

Reflective Journal for Deep Learning - A walk-through of CNN's using the MNIST

Dataset

This lab was a huge step of mine down the rabbit hole. Transitioning from the idea of what a neural network is, to an actual high-performing image classifier was at once difficult, yet also incredibly rewarding. In this reflection, I discuss the most important things I learned, obstacles that were overcome, and how it affected my comprehension of the subject at hand.

I thought of neural networks as a long chain of dense layers before but the Conv2D and MaxPooling2D seemed quite different. Now I can think of the convolutional layers as the eyes of the network. As I mentioned in my lab, the Conv2D layer works as a specialized kind of feature detector and similar to a scanner sliding over the image. I can have 32 or 64 of those scanners all learning different features at the same time; some of them are finding edges, while others are detecting curves and still others more complex shapes.

This is particularly relevant to my lab by emphasizing what dense networks miss for image-based tasks. A dense network itself would only get an uninformative 784-dimension vector after flattening and all spatial information (which images have) would be lost. The CNN perceives proximity and spatial hierarchy thanks to kernels. The next layer MaxPooling2D continues this by decreasing the dimensions of the feature map, leaving behind only the valuable detected features. This not only makes the model more efficient, it also adds in some robustness, or translation invariance so the network is able to recognize a “5” even if it is written slightly off-center.

What really stunned me was just how well this entire architecture worked. My last model reached 99.15% on the test set, which is great for a very simple model. It was also interesting to learn some of the pre-processing that took place. The

requirement to re-form the grayscale images to (28, 28, 1) with channel and one-hot encode the labels via function `-to_categorical` are both hands-on realities that are sometimes obscured during theoretical discussions. One-hot-encoding, as I remarked, is essential in this case to ensure that the model does not learn a spurious and non-existent order of magnitude between the digits.

Although the lab time was good structure, I struggled on the "fill-in-the-blank" bits where you have to know what is going on. The first task was really choosing the architecture itself, especially for what concern in the parameters of the second Conv2D layer. I chose to use 64 filters, which is generally a good practice to follow by using an increasing number of filters as the spatial dimensions get lower after pooling. This is believed to enable the network to learn higher-level characteristics based on lower level details identified in the first layer.

A harder task was to choose the right optimizer and loss functions. I got past this by thinking of it as a different problem: multi-classification. The output for the soft-max activation in a final layer is a probability distribution. For that reason categorical crossentropy made most sense as a loss function, as it measures the distance between two probability distributions which are the prediction and the true label, and heavily penalizes confident but wrong predictions. Adam is one of the chosen best, robust and efficient default for a task like this so I used it as an optimizer.

Finally, choosing the `batch_size` and `epoch` to fit the data was a trade-off. I settled on 128 for `batch_size`; small enough that the neural network can converge more quickly with each epoch, but large enough that computation using them is stabilized. I began with 15 epochs, which was an okay number that didn't let the model massively overfit. I then reran it in 5 epochs but didn't get accurate. Monitoring the validation accuracy during training showed that this was a good place to start.

This lab has deeply altered my perception of deep learning. It's not just layers as an abstract concept, but the actual engineering around what tools should be used for a given job. I have to finally see the separation of 'feature extraction' part of the network (containing convolution and pooling blocks) and a classification layers. The Flatten layer is the critical bridge between these two stages, transforming the 2D feature maps into the 1D vector that the classifier can understand. I also started understanding why Dropout has value as a practical and necessary technique to avoid overfitting, which I observed was an actual danger as the training accuracy was going up.

I was absolutely impressed how the model turned out to be so accurate (99.15%), but as I've been doing with my lab notebook, this lead me on a journey of thinking how could push this further. Now I am looking forward to trying data augmentation. If I used slightly altered versions of the images during training, it is possible that I could build a more robust model. I also would like to try a LearningRateScheduler that is decreasing the learning rate step by step, possibly allowing for more fine-tuning of weights in the later epochs. Also, I'd like to explore the concept of ensembling which is training many good models and averaging their predictions.