Kobee Raveendran

PA4 – Hyperparameter Tuning

**Hyperparameter Tuning Report**

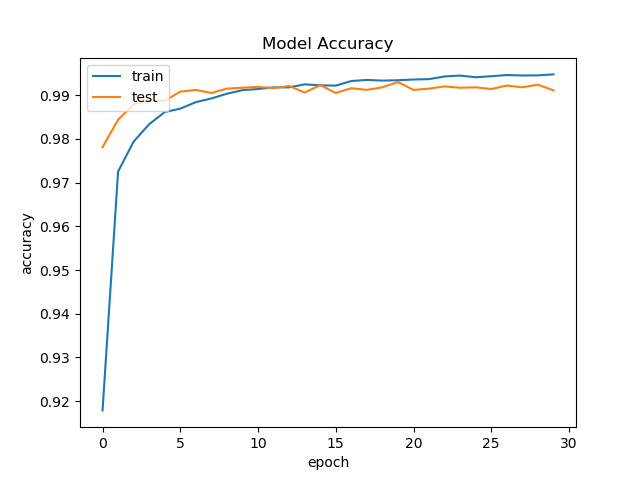
2) Accuracy after training on 12 epochs (default): 99.08%

3) model: conv 🡪 conv 🡪 max-pooling 🡪 dropout (and flatten to prepare output for FC layers) 🡪 FC 🡪dropout 🡪 FC

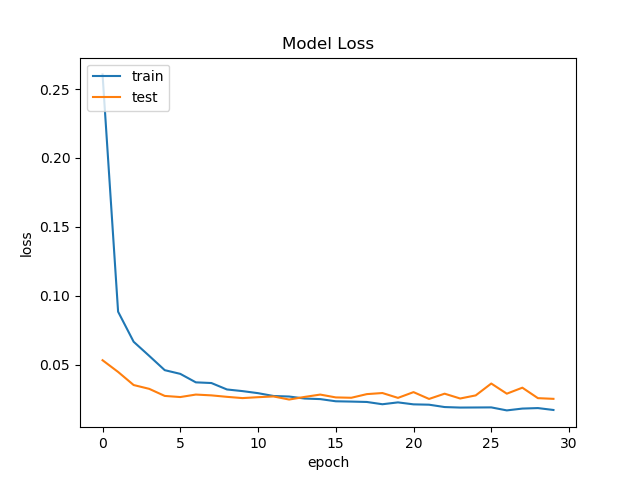
4) Accuracy after training on 30 epochs: 99.27%

Loss after training on 30 epochs: 0.02566379

Part 4 accuracy plot:



5) loss plot:



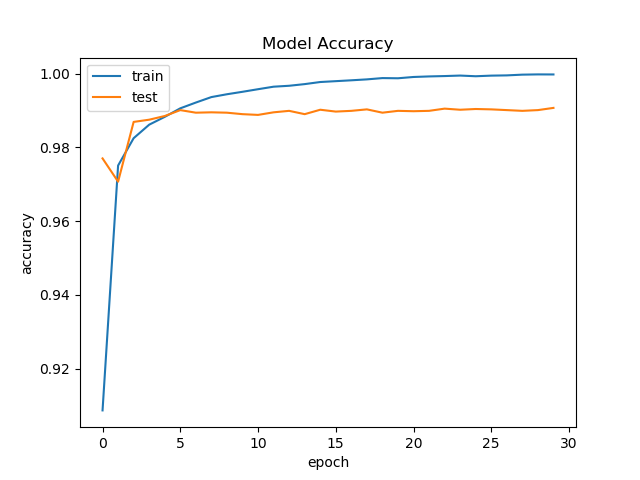
6) **LeNet-5:**

*Using maximum pooling*:

test accuracy: 99.07%

Test loss: 0.0487900099

Train + test accuracy plot:

