CIFAR 10 transcript

The CIFAR-10 data

The CIFAR-10 data was downloaded using the keras package and it’s an image dataset of 60,000 images. We decided to split the dataset into a training dataset of 50k images and a validation set of 10k images

The rationa;le for splitting into a training and validation dataset its that neural networks are prone to over-fit or memorise the training data so the performance of the model must be assessed in validation data that the network has not seen during training

The data structure. The shape of each data instance was 32x32 pixel by three channels which is rgb. The values were normalised by division by 255 putting the values in the range 0-1. There were ten classes or labels of the images

Approach to model building experiments

In terms of the approach to model building, first we trained an ANN. The aim was to test the performance of the network without any convolutional layers and the rationale was to give a baseline performance for comparison

Next we tried adding conv layers to build a conv neural network. Here we aimed to test the change in performance with a small number of conv layers and then to viz the kernels and the feature maps learned for a particular image and this helps to interpret how the network is learning.

In the third phase we build a deeper cnn with many layers including features such as dropout and batch normalisation to optimise performance on the image classification task. For all these model building experiments we used the python package keras.

Modelling choices

We used a loss function of sparse categorical cross entropy loss which is appropriate for a multi class classification objective. As I said, in this class there were ten different labels.

For metrics, we used accuracy, which was appropriate for this balanced multi-class problem. However we noted that if the classes were imbalanced this would not have

been appropriate and other metrics such as precision, recall and an F1 score would be more appropriate.
For the optimizer, we used Stochastic gradient descent and empirically selected ten epochs as the models seemed to plateau after ten epochs.

ANN: from one to three FC layers

For the first experiment with an ANN from one to three FC layers. On the left we can see a single FC layer with ten neuros with 30k parameters. Over ten epochs the training accuracy increases to about 37% but the validation accuracy is quite poor less than 30% and doesn't improve
On the right we test the addition of two more fc layers,100 then 50 and finally 10 neurons, and we see that the accuracy is very similar to the single layer and maybe marginal improvement in the validation accuracy.
The conclusion here is that the ANN is not performing well on the image classification task, and so we need to try a convolutional neural network instead

CNN: one convolutional layer

The first experiment on the CNN used one convolutional layer with 50 kernels in the layer. The kernel size was (3,3), the stride = 1, the activation = ReLU, the optimizer was Adam. Adam is an extension of SGD that maintains per parameter learning rates. This is good for problems like computer vision with sparse gradients. It also adjusts the learning rate based on the recent changes in the weight for each parameter. The metric = accuracy, and one fully-connected layer was used for classification.
The number of parameters has risen to 500k. Whilst it does well on the training set, and could be said to memorise the training set, reaching accuracy of 90%, it does relatively poorly on the validation set and does not improve much after the first couple of epochs, around 40-50%. We note the large number of parameters in the FC layer, as the output of the convolutional layer is 50k parameters onto the ten neurons.
Here we visualise the feature maps and kernels for this one convolutional layer network. On the left the feature maps for this image of a cat show it has learned

various patterns. Such as, change in background (blue vs. yellow parts), horizontal lines, and vertical lines.
On the upper right we can see examples of 3,3 kernels that the layer has learned.

CNN: 3 convolutional layers

In the next experiment, we tried three convolutional layers. With 50, 100, then 200 kernels, as empirically, many models double the number of kernels from layer to layer to learn higher order and more complex patterns. The kernel was (3,3), stride 1. We tried max pooling layers size (2, 2) to downscale the number of parameters in the kernels. We tried 3 FC layers: 40, 40, 10. Activation, Optimizer, and Metric stayed the same.
For this network we saw a decrease in parameters from the one layer convolutional network to three layers with max pooling, to roughly 350k parameters. This is due to the downscaling of their feature maps due to max pooling.
Performance was improved and after three to four epochs, the validation accuracy was around 65%. This suggests that with more convolutional and dense layers we can improve performance of the model.
So now we're going to go through each convolutional and max pooling layer of the network to look at the activation maps learned. In this first convolutional layer, again for this image of a cat, we can see that it is learning the difference between the background of the image, horizontal and vertical lines. The max pooling layer down-scales the activation map to show coarser features. In the second convolutional layer, the features learned are of a higher order and some of the activation maps are blank. Max pooling again coarsens the features. At the third convolutional layer, we can see some activation maps are blank suggesting they are specialised to pick up certain features which are not present in the example image.

CNN: adding dropout

Finally, we tested adding a dropout layer to the network. Here, with five convolution layers, with kernels of size (3,3) and max pooling layers of size (2,2), we add a dropout layer before the FC layers, with a dropout rate of 0.3. This means that during training, 30% of the inputs to the FC layers are silenced, but during testing on the validation set, all inputs are used.

We see 800k parameters were learned. The experiment demonstrated that adding dropout narrowed the gap between the training and validation accuracy, which was the expected effect as dropout aims to regularise the neural network. Validation accuracy approached 70% after seven epochs, which is an improvement on the previous network.

