Kobee Raveendran
PA4 – Hyperparameter Tuning

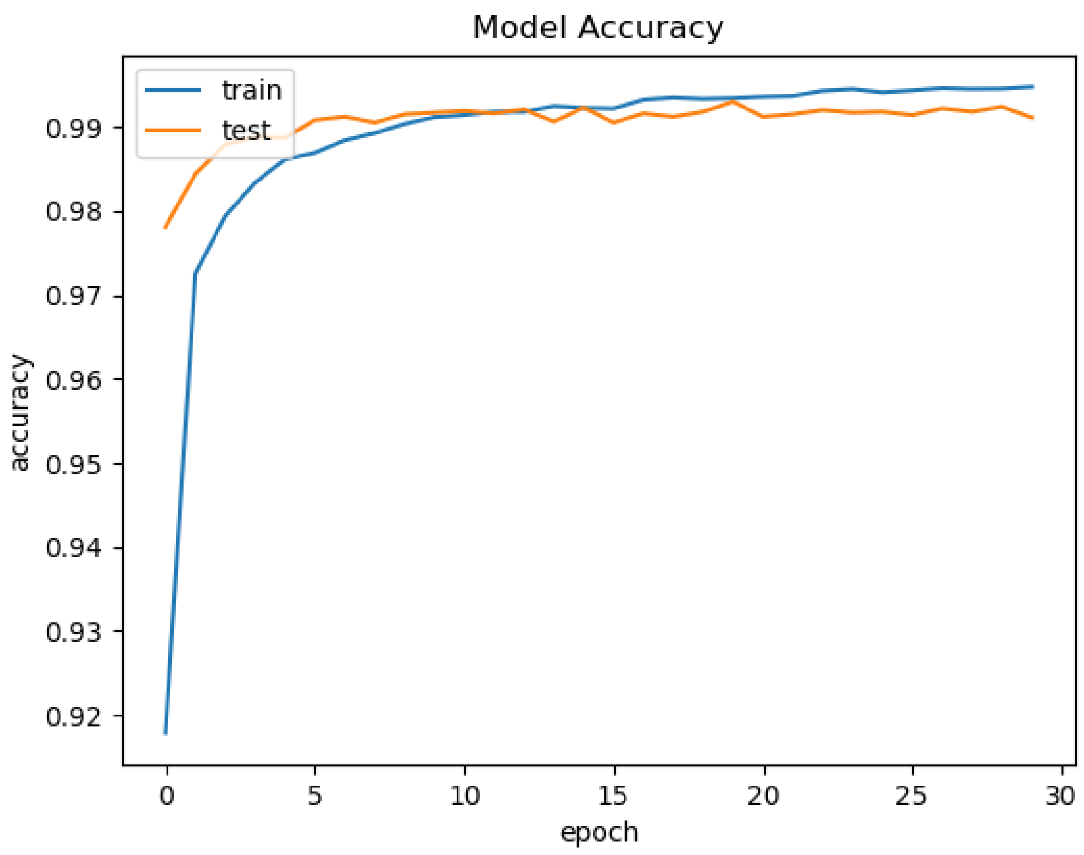
Hyperparameter Tuning Report

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

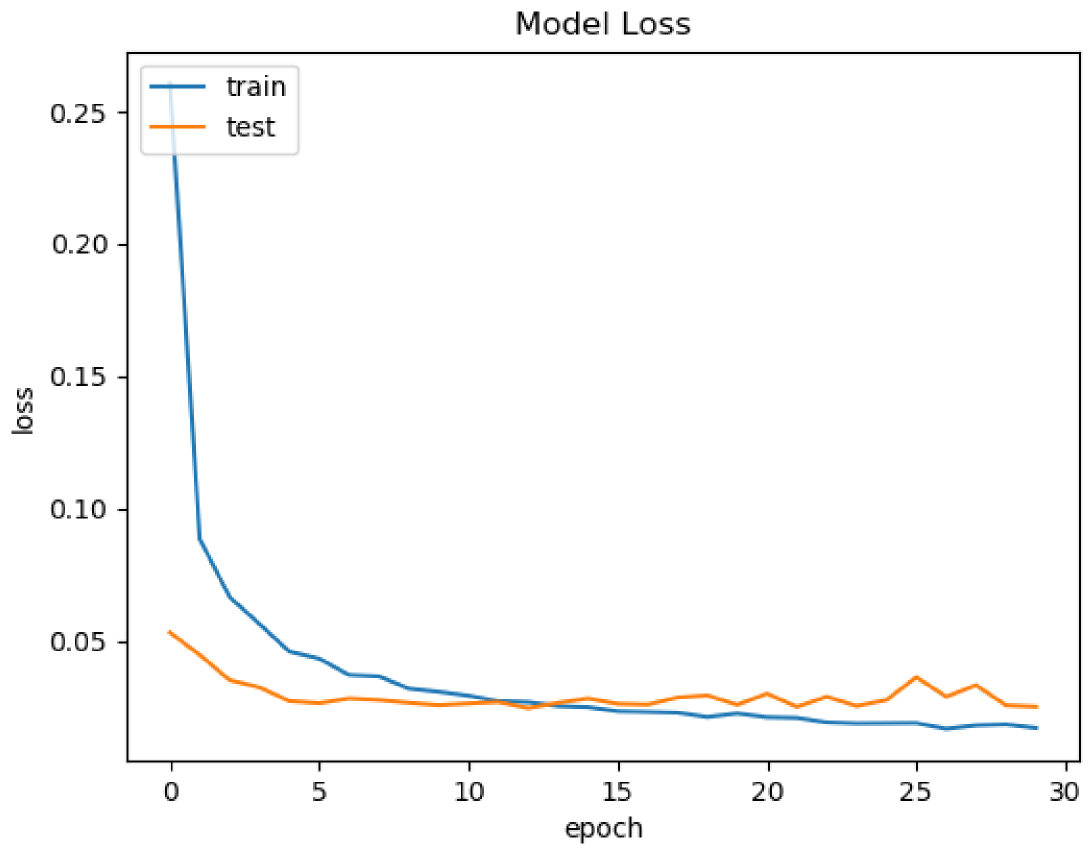
3) model: conv → conv → max-pooling → dropout (and flatten to prepare output for FC layers) → FC →dropout → FC

4) Accuracy after training on 30 epochs: 99.27%
Loss after training on 30 epochs: 0.02566379

Part 4 accuracy plot:

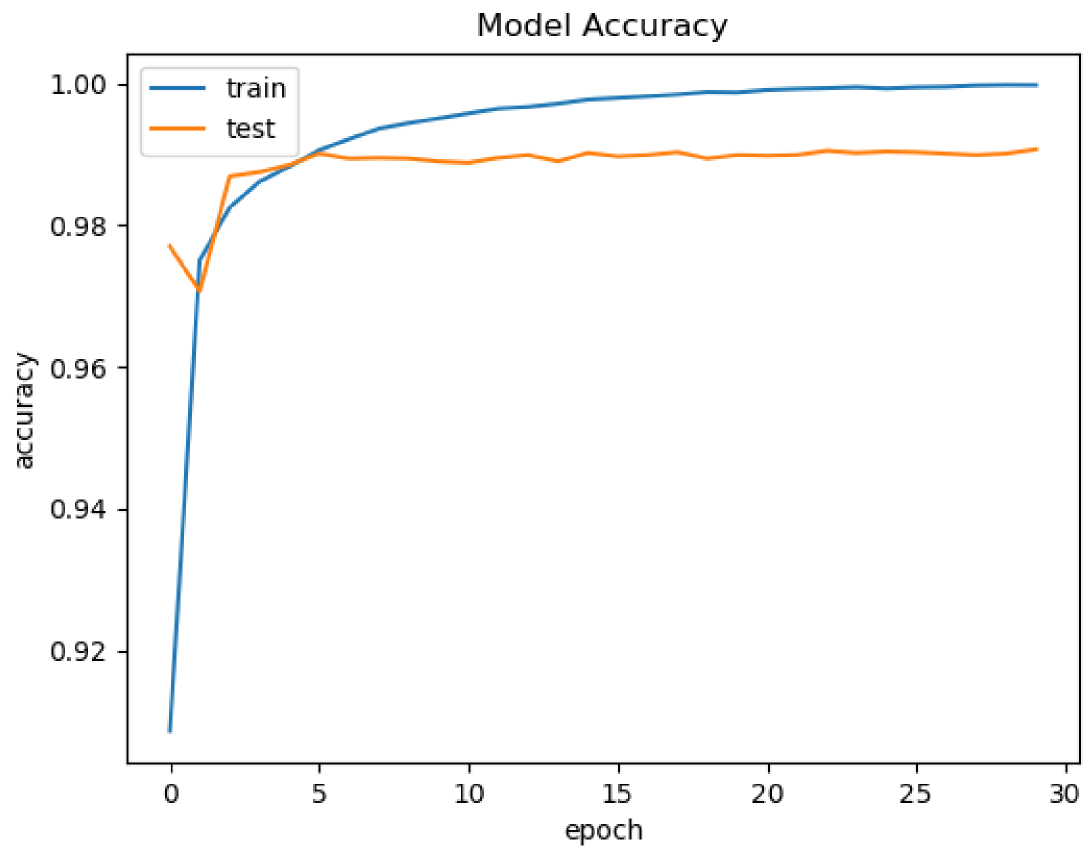


5) loss plot:



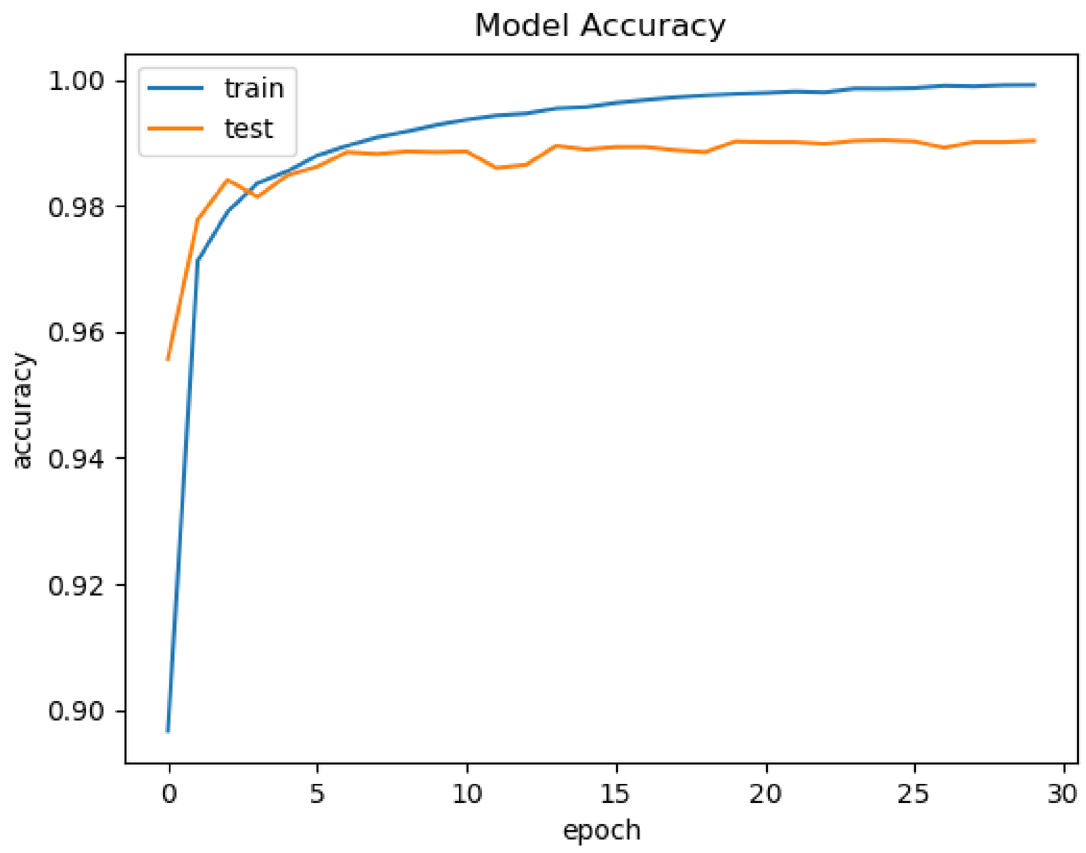
6) **LeNet-5:**
Using maximum pooling:
test accuracy: 99.07%
Test loss: 0.0487900099

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%

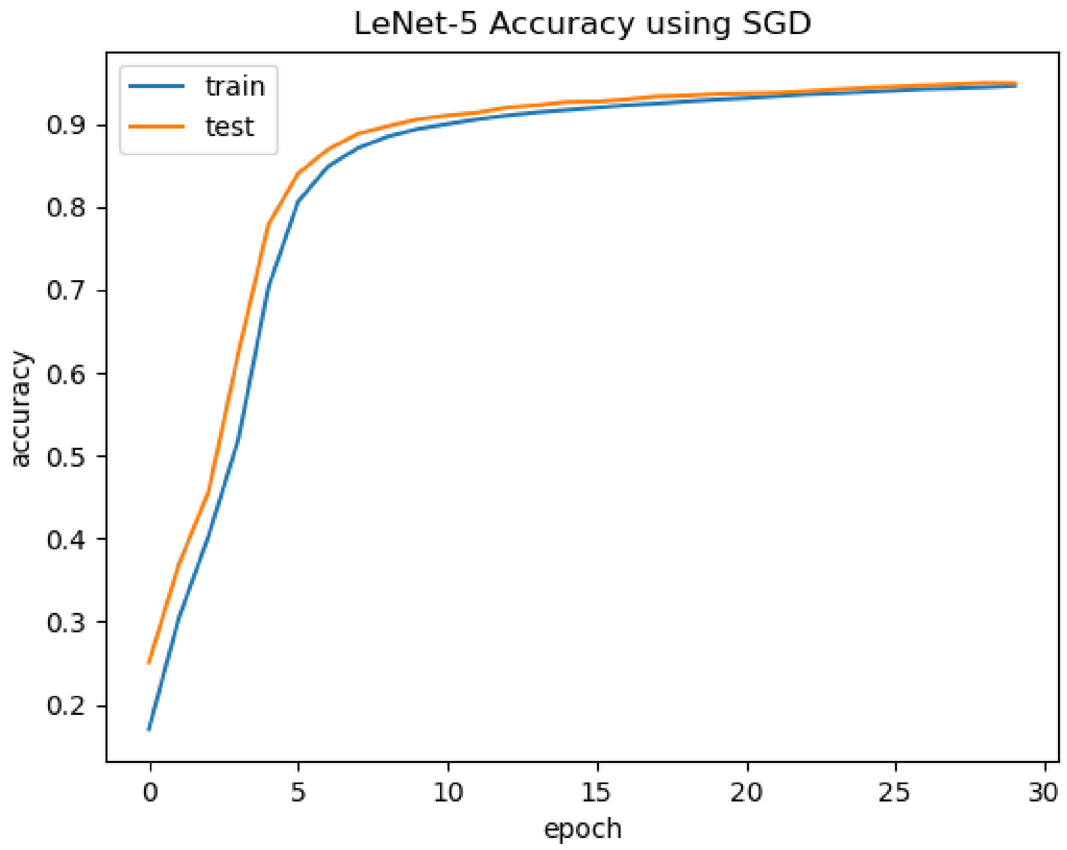
Base model on Fashion-MNIST:
Training time: 191.86 seconds
Test loss: 0.232143
Test accuracy: 93.14%

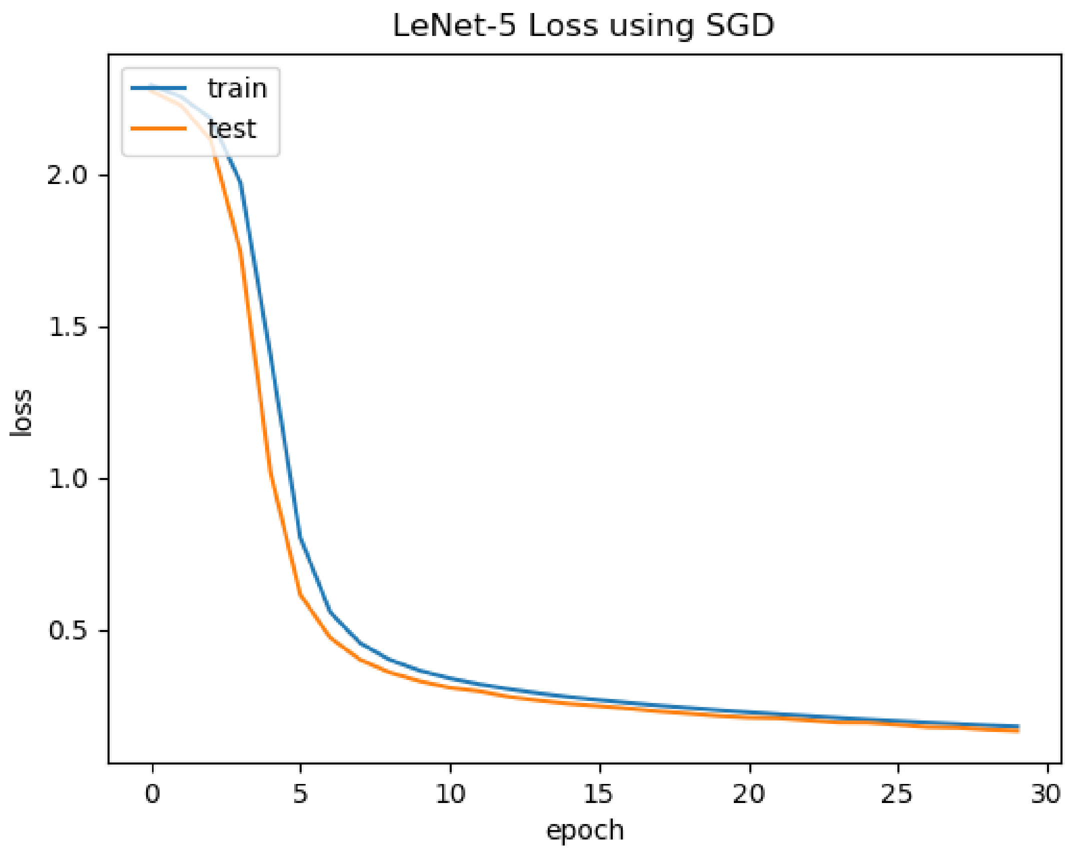
(plots for both below)

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.

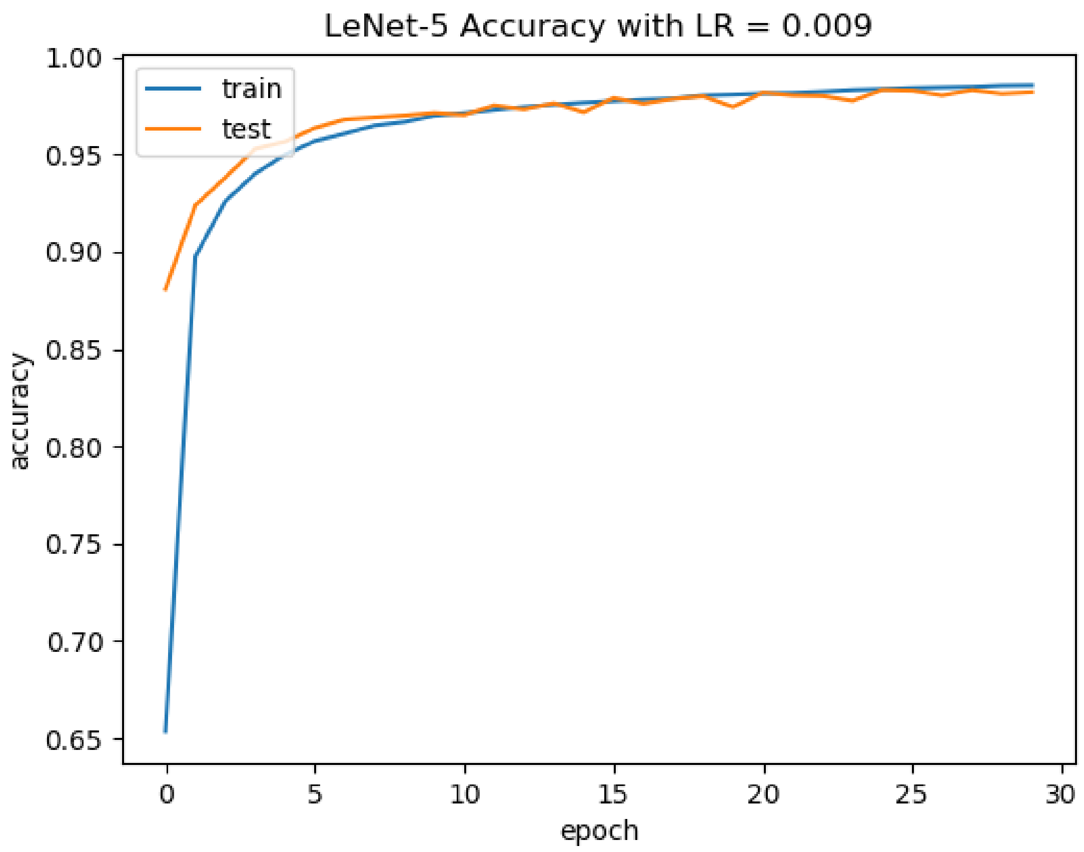
8) LeNet-5 Accuracy using SGD: 94.88%
LeNet-5 Loss using SGD: 0.1675552