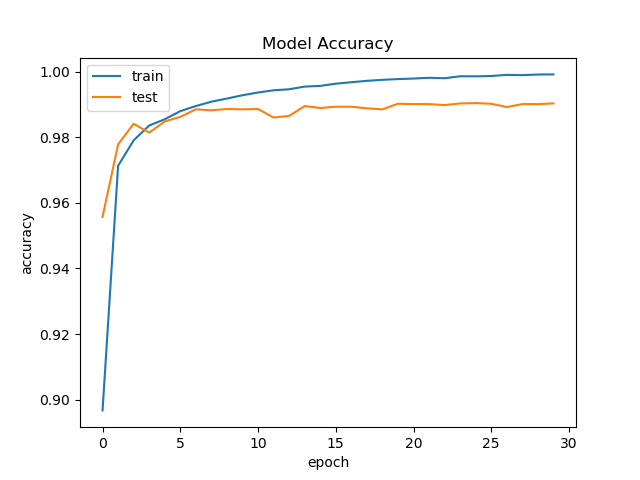


*Using average pooling*:

Test accuracy: 99.03%

Test loss: 0.0486580606

Train + test accuracy plot:



7) LeNet-5 on Fashion-MNIST:

Training time: 93.94 seconds

Test loss: 0.35456

Test accuracy: 89.77%

Base model on Fashion-MNIST:

Training time: 191.86 seconds

Test loss: 0.232143

Test accuracy: 93.14%

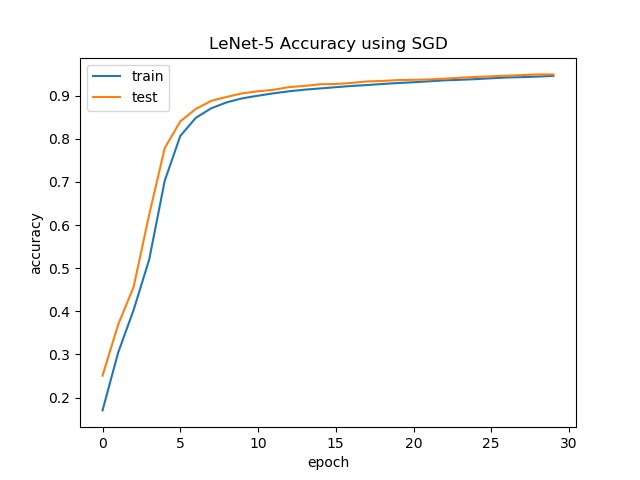
(plots for both below)

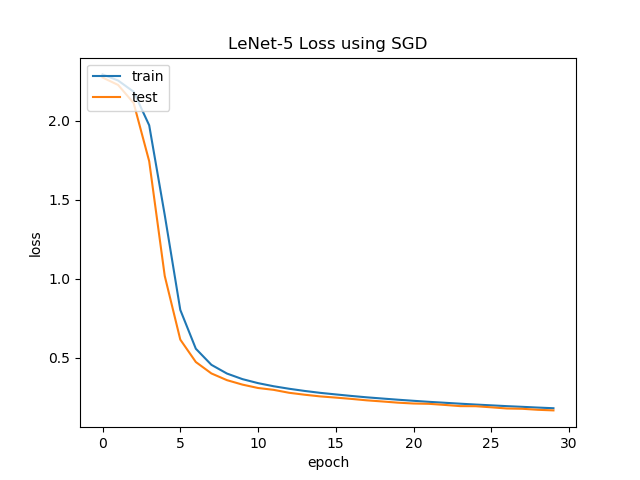
LeNet-5:

The results somewhat surprised me, since LeNet (although a deeper model) performed worse than the base model (with respect to accuracy) on the Fashion dataset while performing about equally with the base model on the MNIST digits dataset. One other thing was the training time; I’d expect a shallower model like the base model to train much quicker than LeNet, though that also wasn’t the case here, for a reason that I’m unsure of. I did include both models in the same file, though they were both trained on the same initial data so I doubt that was the problem here.

