**MNIST – State of the art**

1. Start point: After a lot of tests I got this model.

Model: "sequential"  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
Layer (type) Output Shape Param #   
=================================================================  
reshape (Reshape) (None, 28, 28, 1) 0   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
conv2d (Conv2D) (None, 24, 24, 64) 1664   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
max\_pooling2d (MaxPooling2D) (None, 12, 12, 64) 0   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
conv2d\_1 (Conv2D) (None, 8, 8, 32) 51232   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
max\_pooling2d\_1 (MaxPooling2 (None, 4, 4, 32) 0   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
flatten (Flatten) (None, 512) 0   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
dense (Dense) (None, 512) 262656   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
dropout (Dropout) (None, 512) 0   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
dense\_1 (Dense) (None, 256) 131328   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
dropout\_1 (Dropout) (None, 256) 0   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
dense\_2 (Dense) (None, 64) 16448   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
dense\_3 (Dense) (None, 10) 650   
=================================================================  
Total params: 463,978  
Trainable params: 463,978  
Non-trainable params: 0  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Before this model, I tried different number of many kinds of convolutional layers, and dense layers.

I checked also the activation function (picked ‘relu’ which was the best from sigmoid, leaky relu and elu), regularizations (picked L2), and optimizer (picked [Adamax](https://ruder.io/optimizing-gradient-descent/index.html#adamax)).

I have added extra 120k samples, which are half negative images with some normalized noise, and half are rotated original images (max ±20°) with some normalized noise.

Until now, I have managed to perform about those results:

Train loss: 0.072591  
Train accuracy: 99.61%  
Test loss: 0.081583  
Test accuracy: 99.36%

**Dropout modification:**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Rate | 0.1 | 0.2 | 0.3 | 0.4 | 0.5 | 0.6 | 0.7 | 0.8 | 0.9 |
| Train Loss | 0.034424 | 0.037574 | 0.043272 | 0.047848 | 0.054030 | 0.058203 | 0.068224 | 0.076419 | 0.093858 |
| Train Accuracy | 99.99% | 99.98% | 99.96% | 99.94% | 99.91% | 99.86% | 99.74% | 99.65% | 99.04% |
| Test Loss | 0.053189 | 0.053080 | 0.059763 | 0.62475 | 0.068946 | 0.069239 | 0.079066 | 0.086118 | 0.10193 |
| Test Accuracy | 99.43% | 99.49% | 99.41% | 99.47% | 99.38% | 99.48% | 99.41% | 99.38% | 98.73% |

Each of these tests done with **75** epochs, so from now on I will use dropout rate 0.2 (20%).

