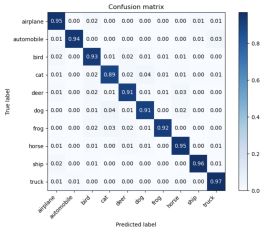

Figure 8: 24 Epochs Testing

Figure 9: 24 Epochs Loss and Accuracy