1. Introduction

This report details the design of a convolutional neural network for performing multi-class classification on the Cifar-10 dataset. It examines impact of several different hyper-parameters on model performance, with the objective of fine-tuning these in order to create a robust design. In the early stages of development, a simple model was used to perform of hyper-parameter tunning. As the project advanced, this information was leveraged in the design of a superior model. The results for previous runs were considered throughout the training process and updated accordingly to ensure best possible performance could be achieved.

1.1 Base Model Architecture

The base model architecture was constrained to having three layers for this task. Two of the three layers were convolutional layers to extract useful features from images and maintain a spatial relationship between pixels and learnt image features. The padding for these layers was set to “same” ensuring that the original input size was preserved, and no features were lost during convolutions. The stride was set to 1 as images in the cifar-10 data set were relatively small. To introduce non-linearity, relu activation functions were positioned after each convolutional layer. This ensured an increase in complexity and depth of abstraction within the network, which is especially important to encourage learning a relationship between different input variables. Layers would otherwise collapse without nonlinearities. Max-pooling layers were positioned after relu layers in order to reduce the dimensionality of the images. This operation was chosen due to favouring higher pixel values, this ensured the most useful features are preserved during dimensionality reduction. Typical pooling kernel sizes of 2x2 were used during pooling. In the final layer of the model, high-level features extracted from previous convolutional layers were flattened and passed through to a linear layer. It was necessary to include linear layers deeper into the network as so that a relationship between values in the data could easily be established. This was followed by a non-linear relu activation function before finally being outputted through the final linear layer for class predictions. As cross entropy loss was used, there was no need to include a soft-max outputs as the derivatives for this loss are already based on soft-max.

The dataset was normalised before being fed through to the input layer. The rationale behind this decision was to ensure that data points were in the same numerical range, so that large feature outliers do not incur a substantially higher/lower losses and all weights remained numerical stable. The data was mean-centred by subtracting the average from each individual value and variance-normalized by dividing the standard deviation. This ensured a zero mean and standard deviation of 1 for all data points. For all experiments detailed in this report, a validation set will be used to evaluate model performance by examining accuracy, loss and macro f1 score. Macro f1 score metrics were necessary for this task to montitor the performance per class. To isolate the impacts of each hyper-parameter, all other hyper-parameter values will remain fixed to the previous best parameter values unless otherwise stated in the text.

2. Hyper-parameter tuning

2.1 Network setup

To develop a strong base model for subsequent hyper-parameter exploration, it was necessary to choose suitable values for the number of convolutional channels and their corresponding kernel sizes. This describes the first line of investigation: Fine-tunning convolutional channels and kernel sizes. The list of channels and kernel sizes explored for convolutional layer 1 (C1) and convolutional layer 2 (C2) are detailed below.

C1:

No. channels: [32, 64]
Kernel sizes: [3, 5

C2:

No. channels: [64,128]
Kernel sizes: [5, 7]

The code was structured to allow all possible combination of convolutional channels and kernel sizes to be tested per run. When choosing values for these parameters the following constraint was imposed: for a given parameter value x it must hold true that C1(x) <= C2(x). The rationale behind this decision was twofold: firstly, local features are collected in shallower portions of the network and then reduced in deeper portions to represent more high-level representations. Secondly, the depth of the feature space should increase with the number of layers to represent increasingly abstract global structures. There were 16 total possible combinations, and the model was trained for 11 epochs per run.

Results

The general trend across runs was equal kernel sizes of 5x5 in C1 and 5x5 in C2 exhibited the worst performance in terms of accuracy. This is not surprising considering the constraint previously discussed. It is important that model complexity is increased as we move deeper into the network to ensure higher-level representations can be obtained. The best performance case in this instance were 3x3 in C1 followed by 5x5 or 7x7 in C2. This demonstrates that a larger gap between kernel sizes at each layer are most successful. In terms of the number of channels at C1 and C2, 32 and 128 exhibited the best performance in this case reaching an accuracy >= 70% (figure 2). The worst performance w.r.t the number of channels were 16 in C1 and 128 in C2. This is not surprising as for a relatively small network, less channels represent a smaller features space. A probable cause for this result is that 16 channels in the shallow layers of the network ceased to learn enough low-level information for this task. The overall best-case run was as follows:

C1:
No. channels: [32]
Kernel sizes: [3]
C2:
No. channels: [128]
Kernel sizes: [5]

These parameter values will be used for all subsequent exploration of hyper-parameters.