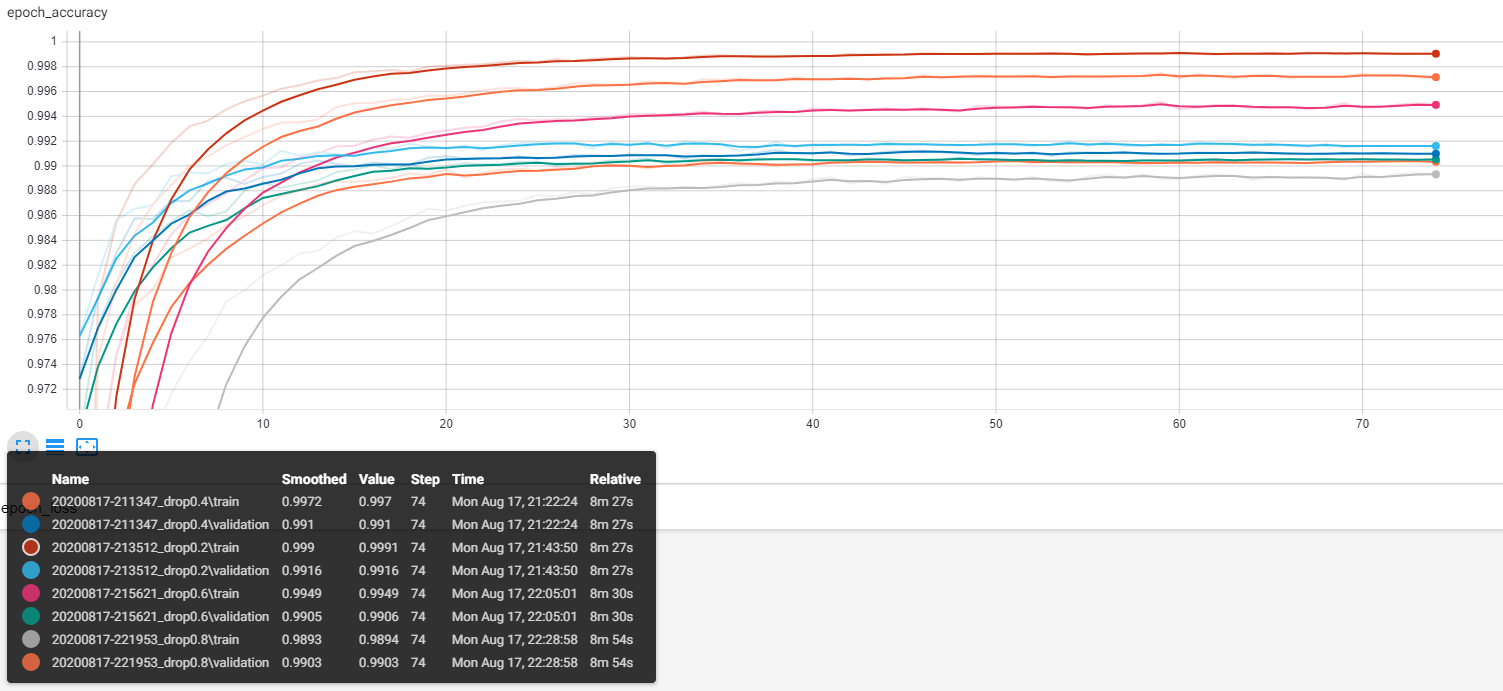


Figure 1 - dropout tests

**Adding Batch Normalization Layer:**

After I stayed with the dropout layers of rate 0.2, I have added [Batch Normalization](https://machinelearningmastery.com/batch-normalization-for-training-of-deep-neural-networks/) layer between all layers.

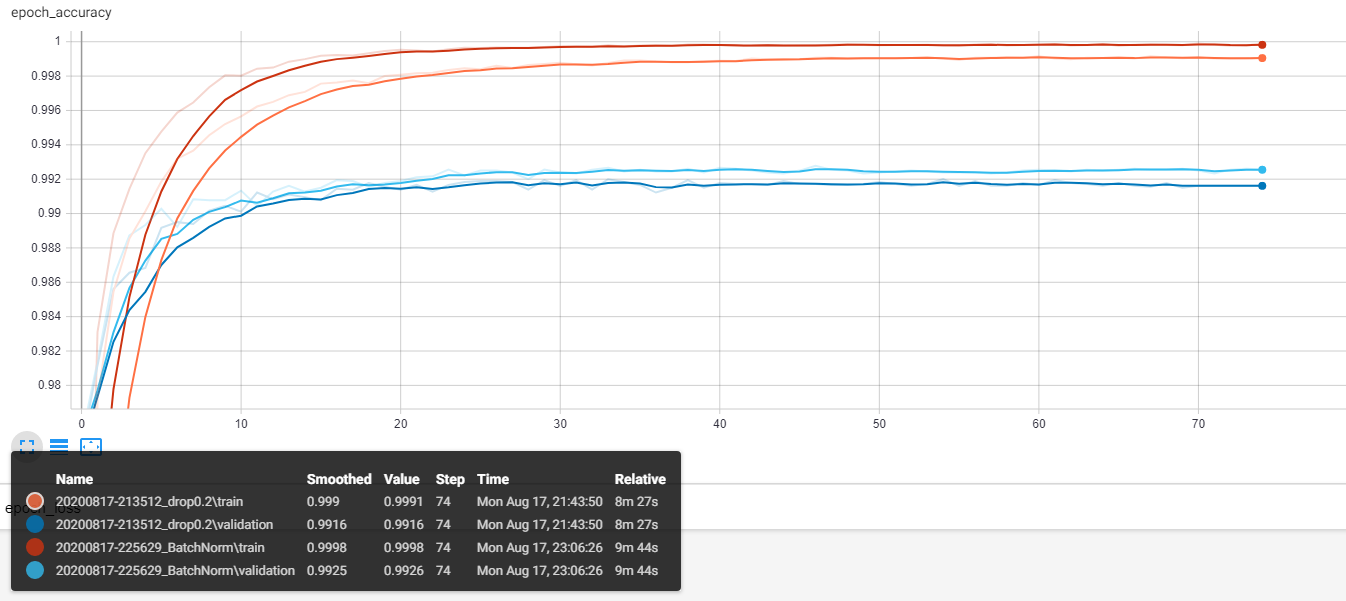


Figure 2 - with and without batch normalization

With batch norm: Without batch norm:

Train loss: 0.036032  
Train accuracy: 100.00%  
Test loss: 0.056987  
Test accuracy: 99.46%

Train loss: 0.037574  
Train accuracy: 99.98%  
Test loss: 0.053080  
Test accuracy: 99.49%

Although the validation graph of the Batch Normalization has **higher** performance, the Test Accuracy at the evaluation time is **worse**.

Another try –

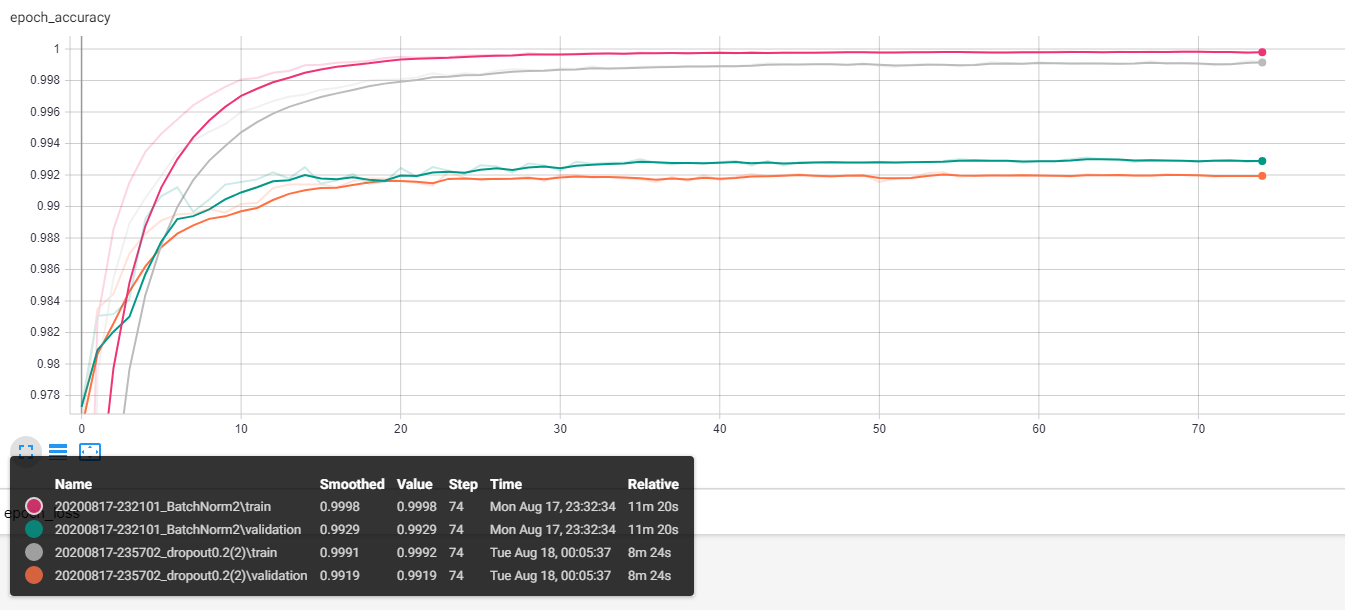


Figure 3 - with and without batch normalization

With batch norm: Without batch norm:

Train loss: 0.036508  
Train accuracy: 100.00%  
Test loss: 0.061265  
Test accuracy: 99.40%

Train loss: 0.036530  
Train accuracy: 99.98%  
Test loss: 0.055017  
Test accuracy: 99.42%

Same results – BN has better validation, but worse test. I’ll use the batch normalization.

**Modifying the Augmentation:**

As we saw, enlarging the training data set helps a lot to train the model.

So I’ve changed the extra 120k samples. I used the ImageDataGenerator package to create these modifications.

Each sample (of the 120k **extra**) has 3 special elements that modified it:

* Rotation: the rotation is up to ±20° from the normal image, so there won’t be samples that will look like the other numbers (e.g. 6 and 9).
* Zoom: the zoom is done by ±0.15. means the image will be ×1.15 of it’s size, or ×0.85 of it’s size. no important data will be lost.
* Noise: added normal noise to the image. Mean is 0, and *std* is 0.02.

