20COC102 - Advanced Artificial Intelligent Systems

Part A

Setup

To begin working on this project, I downloaded Anaconda and started installing all the libraries that I would be using, including numpy and matplotlib, and then had to decide whether I wanted to use Pytorch or Tensorflow.

I started by installing Pytorch and using the MNIST dataset. However, after a few hours of exploring and using different models for this data, I realised that it was too simple for me to make meaningful comparisons between models if they are achieving over 90% accuracies regardless of changes. Therefore, after doing some more experimenting, I decided to change my dataset to CIFAR-10, and while I was doing this, I also decided that I was going to use Keras with Tensorflow as it seemed easier to use.

With all my libraries and environments prepared. I began looking online for tutorials to get started. The tutorial I used was on the Tensorflow website [1] (Convolutional Neural Network (CNN) | TensorFlow Core, 2021).

This source introduced me to the simple data loader and data normalisation that can be seen and cited in the first block of my code. In this section I also split the data into two sets, the training set, and the testing set. The training set will be used to train my models and then the test set will be used to validate the accuracy of the models on data that they have not yet seen.

This source also introduced me to the technique used to compile and train my model, then evaluate its final accuracy.

First Model

With the basic code layout sorted, I had to start building my two architectures to compare. For my first model, I used the AlexNet as inspiration for the basic structure (source was from the week 4 lecture slides), but I had to greatly simplify and minimise both the filter sizes and the kernel sizes. This was to greatly reduce the training time and to make the model more fitting to the CIFAR-10 dataset. The model ended up having the same sections of filters, and the sizes still had similar ratios as the original, but the filter sizes were half or smaller than half their original size. The same can be said for the fully connected layers, where the original had sizes of 4096, mine now has sizes 128 and 256. This was one of the biggest changes to speed up the training time.

When I simplified the AlexNet model, I struggled with big overfitting problems. The model would train to accuracies of 80% to 90% but would be only about 60% accurate when tested on the validation set. To fix this, I did some research and found a very helpful source that suggested many different approaches I could take [2] (Schlüter, 2021).

To start, I added Batch Normalisation layers after each layer. This was to improve the quality of the training by normalising the output of each layer to remove any biases or influence different parameters may have imposed on the data.

Next, I added dropout layers between each of the fully connected layers. This helps by “turning off” some of the nodes in the neural net to stop any one node being too influential on the network. This helps the model have a more generalised view of how to classify data instead of looking for small individual bits of information.

With the drop out layers, I played around with the dropout rate to see what was best. I found that 0.4 was good but reduced the overall accuracy too much. 0.2 was more accurate but hadn't closed the gap between accuracy and val_accuracy enough. I found that 0.3 was the sweet spot and went on to use this for the model.

After this, I decided to add a decaying learning rate to the optimiser to reduce loss near the end of the training stage. I decided to customise the Adam optimiser with an initial learning rate of 0.001, decay steps of 10000 and a decay rate of 0.99. As well as this, I settled on using Sparse Categorical Crossentropy as my loss function.

With all these additions made to the model, I finally achieved a validation accuracy of 77% on my first train and test. For the sake of preserving report space, you can see the final structure of my model within the submitted Jupyter Notebook, and further details in part B of this report.

Second Model

For the second model, I decided to go with a very simple architecture to contrast with my complex model. I settled on using just two convolution layers and just one fully connected layer (not including the final classifying layer). I played around with the sizes of the filters until the accuracy improved. As well as this, I also added a max pooling layer and batch normalisation after each convolution layer.

These minor additions to a simple model really helped improve the accuracy, and I now get 68% to 71% validation accuracy. However, the gap between the training accuracy and the validation accuracy is really large, and so I thought it may be beneficial to add a dropout layer after the fully connected layer.

While this dropout layer didn't greatly increase the validation accuracy, it did greatly reduce the difference between the training accuracy and the validation accuracy.

Graphs

The tutorial mentioned previously in this report (Convolutional Neural Network (CNN) | TensorFlow Core, 2021) also introduced me to the technique used to visualise the model progress on a graph using matplotlib. I used these graphs to show my model's accuracy over time, and to help visualise any overfitting. This was very useful as it made fine tuning easier for me. I took this basic guide from the source and adapted it to make my graphs show both models on a single graph, and to have graphs for other metrics.

Bibliography

[1] TensorFlow. 2021. *Convolutional Neural Network (CNN) | TensorFlow Core*. [online] Available at: <https://www.tensorflow.org/tutorials/images/cnn?fbclid=IwAR2N3-3QOzwejFV-Bj0jpULwzI4M3GHBzNcy3HhkmTbXM7rSmLNPa-weYKY> [Accessed 10 March 2021].