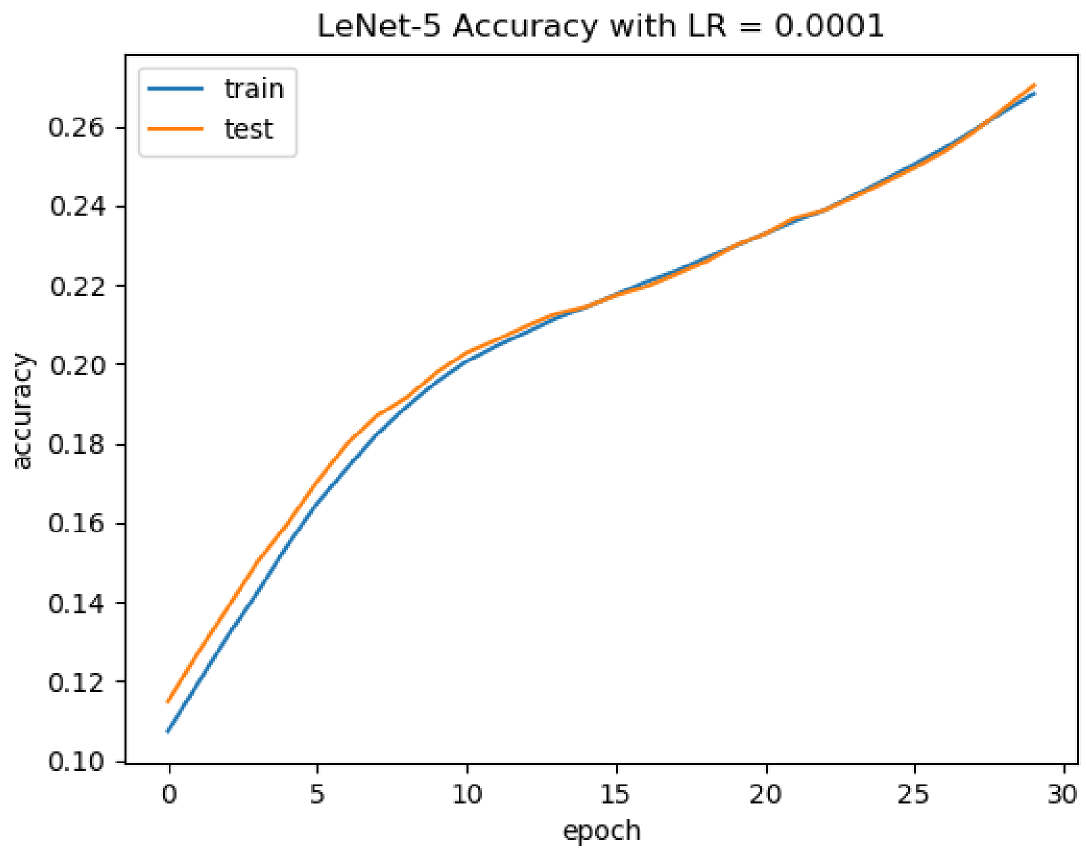


9) **NOTE:** all models below were trained using the stochastic gradient descent optimizer, and the learning rates chosen are reflective of that (compared to the base model's use of Adadelta, which has a default learning rate of 1.0 in Keras)

learning rate = 0.009:

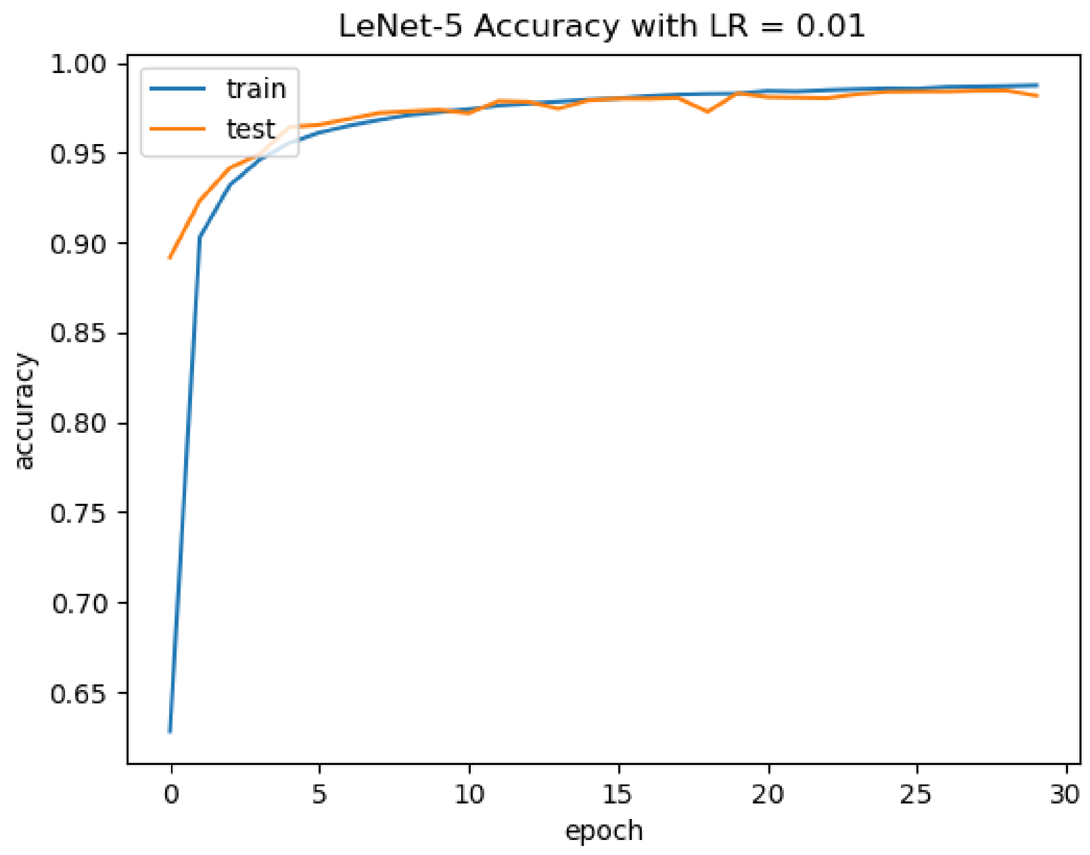


Learning rate = 0.0001:

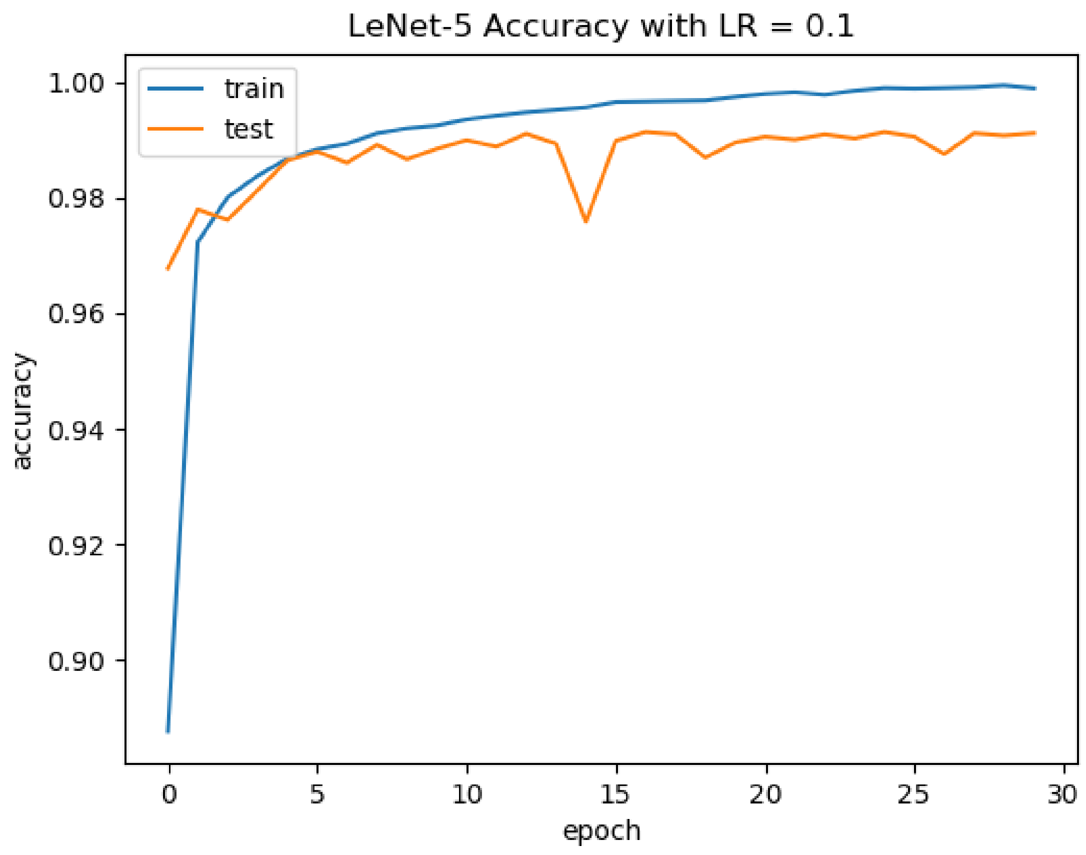


Note on this learning rate in particular: this learning rate deviates from the rest in a peculiar way, since both the initial and terminal accuracies are both relatively low when compared to the other learning rates. I believe this may be because the learning rate is too many orders of magnitude smaller than the more "optimal" learning rates; as such, it learns too slowly and

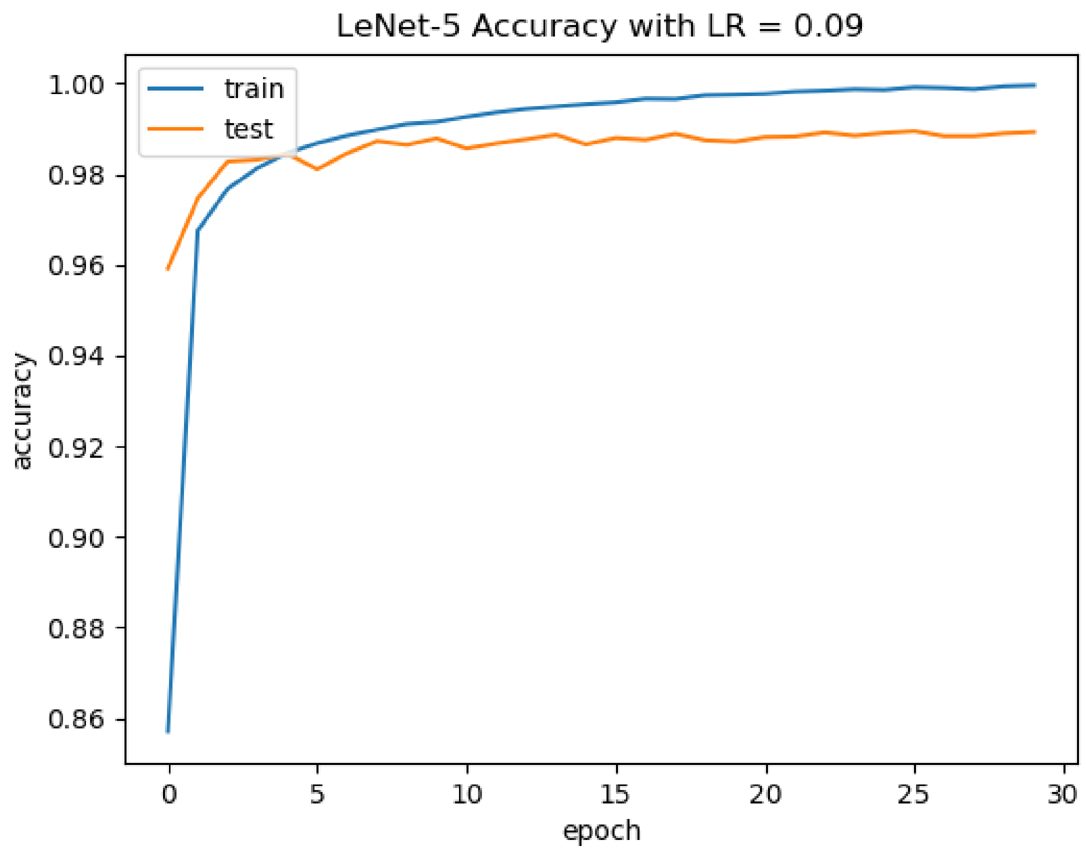
Learning rate = 0.01:



Learning rate = 0.1:



Learning rate = 0.09:



10) In my case, AdaDelta seemed to slightly outperform SGD in terms of accuracy (sometimes by even 5%), though I did notice that SGD definitely seemed to converge a lot smoother and without the wilder bumps that were characteristic of AdaDelta.

When I changed the learning rate, I noticed that there wasn't much of a difference between them for the most part, until the learning rates changed by orders of magnitude. For instance, learning rates of 0.01 and 0.09 performed relatively similarly, but learning rates of 0.1 and 0.0001 were extremely different. Surprisingly, all of them were useful except the learning rate of 0.0001. I somewhat expected the accuracy to be low for this learning rate, since I thought the model would learn too slowly. However, I also expected the learning rate of 0.1 to be too high, but surprisingly the model performed similarly on this learning rate to the others (but had a bit of a discrepancy between the train and test accuracies compared to say, lr = 0.01, which matches closely). In the case of these learning rates, there is a tradeoff: the lower the learning rate, the less likely the model is to converge, and the same can be said for an excessively high learning rate (such as 1.0), for SGD.