Kaggle Competition ICL2018-LHC

Seyed Nasrollahi

Originally, I began the problem by trying the architectures and methods that we used in the seminars.
A layer with satisfactory results was,
Conv -> BatchNorm -> MaxPool -> ReLU

The notebook itself used,
MaxPool -> Conv -> MaxPool -> ReLU

Another seminar used
Conv -> ReLU -> Conv -> ReLU -> MaxPool

Hence the general idea is to use convolution to extract the features of the images, ReLU to add non-linearity, MaxPool to shrink the image size and dropout to set to zero a fraction of the neurons.
I used the original dense layer of the notebook throughout the work.
Linear -> ReLU -> Linear -> Sigmoid

For very quick initial trial and errors, I used a small training sample size of 20000. I tried the single above layers to see which one produced the best results. I realised that Conv -> BatchNorm -> MaxPool -> ReLU was the most sensible one. However, from online articles, I found out that theoretically, the order of MaxPool and the non-linearity function such as ReLU does not matter, whereas the order of MaxPool and Convolution do matter. My intuition was that it may be computationally faster for ReLU to be after MaxPool as there would be less non-linear operations being done on the features, but it didn't really make any difference when I tried it.

Also a common practice is to make a sharp increase in the number of convolution out channels at the beginning and then the changes are done in smaller steps. VGG doubles the out channels after every few stacks, and some others do the opposite. The latter is of course faster, but then MaxPooling cannot be over-done otherwise the final output before dense layer would be too small.

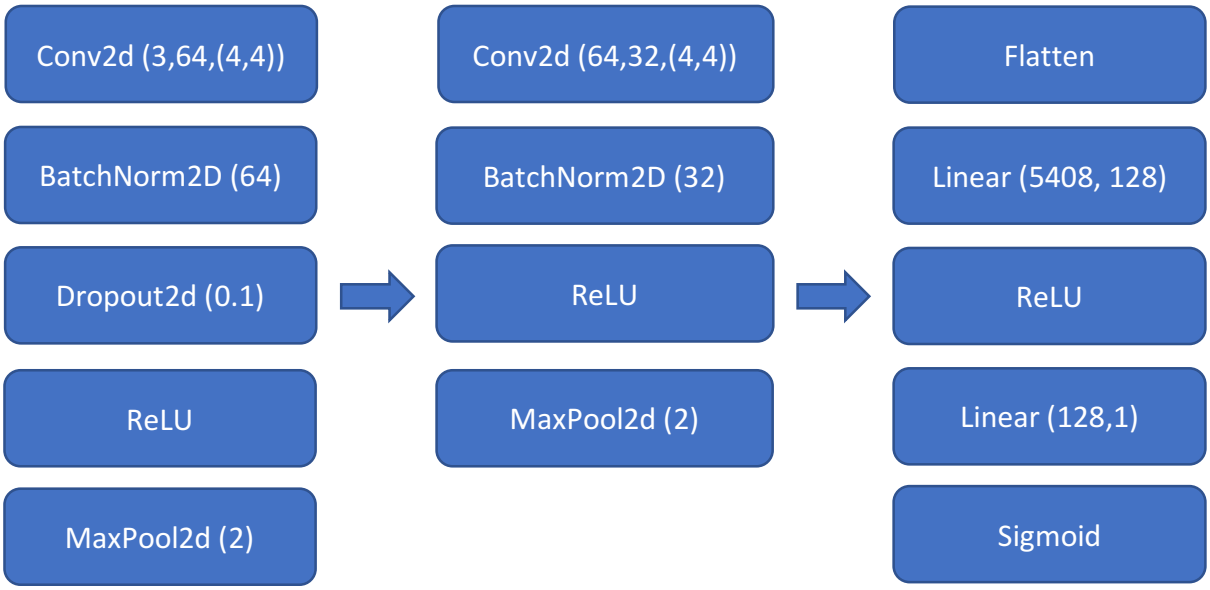
Although a small convolution filter size seems more desirable, trying all 5x5, 4x4 and 3x3, I found out that 3x3 is less accuarate than 4x4 and 5x5.
I perused the strategy of sharp increase to 64 or 128 channels and then lowering the number of channels gradually with some layers having same in and out number of channels (depending on number of layers). I also did MaxPooling on some layers (depending on number of layers) to keep the final dimension reasonable (~ 200 - 1000).

I also tried using 10% dropouts with inplace being False. It slightly improved the results. Overdoing the dropouts did not improve the results, but seemed to slightly negatively affect.

Moreover, Adam optimiser significantly improved the results compared to the original SGD. A larger sample size also had an enormous positive effect on accuracy, but also increased the training time drastically.

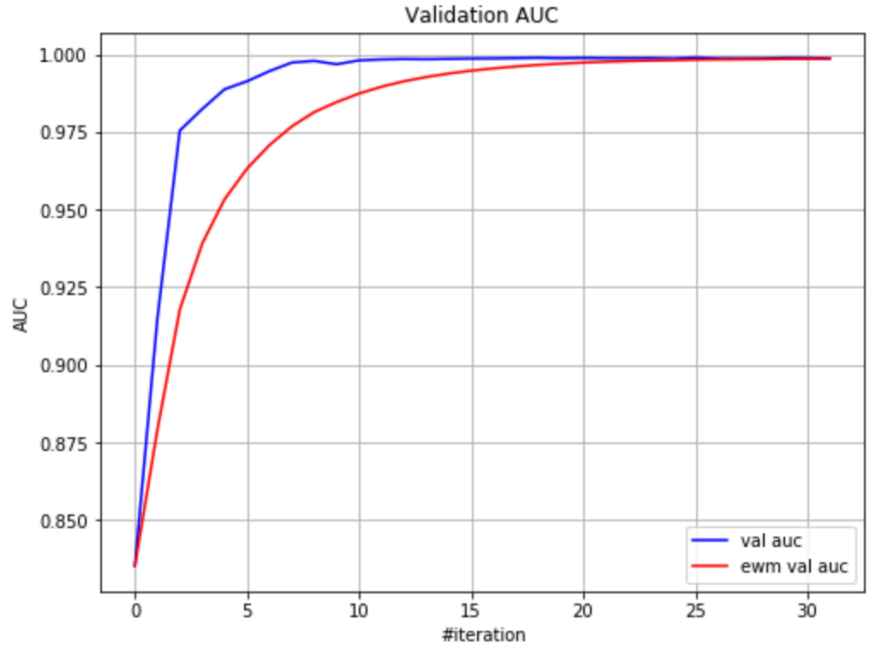
First Submission: Simple, fast, accurate

My first submission was a two layer convolution and one final dense layer.



However this was done on a small sample size of 20000. The accuracy was 99.453% on Kaggle. Later when I tried this on larger sample and batch sizes, I managed to get a maximum of 99.90% accuracy on validation data with a convergence to 99.87%.

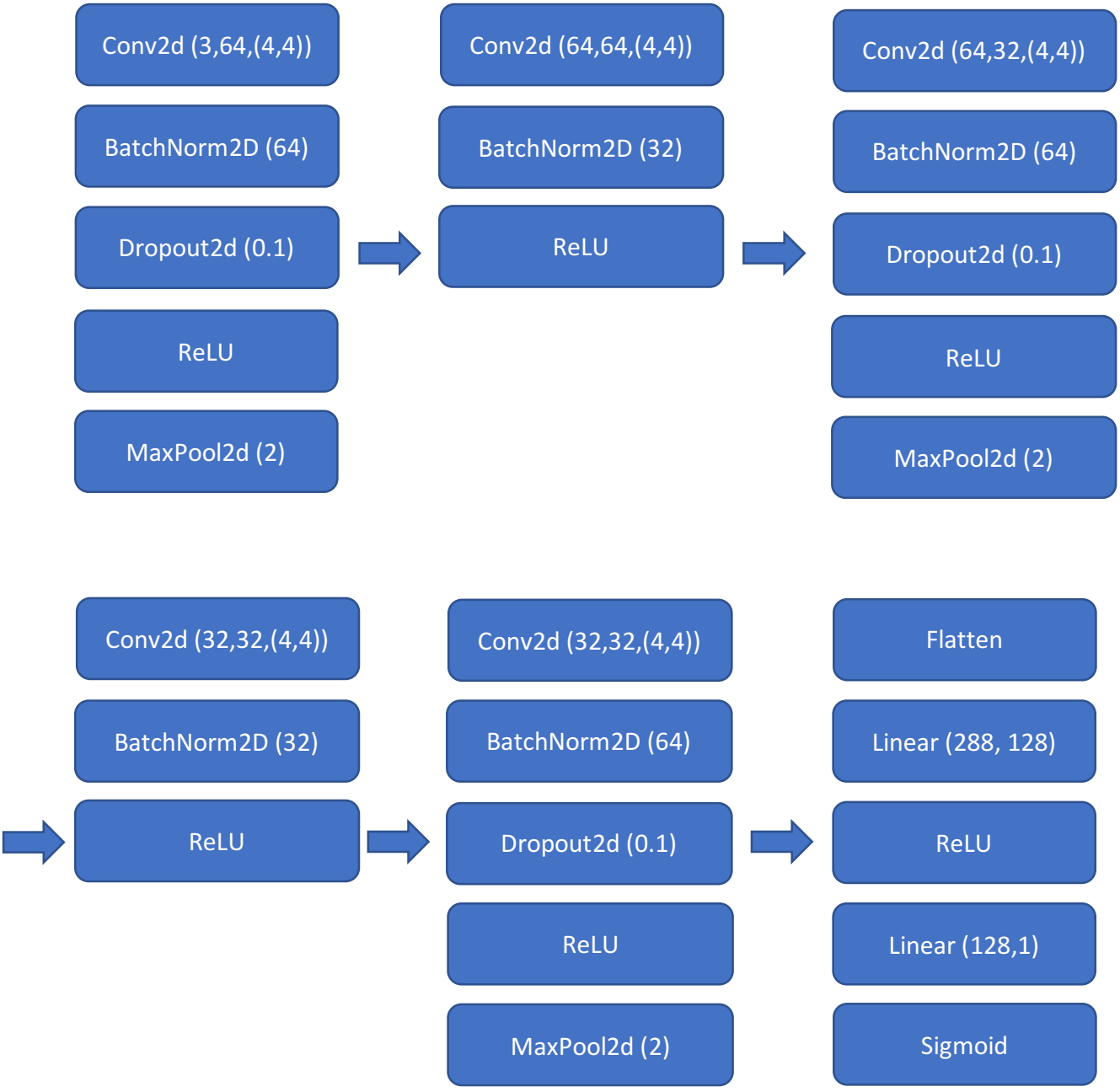
This did not guarantee beating the Hard Baseline so I tried adding one more layer, changing the filter size to 3x3 and 5x5 or using other non-linear functions such as Tanh instead of ReLU (changes were done under controlled conditions). None of these improved the results beyond a maximum accuracy of 99.90% and the convergence was more or less around 99.87 still.

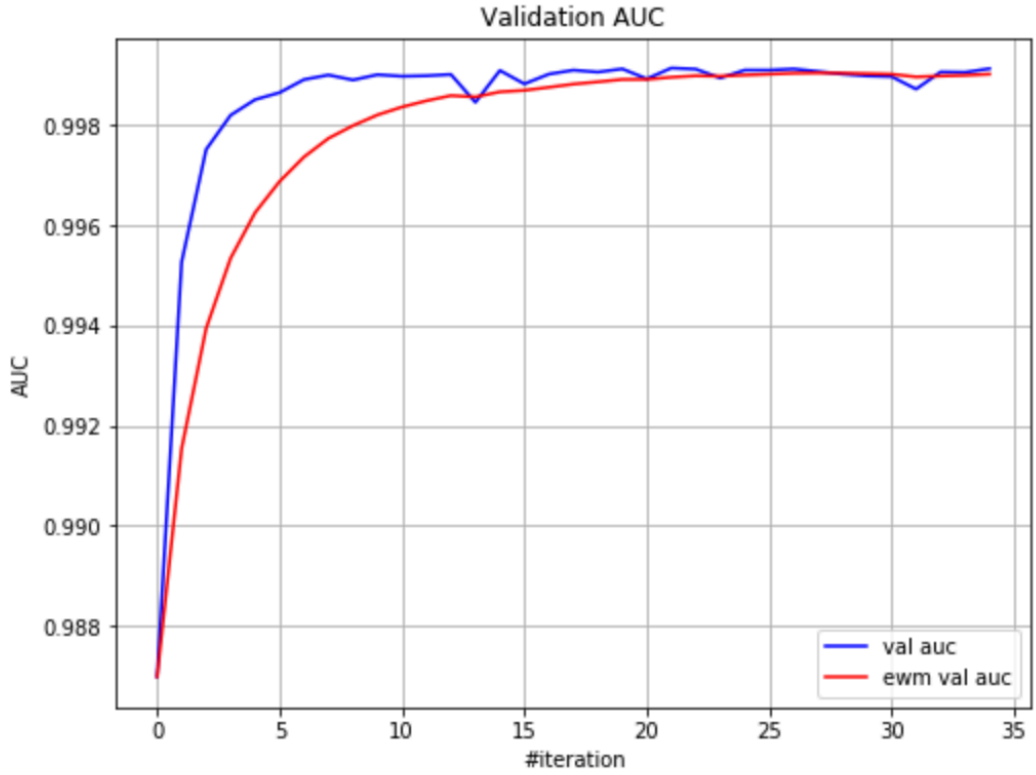



```
Epoch 1 of 1 took 41.693s
  training loss (in-iteration):         0.016853
  train accuracy:                 103.57 %
  validation accuracy:            98.46 %
  validation roc_auc:             99.87 %
```