2.2 Learning rate

The learning rate is perhaps one of the most crucial hyperparameter to consider when designing machine learning models. Its role in regulating the rate of convergence is implicit, and if set too high, it risks overshooting the minimum. In this experiment three learning rates were tested starting at 0.01 and decreasing exponentially (see below).

- L1: 0.01
- L2: 0.001
- L3: 0.0001

Results

In the case of the highest learning rate L1, the model confronts the 'dying relu' problem [1][2]. As can be seen in the graph (figure 3) , the bias and weights gradients in the first and second layers remained stuck at zero during training. Consequently, the network ceased to learn; the weights remained fixed rather than updating. A probable cause is that a learning rate of 0.01 was too high, particularly when used in conjunction with ReLu activation. In this case, updated weight values were more likely to fall below the threshold for neuron activation in the backwards pass, since a higher learning rate was deducted from the prior weight values. A high learning rate could have pushed new weight value calculations into the negative constant range, and as relu becomes non-differentiable at w<=0, relu neurons become inactive [3].

In the case of L2 and L3 the model was not impacted by the ‘dying relu’. The best validation accuracy achieved with L2 was 71% with a loss value of 0.905 at epoch 5. Despite this, the model appeared to overfit to the training data at this point as accuracy began to decline to 69% and loss increased considerably to 1.5 by the final epoch. By point of comparison, L3 achieved a best accuracy score of 70.5% and a loss of 0.877. Overall, the training process was considerably more stabilized when using a lower learning rate. Despite the L2 accuracy score outperforming L3, the loss was notably lower throughout training in the latter case. Furthermore, L3 achieved a higher F1 score of 0.81 whereas the best F1 in the case of L2 was 0.71. As such, the lowest learning rate of 0.0001 deemed to be the most successful in this instance.

2.3 Batch size

The batch sizes were chosen in accordance with the number of samples in the training and validation set. It was important that the number of samples was divisible by the batch size so that all sample were used. Despite common practices of choosing batch sizes to a power of 2 in order to capitalize on GPU processing capability, the decision to ensure no observations were omitted during training seemed more intuitive. A total of 3 mini-batch sizes were trialled in this experiment (see below).

B1: 20
B2:40
B3: 80

Results

The results in this section exhibited better performance when smaller batch sizes were used. In the best case B1 a 71% (figure 4) accuracy was achieved with a loss value of 0.83(figure 5). In the worst case B3, the best accuracy achieved was 67% with a loss value of 0.92. Notably macro f1 score was 0.71 in B1 and 0.67 in B3 demonstration a decline in class prediction accuracy when larger batch sizes were used. A plausible explanation for this was that a smaller batch size had a regularization effect on the network due to its high variance [6]. When we consider that a relatively small learning rate of 0.0001 was used in this experiment, higher batch sizes may have resulted in overshooting the minimum. Conversely, the smallest batch size of B1, training speed was slowed, and smaller steps were taken in the direction of the minimum. A general rule in machine learning is “bigger batch size, bigger learning rate”.

2.4 Regularization procedures

As previous runs exhibited high loss values, regularization procedures were investigated as a means of preventing overfitting to the train set.