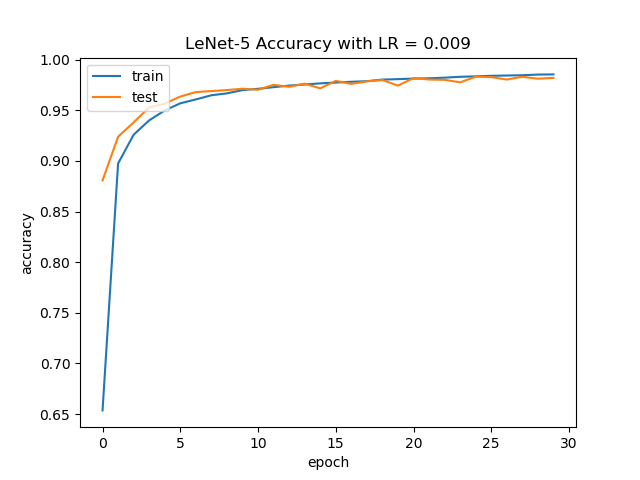
8) LeNet-5 Accuracy using SGD: 94.88%

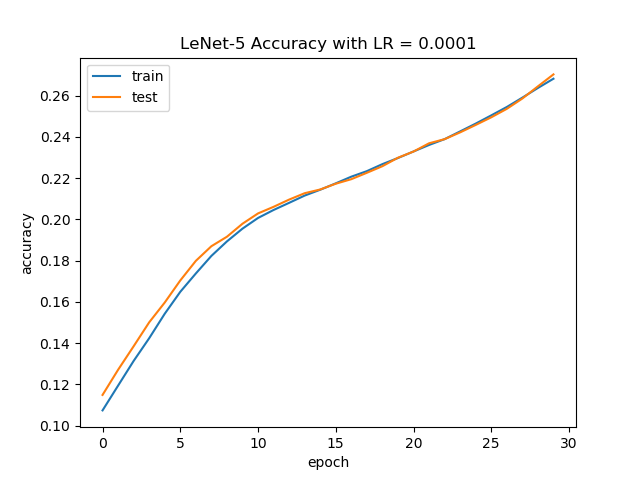
LeNet-5 Loss using SGD: 0.1675552



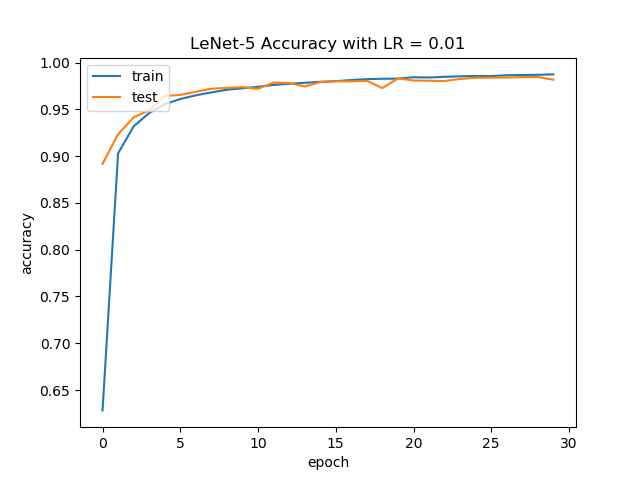


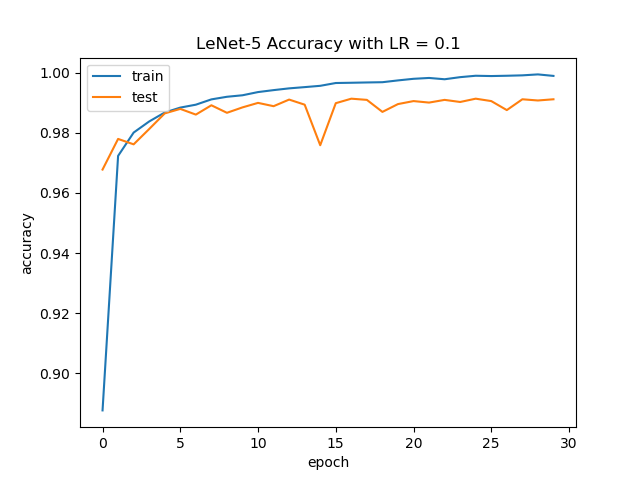
9) **NOTE:** all models below were trained using the stochastic gradient descent optimizer, and the learning rates chosen are reflective of that (compared to the base model’s use of Adadelta, which has a default learning rate of 1.0 in Keras)

learning rate = 0.009:

Learning rate = 0.0001:

Note on this learning rate in particular: this learning rate deviates from the rest in a peculiar way, since both the initial and terminal accuracies are both relatively low when compared to the other learning rates)

Learning rate = 0.01:

Learning rate = 0.1:

Learning rate = 0.09: