22COC102 - Advanced Artificial Intelligent Systems

Student ID: B927574

Part A

In this project, TensorFlow was used to construct various Convolutional Neural Network (CNN) and Residual Neural Network (ResNet) configurations. Numpy and matplotlib were used for all graphs bar the confusion matrix which instead used sklearn. Therefore, dependencies include the installation of Python (3.7 or later), TensorFlow (2.11) and sklearn libraries.

Architecture Changes

For the CNN, the example from [1] was used as a foundation. My changes to the model architecture between configurations include the addition of convolutional (and accompanying pooling) layers to increase the model to a five layer network, changes to filter sizes specified for the convolutional layers and first dense layer, and the addition of a dropout layer between the dense layers. All these changes needed to be made to the variable 'model.'

I modified a ResNet34 example [2] and created an 18 layered ResNet (ResNet18) to reduce the computational load. This required a change to the 'block_layers' variable ([3,4,6,3] to [2,2,2,2]) within ResNet34() (changed to ResNet18()) which is responsible for determining the size of sub-blocks. Furthermore, I changed the loss function when compiling the model to match the CNNs' ('categorical_crossentropy'). The activation functions for each layer type remained consistent across architectures. A dropout layer was added in some configurations, again between dense layers (ResNet18()).

TensorFlow's 'EarlyStopping' function was included in some configurations. Once defined, this function is passed as a callback parameter when training.

Metrics

I decided to evaluate performance via accuracy, loss, recall, precision and the Area Under Curve (AUC) which TensorFlow allowed me to specify as metrics to gather when compiling the model. TensorFlow 2.11 does not directly calculate F1 Score so this was done manually afterwards.A Graphical display of the training loss and validation loss against epochs was created using Numpy, as well as the confusion matrix via sklearn.

Part B

Please refer to Table 1 for the details of each configuration. Table 2 shows the performance of the models across accuracy, loss, AUC, precision, recall and F1 score. Highlighted are the best values for that metric (architectures are considered separately).

Since CIFAR-10 is a balanced dataset, accuracy is an appropriate metric to compare the models by. The best accuracy result came from CNN-4, a three layer CNN of filter sizes 32, 64, 128 (dense layer filter size 128) with early stop and a dropout layer included. All models had high accuracy (over 0.94), but overfitting was a problem. Figure 1 and Figure 5 demonstrate this clearly (ResNet18 with a bit more instability) with the quick plateauing and increase in the validation loss over epochs as the training loss continues to decline. I therefore looked to minimise this issue with the following configuration changes, CNN-4 implementing all of these.

Reducing the amount of layers and filter sizes can help prevent overfitting as it minimises the complexity so there are less features that can be memorised from the training data, therefore forcing the model to prioritise learning the most important features. This also adds regularisation to the model, preventing the model fitting noise or irrelevant features of the data. A dropout layer will cause the layer above (the first dense layer) to randomly drop nodes, half in this case. This stops the model from relying too heavily on certain nodes and ensures that individual nodes do not become too highly specialised in certain inputs. It also introduces noise to help the model gain a more general understanding of the data. Early stopping is used to stop training the model once it begins to overfit. If the validation loss does not improve within a specified amount of epochs, training terminates and the best weights are restored.

This combination suited the CNN model, with the loss value improving with the introduction of dropout and early stopping with a lower loss value means the network is better at making accurate predictions. ResNet18 struggled in this metric with ResNet-3 performing the best but at the cost of accuracy.

AUC shows all models to be performing similarly (closer to 1 than 0.5) indicating that predictions were not random guesswork. Precision and recall were used to calculate an F1 score since these two metrics exist in contention with each other. CNN-4 scored highest showing it performed reliably as a classifier. However at 0.7579, this could still be improved.

Overall, in these configurations, CNN-4 achieved the best results with Figure4 showing an improvement in overfitting. ResNet18 took more time to train due to its complexity with loss failing to reduce sufficiently. It could neither outperform CNN-4's accuracy nor reliability. Perhaps changes to strides, batches and the loss function would have benefitted this architecture more.

