1 Introduction

In this report I use machine learning methods to perform object recognition on the CIFAR-100 dataset. The first, and more successful method, is a neural network. The second method is a convolutional neural network. The goal is to optimise these methods to the best of my ability and to compare the two methods. I had a lot more success with the neural network as opposed to the convolutional neural network, but I believe that is down to my own implementation.

2 Method

My approach for implementing both the neural network and convolutional neural network will be fairly similar. I will start by creating basic models which can then be built upon iteratively. I will test at nearly every step so I can better identify the variables that cause improvement. The fine labels will be used throughout implementation.

2.1 Neural Network

As the data set is in the shape *H x W x C x S*, it needed both flattening and reshaping to the form *S x C* to work with the dense (fully-connected) layers. The final bit of preparation before creating the neural network involved standardising the data using *sklearn.preprocessing.StandardScaler*().

2.1.1 Version 1

The first version of the model has 3 layers (2 hidden layers and 1 output layer). The hidden layers have 10 neurons each and the output layer has 100 (the number of classes). The first two use the rectifier activation function (ReLU) as it is very efficient computationally and is known to be good for simple neural networks (which this is). The final layer uses the Softmax activation function because the network is making multi-class predictions.

When calling the *compile*() method, I supplied it with the optimiser, loss, and metrics arguments. The optimiser I have chosen is Stochastic Gradient Descent as it often has a short convergence time[1]. My targets are class labels, so I used Sparse Categorical Cross-entropy to describe the loss to be used by the optimiser as it is a multi-class problem. This relates to the metric as I used Sparse Categorical Accuracy as it will help show the percentage of correctly labelled samples.

Next, I trained the model using the flattened data and fine labels. It has 10 epochs to keep processing time low and a validation split of 0.2 to give the data an 80-20 split of training and validation.

2.1.2 Version 2

The first version only had a test set accuracy of 11.17% which left a lot of room for improvement. In this second version I increased the number of neurons in both hidden layers to 200. This greatly increased the training time (to 155 seconds) but it did yield a much higher test set accuracy of 22.38%. The accuracy curves in Figure 2 show that the validation accuracy barely improved after the third epoch and so I can infer that the number of neurons can be reduced.

Reducing the number of neurons in each of the hidden layers to 100 cut the training time to 61 seconds whilst the test accuracy only dropped to 21.37%. Despite the reduced neurons, overfitting still occurred (Figure 1). In an attempt to fix this, I added two dropout layers.

The dropout layers had a rate of 0.2 as too high a rate could impact learning too heavily. The accuracy rate dropped to 19.33% but the accuracy curves were much closer together which means

that overfitting is now less of a problem. Removing one of the dropout layers improved accuracy by about 1%.

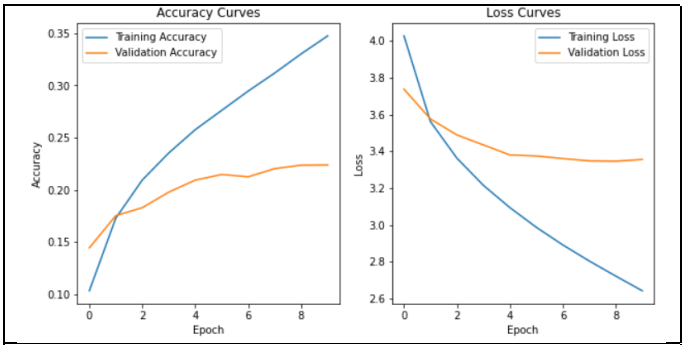

Figure 1

Another change I made was to double the number of epochs to 20. This doubled the training time but it improved the accuracy by about 2%. I tested increasing the number of layers and changing the optimiser but neither resulted in a performance increase.

The final test set accuracy was 23.37%. This was with 20 epochs and 2 hidden layers with 128 neurons each.

2.2 Convolutional Neural Network

Once again, I had to swap the axes in order to obtain the data in the format I wanted. I then normalised it before creating the model

2.2.1 Version 1

My first model was made up of 2 pairs of Conv2D and MaxPooling2D layers. The first Conv2D layer had 32 filters and an input shape of 32x32x3. Both Conv2D layers have a kernel size of 3x3 and ReLU. After these 4 layers is the flatten layer which makes it so fully-connected layers can process the data. I then put one dense layer with 256 nodes before the final output layer which is the same as the output layer in my original neural network.

When compiling the model, I once again used Sparse Categorical Cross-entropy, Sparse Categorical Accuracy, and Stochastic Gradient Descent for all of the same reasons as earlier. I trained the model

All of this resulted in a test accuracy of 1% which is a very poor start. The sparse categorical accuracy did not change in any of the epochs.

2.2.2 Version 2

To improve on the original, I changed the optimiser to Adam [3] and set the learning rate to 0.00001 [4] and added a dense layer with 256 neurons. The test set accuracy improved to 2.91%.

I then tried decreasing the learning rate further, but this worsened the accuracy, so I changed it back to 0.00001.

I added in another pair of Conv2D and MaxPooling2D Layers and this helped raise the accuracy to 5.42% but unfortunately this was the highest I achieved.

3 Results

3.1 Neural Network with Fine Labels

Despite being extremely difficult to analyse in detail due to the large size (100x100), I believe that this confusion matrix in figure 2 is still of use. The colourful diagonal line through the middle shows that images were frequently assigned to the correct class. This holds as the test set accuracy was 23.37%.

The accuracy curves in figure 3 show that the network was slightly overfitted and thus could be made more efficient.

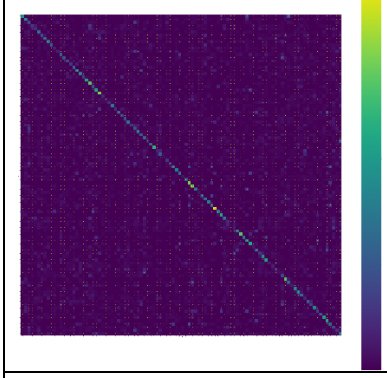

Figure 2

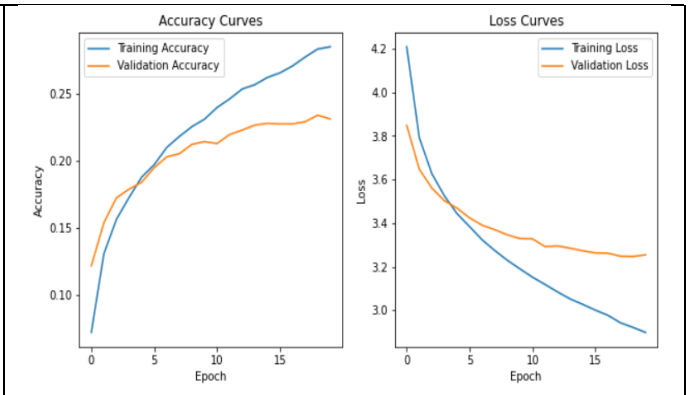

Figure 3

3.2 Neural Network with Coarse Labels

The test set accuracy was 34.58% and I believe this is represented in the confusion matrix (figure 4) as there is a strong diagonal line through the middle. There is some visible symmetry in the matrix which suggests that certain classes were frequently misidentified as one another. The class with the most correct positives was class 17 (trees).

The accuracy and loss curves (figure 5) are almost identical in shape to the curves for the fine labels.

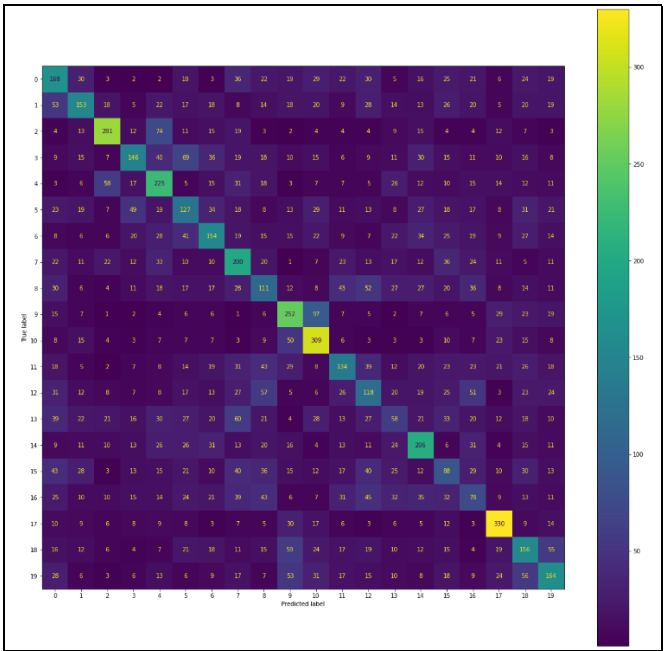

Figure 4

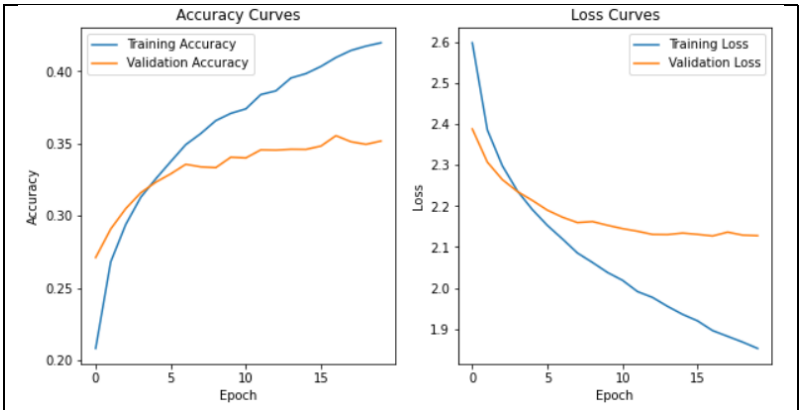

Figure 5

3.3 CNN

The final test set accuracy for the convolutional neural network with fine labels is 5.42% which is much lower than the accuracy achieved using a neural network.

The final test set accuracy with coarse labels is 12.53% which isn't terrible but once again it is much lower than the result received from the neural network.

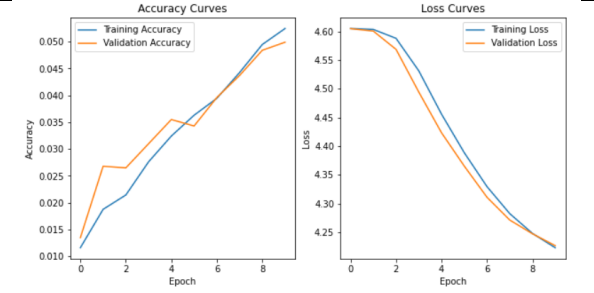

Figure 6

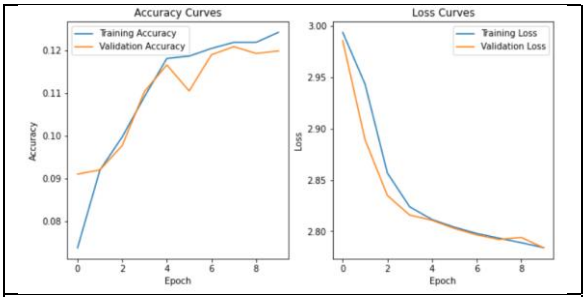

Figure 7

4 Conclusion

Reflecting on this piece of work, I believe that my overall idea for my method was good but that it could have been improved in a variety of ways. For example, I could have made a grid to determine the optimum number of neurons or layers for my neural network. In addition to this, I think there was a big problem with my implementation of the convolutional neural network which led to poor results for both labels. Unfortunately, I have been unable to identify exactly what the issue was. However, the accuracy curve might suggest that more layers were needed.

As expected, the accuracy for the coarse labels was higher than for the fine labels (for both methods). My results show that my implementation of a neural network was much more effective than my implementation of a CNN but I might have to put this down to a personal fault rather than an issue with convolutional neural networks.

References

[1] Kandel, I., Castelli, M., & Popovič, A. (2020). Comparative Study of First Order Optimizers for Image Classification Using Convolutional Neural Networks on Histopathology Images. *Journal of Imaging*, *6*(9), 92. doi:10.3390/jimaging6090092

[2] Brownlee, J. (2018). How to Avoid Overfitting in Deep Learning Neural Networks. Retrieved 9 December 2021, from https://machinelearningmastery.com/introduction-to-regularization-to-reduce-overfitting-and-improve-generalization-error/

[3] Stewart, M. (2019). Neural Network Optimization. Retrieved 9 December 2021, from https://towardsdatascience.com/neural-network-optimization-7ca72d4db3e0

[4] Ramesh, S. (2018). A guide to an efficient way to build neural network architectures- Part I: Hyper-parameter…. Retrieved 9 December 2021, from https://towardsdatascience.com/a-guide-to-an-efficient-way-to-build-neural-network-architectures-part-i-hyper-parameter-8129009f131b