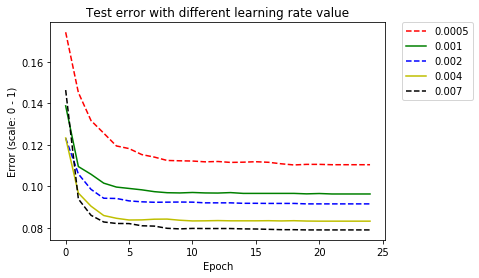
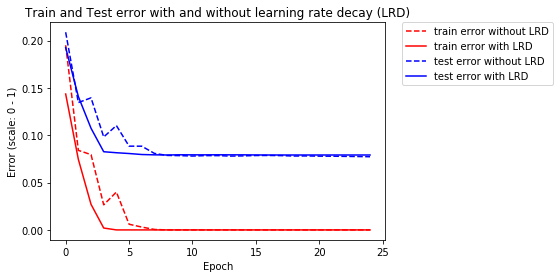
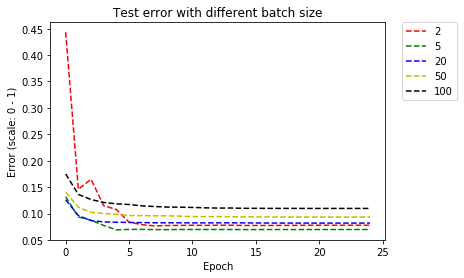
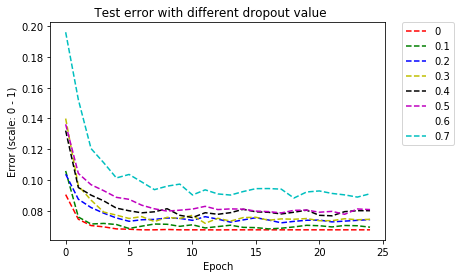
**Fully connected neural network**

Even if a fully connected architecture is not efficient for an image classification task, we wanted to try it. We first built one just for the digit recognition task. We found the best parameters we could by looping through all the parameters such as hidden sizes, number of layers, batch sizes, learning rate values and by trying regularization technique (dropout) and learning rate decay.

Here some graphs showing some parameters’ influences:

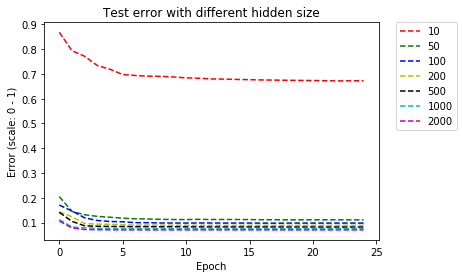
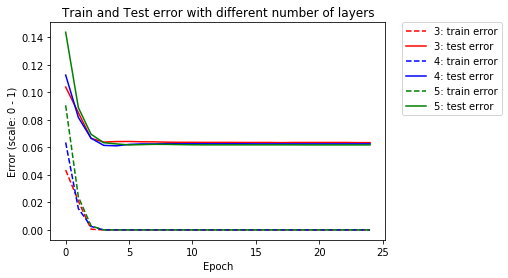


Above we evaluated the influence of learning rate decay technique and the influence of learning rate value. We can see that without a learning rate decay we could go to fast, the curb won’t be optimal cause of some bounds. A higher learning rate (not too high) with a learning rate decay enabled allow to converge quickly.

Thanks to the graph about batch size, we can compare the convergence’s speed. 5 seems to be the best choice for this problem with the parameters chosen.

Dropout is a regularization technique to help avoiding overfitting, in our case we don’t experiment overfitting so it doesn’t help improving our accuracy.

We wanted to test if increasing the number of layers is more efficient than increasing hidden size. Thanks to the first graph, we found that having 50 or 2000 as hidden size doesn’t change much the result. Unfortunately, increasing the number of layers didn’t change the result either.