Table 1: Configurations

Name	Architecture	Layers	Early Stop?	Dropout?	Epochs
CNN-1	CNN	5; Conv filter sizes: 32, 64, 128, 256, 256; Dense layer filter sizes: 256, 10	No	No	10
CNN-2	CNN	3; Conv filter sizes: 32, 64, 128; Dense layer filter sizes: 128, 10	No	No	10
CNN-3	CNN	3; Conv filter sizes: 32, 64, 64; Dense layer filter sizes: 64, 10	Yes, allowance = 3 epochs	Yes, 0.5 dropout in effect on the first dense layer	17
CNN-4	CNN	3; Conv filter sizes: 32, 64, 128; Dense layer filter sizes: 128, 10	Yes, allowance = 3 epochs	Yes, 0.5 dropout in effect on the first dense layer	12
ResNet-1	ResNet18	18; Starting filter size: 64	No	No	10
ResNet-2	ResNet18	18; Starting filter size: 64	No	Yes, 0.5 dropout in effect on the first dense layer	20
ResNet-3	ResNet18	18; Starting filter size: 64	Yes, allowance = 3 epochs	Yes, 0.5 dropout in effect on the first dense layer	9
ResNet-4	ResNet18	18; Starting filter size: 32	Yes, allowance = 5 epochs	Yes, 0.5 dropout in effect on the first dense layer	15

Table 2: Metrics

Name	Accuracy	Loss	AUC	Precision	Recall	F1 Score
CNN-1	0.9506	1.0638	0.9482	0.7653	0.7295	0.7470
CNN-2	0.9472	1.1396	0.9425	0.7485	0.7113	0.7294
CNN-3	0.9522	0.7823	**0.9637**	0.8102	0.6819	0.7405
CNN-4	**0.9547**	**0.7448**	0.9471	0.8144	0.7088	**0.7579**
ResNet-1	0.9501	0.9360	0.9542	0.7662	0.7209	0.7429
ResNet-2	**0.9508**	1.3056	0.9351	0.7622	0.7377	**0.7498**
ResNet-3	0.9469	**0.8474**	**0.9582**	0.7889	0.6406	0.7071
ResNet-4	0.9473	0.8852	0.9553	0.7760	0.6654	0.7165

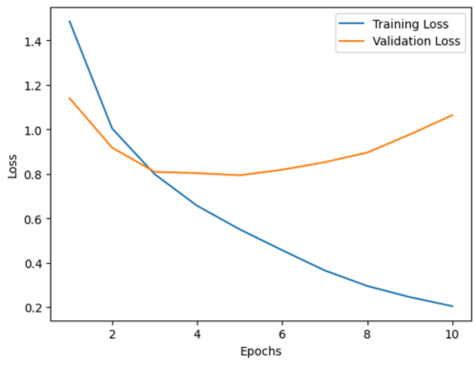

Figure 1: CNN-1

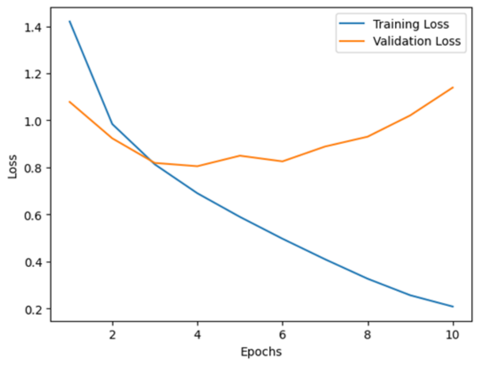

Figure 2: CNN-2

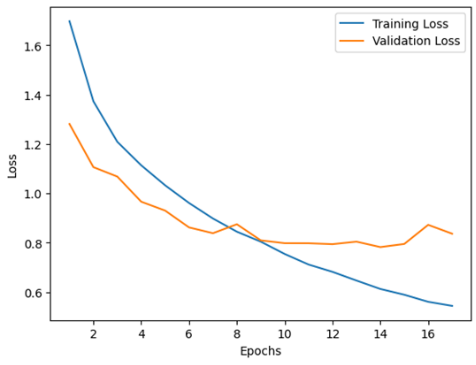

Figure 3: CNN-3

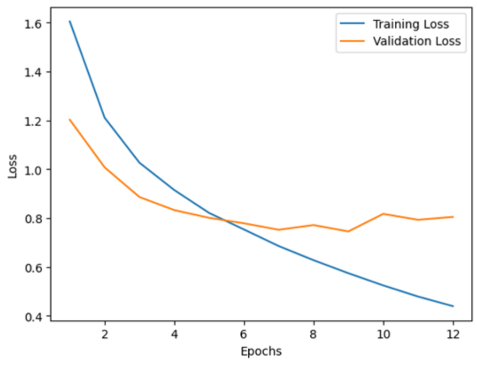

Figure 4: CNN-4

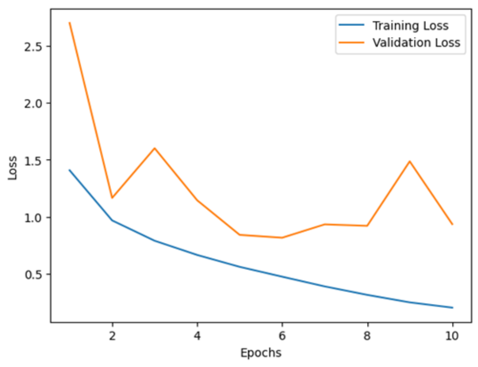

Figure 5: ResNet-1

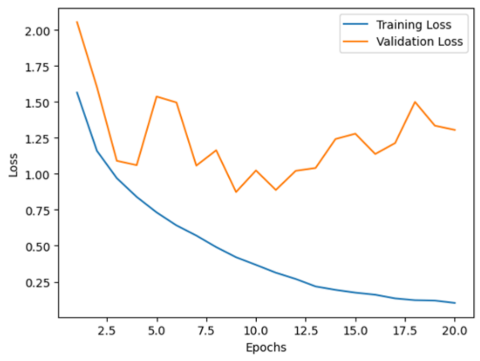

Figure 6: ResNet-2

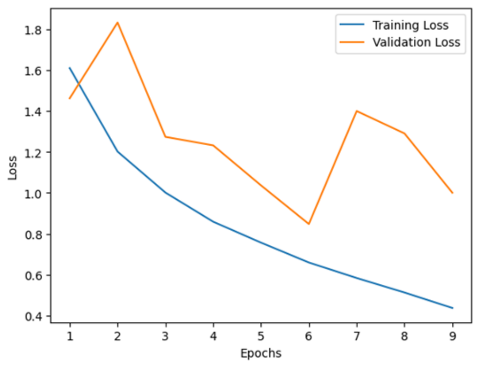

Figure 7: ResNet-3

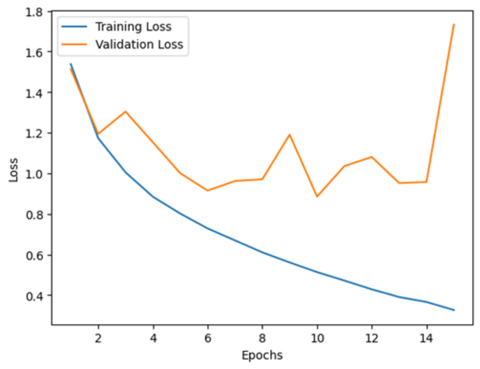

Figure 8: ResNet-4

References

[1] TensorFlow *Convolutional Neural Network (CNN)*[online]. TensorFlow tutorials, Last updated 2022-12-15 UTC [cited 20/03/23]. Available from World Wide Web: (https://www.tensorflow.org/tutorials/images/cnn).

[2] Yashowardhan Shinde, 2021 *How to code your ResNet from scratch in Tensorflow?*[online]. Published August 26 2021, last modified September 7th 2021 [cited 20/03/23]. Available from World Wide Web: (https://www.analyticsvidhya.com/blog/2021/08/how-to-code-your-resnet-from-scratch-in-tensorflow/).