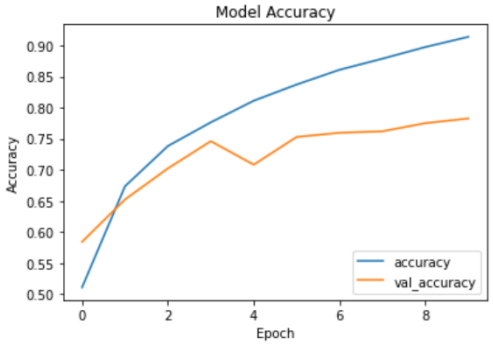
[2] Schlüter, N., 2021. *Don't Overfit! — How to prevent Overfitting in your Deep Learning Models*. [online] TowardsDataScience. Available at: <https://towardsdatascience.com/dont-overfit-how-to-prevent-overfitting-in-your-deep-learning-models-63274e552323> [Accessed 14 March 2021].

Part B - First Model - As stated in Part A of this report, my first model was created with AlexNet as inspiration for the overall structure and design. However, the original AlexNet has very large filter sizes and kernel sizes, and so takes a long time to train. Therefore, I spent a lot of time simplifying the model to suit the CIFAR-10 dataset, and to get the training time down to a reasonable time. The biggest changes I made are that the kernel sizes are now all 3x3 and the filters are a fraction of the original sizes.

The first Convolution layer originally had filter of size 96 and a kernel of 11x11, but I reduced this kernel to 3x3 as the model was looking at an image of size 32x32, so the original kernel wasn't picking up enough fine detail. This layer has the Relu activation function performed on it, and then is passed to a max pool of size 2x2. The original max pool also had a stride of 2, which created an 8x down sample result. I removed this stride size as the small input images didn't need to be down sampled any more, as they could start losing detail.

This is then passed to the next convolution layer, which originally had a filter size of 256 and kernel of 5x5. As mentioned above, I changed the kernel to size 3x3 and the filter size to 128. Lots of trial and error led me to find this filter size to be best for now, as it was still large enough to be useful in the CNN but still reduce the training time. The same trial and error led to the final three convolution layers being given filter sizes of 128, 64 and 64 respectively. I found that if these filters were any larger, the training time gets closer to an hour, and if they get any smaller the accuracy is sacrificed dramatically.

Finally, the fully connected (dense) layers at the end originally had size of 4096. This was a big factor to the slow training time, and so I tested with gradually smaller sizes until the training time was acceptable, and the accuracy was still reasonable. I found that the best result was achieved by having the second dense layer larger than the first, with batch normalisation in between. I settled on having them at sizes 128 and 256 respectively.

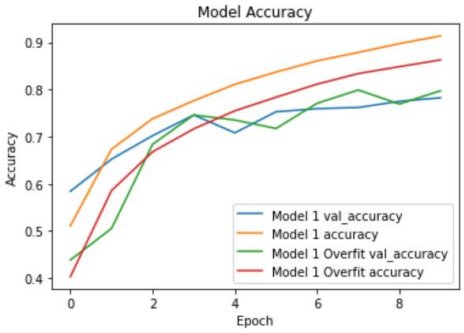


When run, this model performed well and achieved 91% training accuracy and 78.3% validation accuracy.

When I was simplifying the AlexNet model, I tried other ways of simplifying it, including removing some convolution layers, or removing dense layers. This almost always resulted in a model that could achieve very high training accuracy of around 80% to 90%, but the validation accuracy couldn't get past 60%. This showed that I was struggling with serious overfitting, and so did some research to find what I could do to fix this. As mentioned in part A, I found a source [2] (Schlüter, 2021) that suggested a few things I could add to the model.

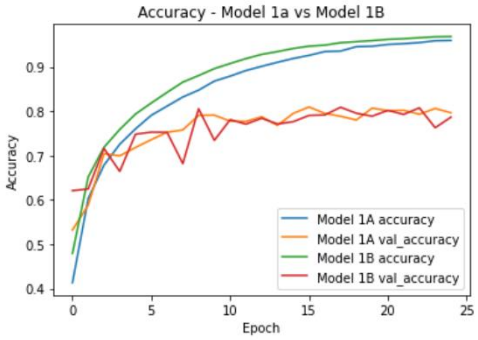
Thanks to this source, I became familiar with adding dropout layers, batch normalisation and even a decaying learning rate. However, when I added these to the model, it would still struggle to get past 65% validation accuracy. Therefore, I changed how I was simplifying the model, and ended up with the solution that I now have.

However, thanks to this learning about overfitting, I thought to try adding these dropout layers and decaying learning rate to the new model. This reduced the gap between the training accuracy and the validation accuracy, but had little effect on the validation accuracy itself:



As you can see, the blue and orange lines show the model overfitting as there is a large gap appearing at the later epochs. The red and green lines show that, with the dropout layers and the other additions I mentioned, the gap is reduced significantly. However, the overall validation accuracy hasn't improved too much, achieving around 79.7%. This could allow for the validation accuracy to continue improving over more epochs in the future.

Because of this, I decided to alter the number of epochs that the model is trained for and the value used for the dropout layer. With these parameter changes we will be able to see if the model continues to overfit, and if so if it is preventable.