Performance optimisation

For high performance we have chosen a CNN model which is usually used for image processing and similar classification problems. We wanted to expand on all of the learnings that we got from the try and fail process that Rory has described in the previous section. The convolution in the CNN extracts features from the image by processing pixels. Convolution is the primary layer of our CNN model and all of this, almost all of these layers, use the rectified linear unit function for activation which replaces negative values with zero in order to achieve nonlinearity in the model. After convolutional layers we applied batch normalisation layers, which standardise the input for each mini-batch hence stabilise the learning process and decrease the number of epochs required for meaningful training. You can see on the left side the number of epochs was also one of the parameters we played with and I'm going to talk about that in a little bit more detail on some of the latest slides. We also used pooling in order to downsample the spatial input dimensions which is yet another layer that helps to decrease computational complexity hence improving the learning process of the algorithm and it helps to prevent overfitting. Another layer we used to prevent overfitting would be dropouts which we applied and it randomly sets input units to zero and then drops a specified fraction of the units. And now flattening layers that we used, well, kind of had to use, allow for transition between convolutional and dense layers by unpacking 3D output into 1D. Dense layers then take that and connect neurons between the last layers of the model with the very last one using softmax activation function instead of the rectified linear unit I mentioned before. And that function gives a probability score for each of the classes and hence why this is used in this task of image classification. We used the standard SGD optimizer, set the learning rate at 0.001 and momentum at 0.9 - this was also set empirically based on the previous models' results. At the try and error approach. The setting allows for slow but steady and meticulous learning rate while momentum helps the algorithm to move gradient vectors in the right directions which helps with better convergence. So this is the overview of some of the tuning that we've done on the model to optimise for performance.

Model's structure

Here we can see a quick screenshot of what the model exactly looks like, we can see the convolutional layers, we can see the normalisation pulling and drop out layers as I described in the previous slides.

Performance metrics

Here we have highly balanced classes which I think this slide only proves but I wanted to touch on a couple of metrics that I usually use for assessing the performance of this model and then we have precision, recall and F1 score. Where Precision answers essentially the question out of all of predicted positive values how many were actually positive; recall is slightly different it answers the question out of all the actual positives how many did we predict correctly and then the F1 score is averaging it it's the weighted average of precision and recall which is trying to find a balance between between these two metrics it's often used when dealing with imbalanced classes which is not the case here. Nevertheless we wanted to show that this is one of the metrics that could be used especially when dealing with unbalanced classes.

Confusion matrix interpretation

This confusion Matrix summarises the positive prediction the correct prediction on the diagonal this is a tabular representation where each row represents actual classes and each column indicates the predicted classes and obviously the higher diagonal values show that what they indicate a better performance and on a quick note we can see at the classes that were used and where the biggest confusion lays for example class five and three cat and dog the only one over hundred mistakes. Where these two could be quite similar or number 2 with number 6 which is bird and the frog. Well, for humans it might not be that similar but for a machine - size wise and feature wise apparently it was high enough to to have a relatively high margin of error.

Epochs, learning and validation

Now we wanted to talk briefly about the number of epochs with chosen them empirically but also we had in mind the computing resources at hand this particular model was trained on a distributed cluster using 40 gigabytes GPU and eight computing units that we had available the model training time was 9 hours in total. Each hundred epochs with a batch size of 664 took around 3 hours to run. We can see that the learning curve for both training and validation starts to seriously flatten around 280-300 epochs. In order to prevent overfitting but also keeping in mind resources available we took a conscious decision to stop at 300 epochs. We assume that adding another 50 to 100 epochs could improve the precision by another 2-4 percentage points. However the costs versus benefit ratio was too low at this point to keep increasing the number of epochs. We can see on the slide that the training accuracy improves very similarly to validation accuracy which is also a proof for us that we are not overfitting the model.

Conclusions

Now to the conclusions of this presentation. The first and probably most important learning we took away is that CNN performs better than ANN for image data we also had loads of learning about I think different layers such as pooling, dropout, batch normalisation and all of them touch on a different aspects of the algorithm where either they standardised the input or stabilise the learning process or help to prevent overfitting which is we discovered many of them and the actual parameters that we need to set empirically and I think it was a great great experience seeing how changes actually impact the output of the model. Then the visualisation of the feature maps of the CNN that shows a lot and gave us also a lot of insights into how the the neural networks algorithm actually work and what they look like it was tremendously helpful in setting up an optimising the model’s performance and also we were able to get relatively good training results with small number of epochs as presented in the first part by Rory but then getting to this to a truly truly good results required a lot of epochs apart from hyper parameter tuning and and a couple of different changes. So to some all of this up I'd say that creating a well-performing model requires a lot of skill as well as time and resources because training on this amount of data with this number of layers that we had to use and different parameters like learning rates or optimiser it just takes a lot of computational resources and I'm confident and happy with the result that we got was the best that we could achieve with the resources we had at hand. Thank you.

References

1. Madhugiri, D. (2021) Using CNN for image classification on CIFAR-10 Dataset, Medium. Available at: https://devashree-madhugiri.medium.com/using-cnn-for-image-classification-on-cifar-10-dataset-7803d9f3b983 (Accessed: 22 June 2023).
2. Garg, A. (2022) How to classify the images of the CIFAR-10 dataset using CNN?, Analytics Vidhya. Available at: https://www.analyticsvidhya.com/blog/2022/09/how-to-classify-the-images-of-the-cifar-10-dataset-using-cnn/ (Accessed: 22 June 2023).
3. Le, J. (2018) The 4 convolutional neural network models that can classify your fashion images, Medium. Available at: https://towardsdatascience.com/the-4-convolutional-neural-network-models-that-can-classify-your-fashion-images-9fe7f3e5399d (Accessed: 22 June 2023).
4. Kubat, M. (2021) Introduction to machine learning. Cham: Springer.
5. Chollet, F. (2021) ‘Getting started with neural networks’, in Deep learning with Python. Shelter Island: Manning Publications, pp. 56–92.
6. Implementing a CNN in Tensorflow & Keras (2023) LearnOpenCV. Available at: https://learnopencv.com/implementing-cnn-tensorflow-keras/ (Accessed: 11 July 2023).
7. Chollet, F., 2021. *Deep learning with Python*. Simon and Schuster.
8. Kingma, D.P. and Ba, J., 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.