Second submissions: Complex, fairly fast, more accurate

In order to push the results beyond the Hard Baseline, I added three more convolution layers hoping that this would do the trick.
Hence my second model, which had 3 extra layers, pushed the accuracy to around 99.7% on the validation data for 20000 samples and the final training was done on a sample size of 100000 which pushed the accuracy to a maximum of 99.92% on validation data. This is a lot of added complexity for a very little change. As was said, for the 5 layers of convolution, I decreased the out channels down to 32 and added Dropouts and MaxPools to every other layer.

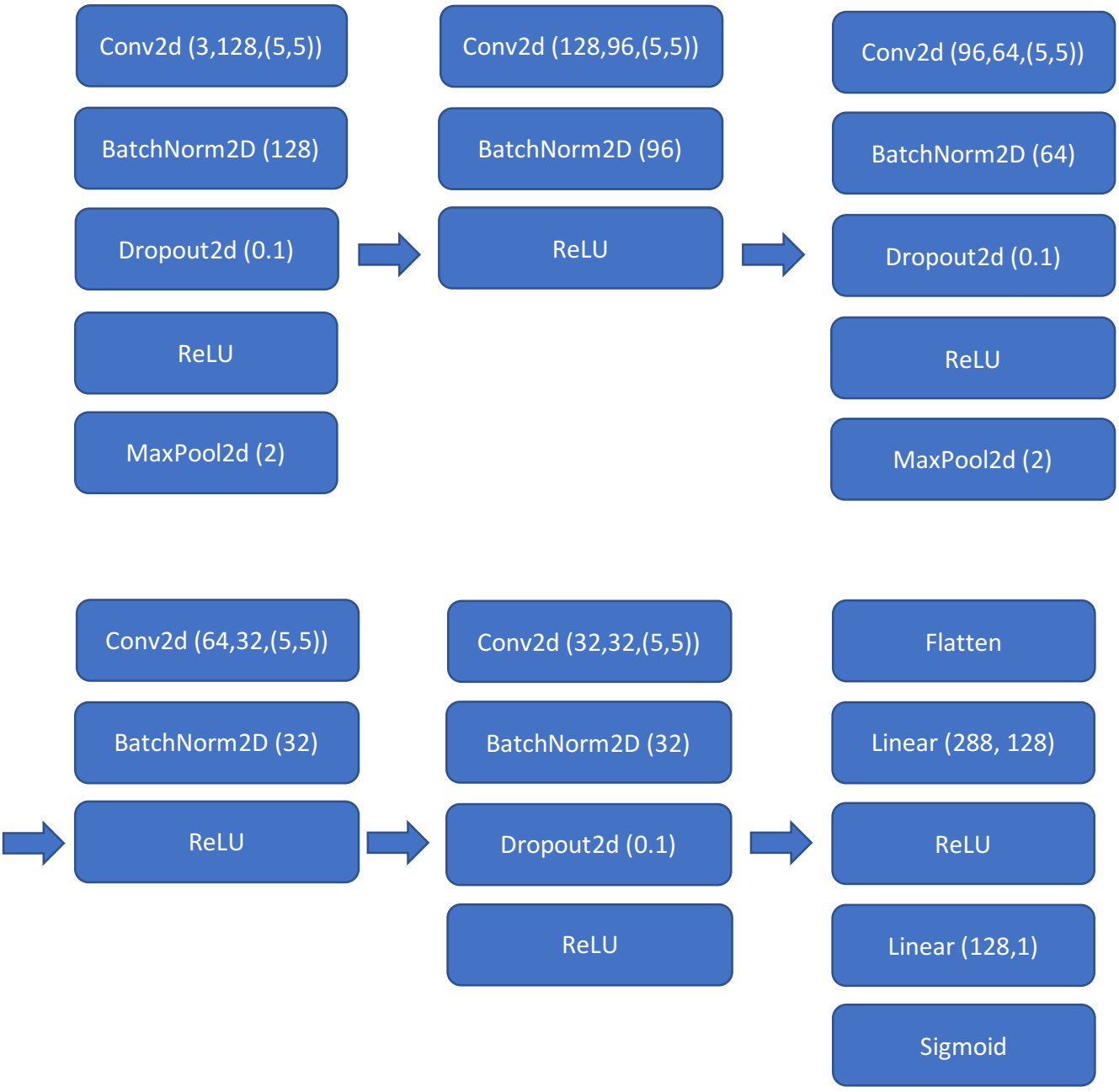




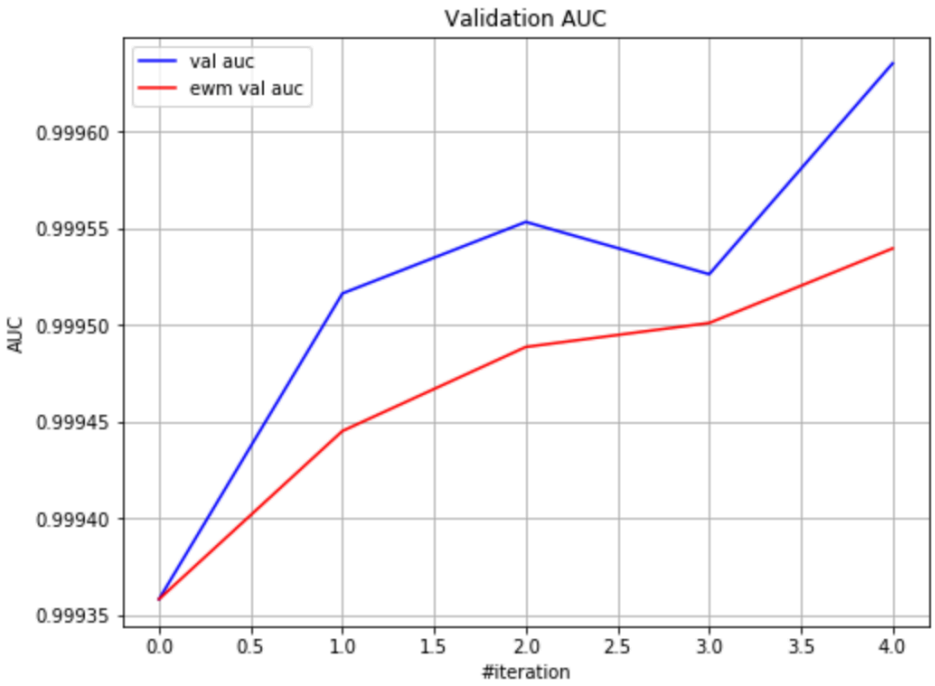

```
Epoch 3 of 3 took 58.005s
  training loss (in-iteration):          0.014527
  train accuracy:                103.62 %
  validation accuracy:           98.76 %
  validation roc_auc:            99.91 %
```

Final submission: Complex, slow, very accurate

To even push my score further, I added more convolution filters, shooting up the first out channel to 128, and gradually decreasing to 32. I had to remove the final MaxPool to keep the dimensions sensible as I tried the 5x5 filter size this time.
The filter size of 3x3 decreased the accuracy significantly, whereas 4x4 and 5x5 were indistinguishable. Initial output channel to 256 was not useful either and slowed down the calculation.



This time I managed to run the calculation on 220000 samples. Memory has been a major problem. I have not been able to read more samples than this. I also took 95% of the data to be for training, and 5% for validation. Even if so, there is a high chance of memory errors occurring during or after calculations. Solutions would be to choose smaller batch_size for the training and deleting the unwanted data such as X_train and y_train after the training (otherwise reading X_test would produce memory error).
This time I managed to achieve an accuracy of 99.96%.




```
Epoch 1 of 1 took 1681.332s
  training loss (in-iteration):          0.030592
  train accuracy:                98.90 %
  validation accuracy:           99.18 %
  validation roc_auc:            99.96 %
```

This submission got me scores of 0.99957 and 0.99954 on the public and private leaderboards respectively.