2.4.1. L2 regularisation

L2 regularisation is an approach commonly employed to prevent overfitting to the training data and encourage generalisation to the target data. This is achieved by penalising the sum of squared weights [4]. The square term ensures that large weights are penalised more heavily than smaller weights and that the weight reduction occurs at a distributed rate according to the weight value. As is common in practice, values for the regularisation parameter λ were chosen based on the logarithmic scale (see below):

- R1: 0.01
- R2: 0.001
- R3: 0.0001
- R4:0.00001

Results

In terms of accuracy, this experiment suggested a negative correlation between accuracy scores and L2 parameter values λ . Performance on the validation set was notably better when smaller values of λ were used. This was also apparent when examining loss and F1 scores which also demonstrated a noticeable improvement. The best-case trial was R4 with an accuracy score of 71.4% and a final loss of 0.82. Comparatively, the worst performance was in the case of R1 where scores for accuracy and loss were 65.7% and 1 respectively. A plausible explanation for these results is that larger values of λ penalised weights too heavily and valuable information was lost during training. Considering the network is limited to only 3 layers, this is not surprising as useful feature extraction is made difficult by a constrained number of layers. In the case of R4 however, it did however show a small improvement in loss.

2.4.2 Batch normalisation

The second line of investigation was introducing batch normalisation. Frequently, in CNNs, the activation distribution characteristics can shift as they pass through each layer in the network. This shift is influence in part by the relu activation function, which only feeds through positive values. This may cause a covariance shift in the network resulting in either vanishing or exploding gradients. Batch normalisation provides a solution to this by normalising the inputs at each layer. The same batch sizes were trialled as in the previous batch test (see section

Result

The results of this experiment indicated a small improvement when compared to not using batch norm. in the best case of B1, the highest accuracy obtained was 72.4% and the best loss score of 0.80. Interestingly, the best performance loss by the final epoch was achieved by the largest batch size B3 with a final epoch loss value of 1 and a lowest loss value of 0.82. The batch normalisation procedure appeared to have a significant smoothing effect in the case of the larger batch size. A probable cause is that the impact of batch normalisation are felt more heavily when larger batch sizes are used as there is more data being normalised at each layer.

2.5 Mixup

Data augmentation strategies provide a useful method for encouraging generalisation. In augmentation procedures, a network is forced to learn invariant representations of the data set ensuring that despite class variations across split groups, objects can still be recognised. One such approach for improving generalisation capabilities of a network is mix up. Mix up takes two images and creates a new image which is a weighted combination of the image pairings. A Λ parameter decide how the pixel values are averaged out across images. This is sampled from a beta distribution and falls in the value range (0,1). This was employed to try and improve generalisation capabilities of the network in this investigation.

Results

The model exhibited a significant decline in performance upon being pretrained with mix-up for 10 epochs. The best accuracy in this instance was 9.7% and the best loss was 2.333. A probable cause for this was that the model required normal pre-training before mix-up was attempted. However, due to time constraints, this could not be investigated further.

2.6 Conclusion

In the final evaluation, model performance was evaluated using the test set. The best test accuracy achieved was at epoch 4 with a value of 72.5% accuracy (figure 9). Hereafter, the model appeared to overfit as the accuracy oscillated with a final epoch accuracy score of 71.75%. In terms of loss, the lowest lost value was on at epoch 2 at a value of 0.84 (figure 8). Past this point, the model was unable to make predictions and loss increase to 1.28.

The decision to test on the entire test set per batch of training was for deeper understanding since the number of epochs was not controlled for in hyper parameter tunning. This in turn revealed at which epoch the model no longer learnt any more useful information pertaining to the test set (between epoch 2-4). The best macro f1 score on the test set was 0.72 suggesting a reasonably strong ability to make distinctions between classes. One concern in the design process was that without implementing cross-validation, the validation scores would not be a strong indicator of performance on the test set. Nonetheless this appeared not to be the case as, as the accuracy, loss and f1 scores reported during validation were close in range to those in the final test evaluation.

Overall, this model exhibited a strong accuracy score and f1 scores despite restrictions in the number of layers. The loss value, however, was relatively large throughout the training and test procedure. This indicates a high error rate with regards to making prediction.

To conclude, a range of different strategies were explored throughout the duration of this project, a total of 6 hyper-parameters and mix up image augmentation. The key takeaways from this research are as follows:

1) The number of channels in shallow layers need to be large enough to pick up local information whilst remaining smaller than deeper layers to encourage increased depth of feature space
2) Kernel sizes in convolutional layers should increase with the network size in order to promote gradual high-level representation discovery
3) If learning rate is too high and used in conjunction with relu, a model could encounter the "dying relu problem"
4) Smaller batch sizes operate optimally when used in conjunction with smaller learning rates for this experiment
5) Regularisation using L2 is requires small values for λ when regularising a small network
6) Batch normalisation effects are felt more heavily when larger batch sizes are used
7) Mixup provides an interesting mechanism for encouraging generalisation; however, the network must be pre-trained in a regular fashion for this to take effect.

2.7 References

1 Lu, L., Shin, Y., Su, Y. and Karniadakis, G.E., 2019. Dying relu and initialization: Theory and numerical examples. *arXiv preprint arXiv:1903.06733*.

2 Douglas, S.C. and Yu, J., 2018, October. Why RELU units sometimes die: analysis of single-unit error backpropagation in neural networks. In *2018 52nd Asilomar Conference on Signals, Systems, and Computers* (pp. 864-868). IEEE.

3 Laurent, T. and Brecht, J., 2018, July. The multilinear structure of ReLU networks. In *International conference on machine learning* (pp. 2908-2916). PMLR.

4 Ng, A.Y., 2004, July. Feature selection, L 1 vs. L 2 regularization, and rotational invariance. In *Proceedings of the twenty-first international conference on Machine learning* (p. 78).

5 Loshchilov, I. and Hutter, F., 2018. Fixing weight decay regularization in adam.

6 Kandel, I. and Castelli, M., 2020. The effect of batch size on the generalizability of the convolutional neural networks on a histopathology dataset. *ICT express*, *6*

Illustrations and graphs

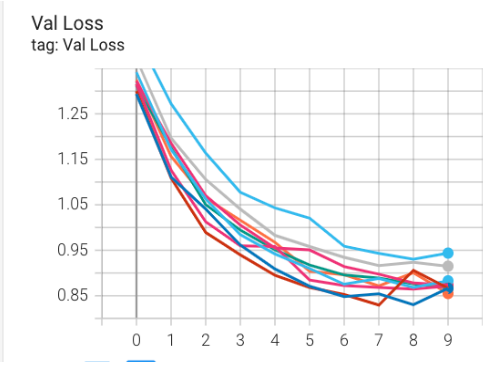

Figure 1: Depicting validation loss for convolutional channels and kernel size hyperparameter tunning. Worst case run (light blue), best case (red)

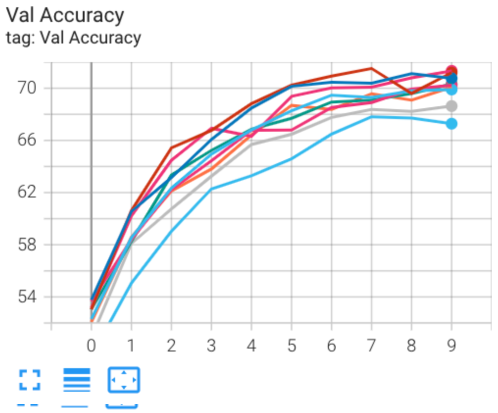

Figure 2: Depicting validation accuracy for convolutional channels and kernel size hyperparameter tunning. Worst case run (light blue),

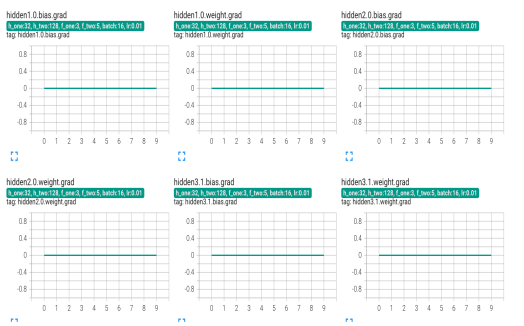

Figure 3: Depicting the weight and bias gradients when confronted with the dying relu (in the case of the highest learning rate)

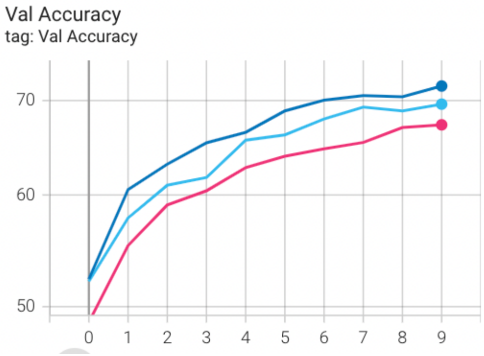

Figure 4: Validation Accuracy for batch size experimentation, batch size: 80 in pink, batch size:40 light blue, batch size:20 in dark blue

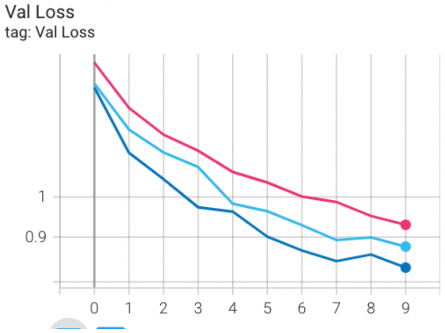

Figure 5: Validation loss for batch size experimentation, batch size: 80 in pink, batch size:40 light blue, batch size:20 in dark blue

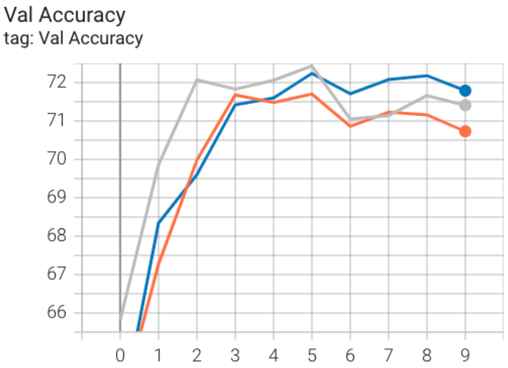

Figure 6: Batch norm with differing batch sizes, batch size 80 (dark blue), batch size 20 (grey) and batch size 40 (orange)

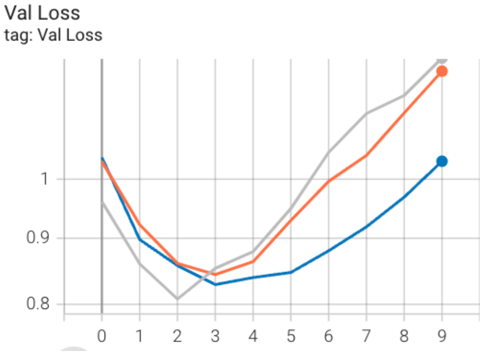

Figure7: Batch norm with differing batch sizes, batch size 80 (dark blue), batch size 20 (grey) and batch size 40 (orange). Graph depicting loss.

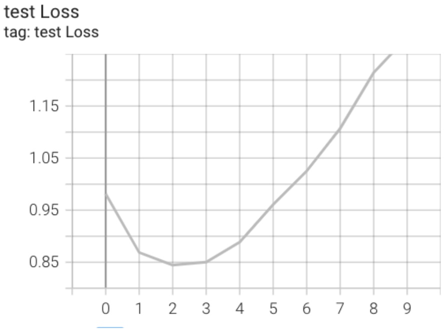

Figure 8 : graph depicting loss on final test set

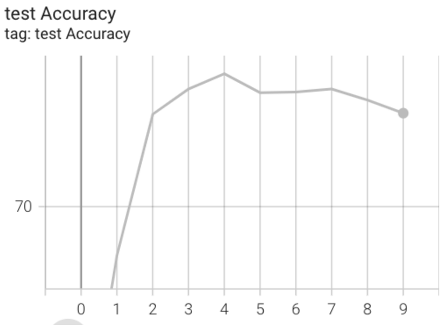

Figure 9: graph depicting accuracy on final test set