Model 1A – 25 epochs and 0.4 dropout value - With this high dropout value and extra training time, we expect to see less overfitting but perhaps a lower overall accuracy. After running this altered version of the model, this was the result:

The blue line shows that the training accuracy reached 96%, and the orange line shows that the validation accuracy reached 79.6%. This difference isn't too large but seems to get worse at around the 10th epoch. Surprisingly the overall accuracy wasn't affected too badly. The extended number of epochs didn't result in much more added accuracy, as it seems the model begins to plateau between epochs 10 and 25.

Next, I will try the same number of epochs but with a lower dropout value to see if the gap grows, or if the accuracy improves.

Model 1B – 25 epochs and 0.1 dropout value – Looking at the graph above, the green line shows that the training accuracy reached 96.8% and the red line shows that the validation accuracy reached a max of 80.7% and finished at 78.6%. Surprisingly the difference in the dropout rate is very minor, if not negligible. This is likely because there are other factors within the model that means it cannot reach more than about 80% accuracy without major changes to parameters like filter size and overall structure. As well as this, we can once again see the additional epochs haven't provided any benefit to the model's accuracy, as it begins to plateau again after the 10th epoch.

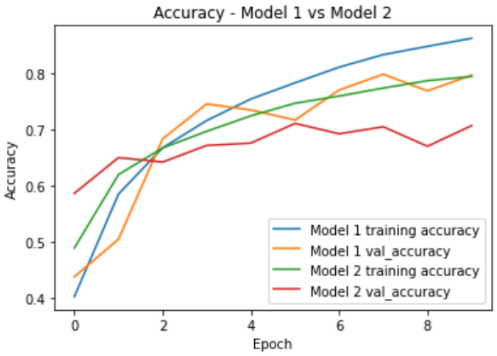
From these observations, we can speculate that this model cannot reach validation accuracies over 80% without major alterations to the core structure and design of the model. Seeing as I simplified the model to a point where I thought the training time was minimal, I see this accuracy as acceptable for now. If I had more time and continued to explore this model, I could possibly find what the next "roadblock" is for it to progress past 80% accuracy – possibly involving adding more layers and therefore increasing the training time.

Second Model - Again, as partly mentioned in part A of this report, I chose to create a very simple architecture for the second model so that I can make comparisons between a complex model and a simple one.

I did some experimenting with structures that were the bare minimum, using just a single convolution layer and a single fully connected (dense) layer. This didn't provide very high accuracy, so I added another convolution layer, and then gave each convolution layer a max pooling layer of

size 2. I decided to also add batch normalisation layers after the max pooling, and a single dropout layer between the fully connected layer and the classifying layer. This is to hopefully improve the training accuracy, and then bring the validation accuracy up with it. The result was a training accuracy of 79% and a validation accuracy of 70.7%.

The architecture is still very simple, with the two convolution layers having filter sizes of 32 and 64 respectively, and the dense layer having a size of 64. However, such a simple model still has an accuracy close to that of the very complex model I created inspired by the AlexNet.



As you can see from the graph, the orange and red lines show a very different shape and trend of the validation accuracies of models 1 and 2 respectively. Model 1 starts low but quickly learns over 2 epochs, then begins to level out. Whereas model 2 starts quite high but doesn't increase too much over the 10 epochs. The final accuracies only differing by less than 10%, but with training times over 15 minutes apart.

The blue line shows that model 1 can train to very high accuracies of around 88%, and the green line shows that model 2 can train to accuracies just shy of 80% - again a difference of less than 10%. From this we can speculate that if we altered just a few parameters of model 2 we could get it to perform at a similar standard to model 1, as it currently seems to consistently be 10% less accurate. Therefore, I will experiment with changing the filter sizes of the convolution layers and increasing the size of the fully connected layer. As well as this, I will be training these altered models over 25 epochs to see how the accuracies improve over extended time, and whether they plateau at all.

Model 2A – 25 epochs, double filter sizes and double FCL size – The graph shows that surprisingly there is a severe amount of overfitting happening, with the blue and orange lines showing the model training to an accuracy of 95% but reaching only 73% validation accuracy. This gap of over 20% shows that there is a real issue of overfitting that could be potentially aided with more dropout layers, however it is possible that the model is just limited to this accuracy due to its simple architecture. In the next alteration, I kept the filter sizes the same but continued with the doubled size of the fully connected layer. This is to try and isolate whether this overfitting is due to the filter sizes, or the FCL.

Model 2B – 25 epochs, original filter sizes and double FCL size – The graph shows yet again a lot of overfitting is taking place. The green and red lines show a training accuracy of 92.7% and a validation accuracy of 71.3%. With the original sized filters but a double sized fully connected layer, we can conclude that it is not the filters that are causing this overfitting or underperformance, but it is the simplicity of the model. If we had more convolution layers or more fully connected layers, we would likely see much better performance. There would also be more opportunity to have dropout layers, and therefore more opportunity to reduce overfitting. However, if we made these changes, the model would strongly resemble that of the first model inspired by AlexNet. Therefore, we have found the limitations of such a simple model and shown that a more complex one is needed for this dataset and task.