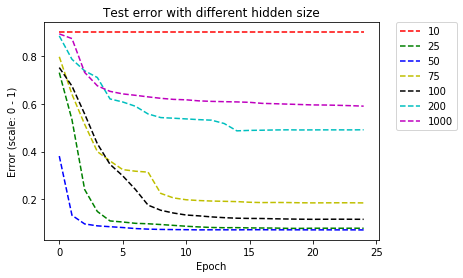
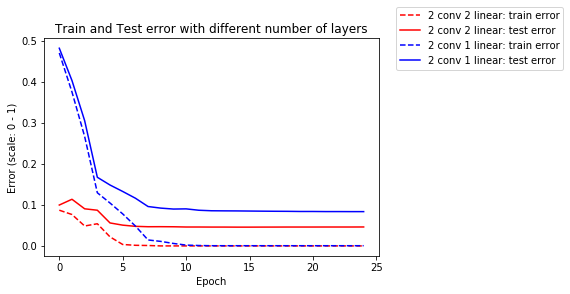
The test errors have been evaluated against 10+ different samples of pairs of images, each samples of size 1000. Plus, we stored the training time we got with the different parameters. We achieved in 4s, 25 epochs, 3 layers and batch size of 20: ~11% of error (we converge before epoch 10 so no need to go until the 25th which would reduce the time to 2s). But, without the limitation of a few seconds for the training, we achieved 6.2% in a few minutes with hidden size: 2000, batch size: 5, 4 layers and with learning rate decay turned ON.

We then applied this neural network to pairs of 1000 images and compared the results with a logical operator. We got 3.9% of error to the digit comparison problem in a few minutes of training, around 10% with 4s of training.

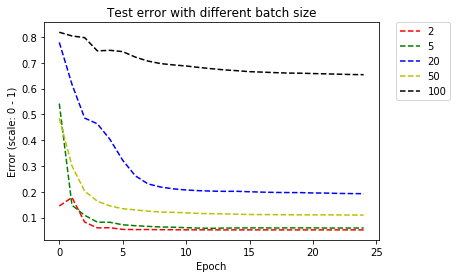
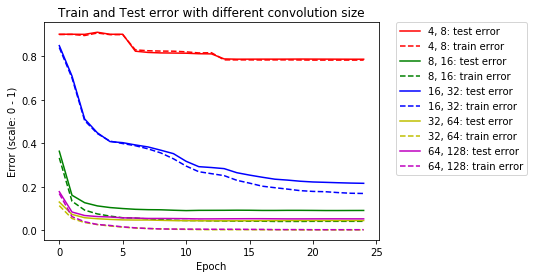
**Convolutional neural network.**

A convolutional neural network should be more appropriate to an image classification task. We also wanted to experiment the influence of the convolutional layers’ parameters. This architecture is much slower but more efficient.

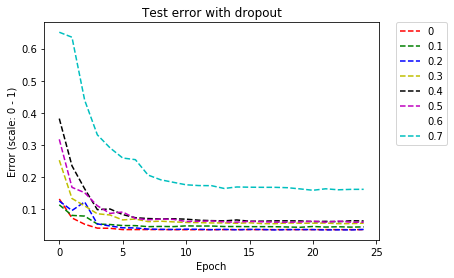
Here some graphs showing some parameters’ influences:

Contrary to the fully connected neural network, increasing the number of hidden size in our CNN changes a lot the accuracy.We also wanted to verified the impact of using only one or two fully connected layers. Two fully connected layers is more appropriate.

The batch size has also a big impact on the accuracy and the speed of convergence. 5 seems to be the optimal one for our problem with the parameters we chose. The size for the convolutions has also an noticeable impact.



As we don’t experiment overfitting in with our CNN, a regularization technique such as dropout doesn’t help reducing the test error.

All the test errors we got were smaller than the ones we previously got from the fully connected neural network. We got 4.5% error in digit recognition and then 2.7% of test error in the digit comparison. We almost divided by two our test error thanks to the convolutions. However, the training time is several minutes.

**Weight sharing and auxiliary loss.**

For this architecture we are using weight sharing, the same neural network for the recognition of the two images in each pair. Then we implemented layers and loss for the digit comparison. The loss we want to minimize is the addition of the two losses coming from the digit recognition of the two images in the pair and a loss based on the digit comparison. The main difference between this architecture and the previous ones is how we do the digit comparison. It was simple logic comparison, now, the neural network learns how to compare the digits in the neural network itself. We used the best parameters we previously found for the digit recognition part, and we used fully connected layers to learn how to compare the digits. We finally got 1.9% of test error. The training is unfortunately too slow. The best architecture to respect the performances wanted of 2s of training for 15% of error is a fully connected neural network with hidden size 50 and 3 layers.