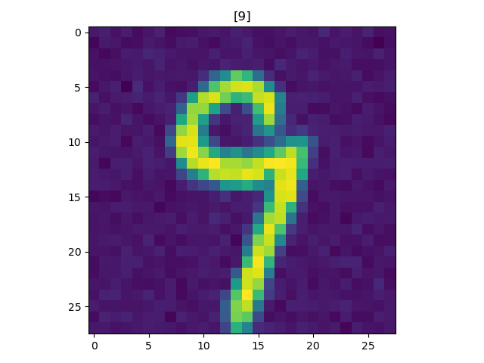
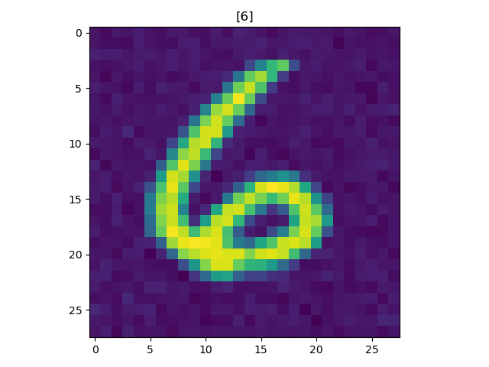
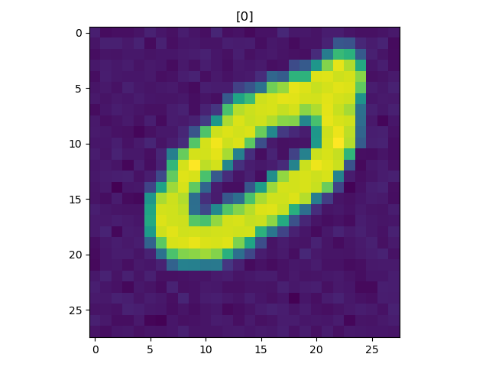


Figure 4 - augmented samples

With dropout rate=0.2 and batch normalization, we got those results:

Try 1: Try 2:

Train loss: 0.042403  
Train accuracy: 100.00%  
Test loss: 0.063537  
Test accuracy: 99.53%

Train loss: 0.041572  
Train accuracy: 100.00%  
Test loss: 0.058892  
Test accuracy: 99.52%

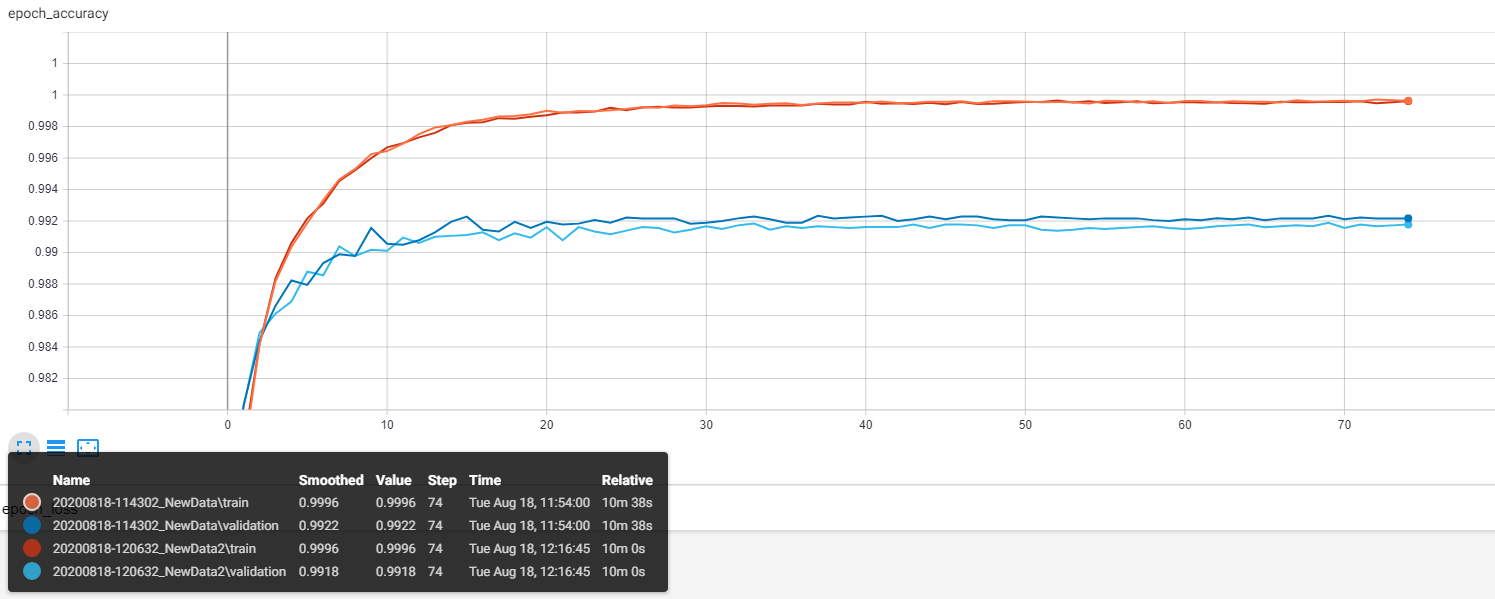


Figure 5 - accuracy with new data

Changing the net:

After we saw dropout 0.2 rate, batch normalization and new data help us improve the model and get to 99.53% test accuracy, I tried to change the net.

* Trying to figure out what is the optimal number of filters for the first convolutional layer:

512 filters

128 filters

32 filters

\*validation

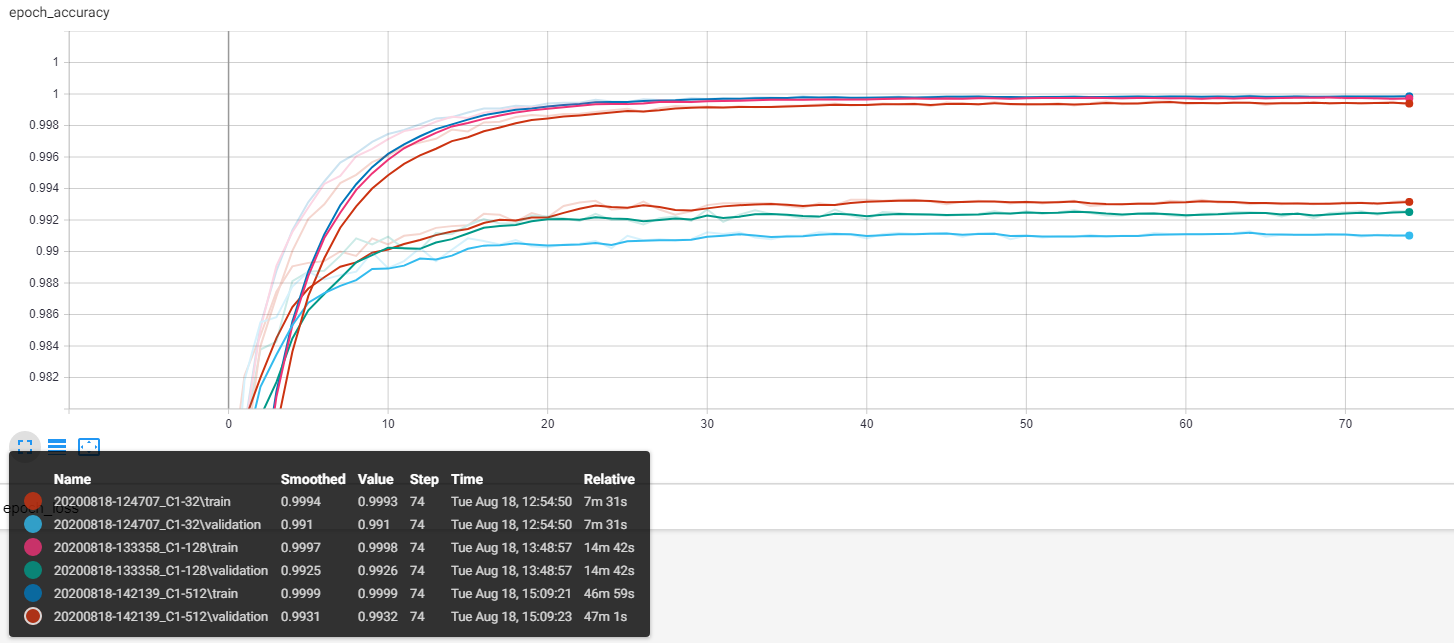


Figure 6 - conv2D layer 1 filters comparison

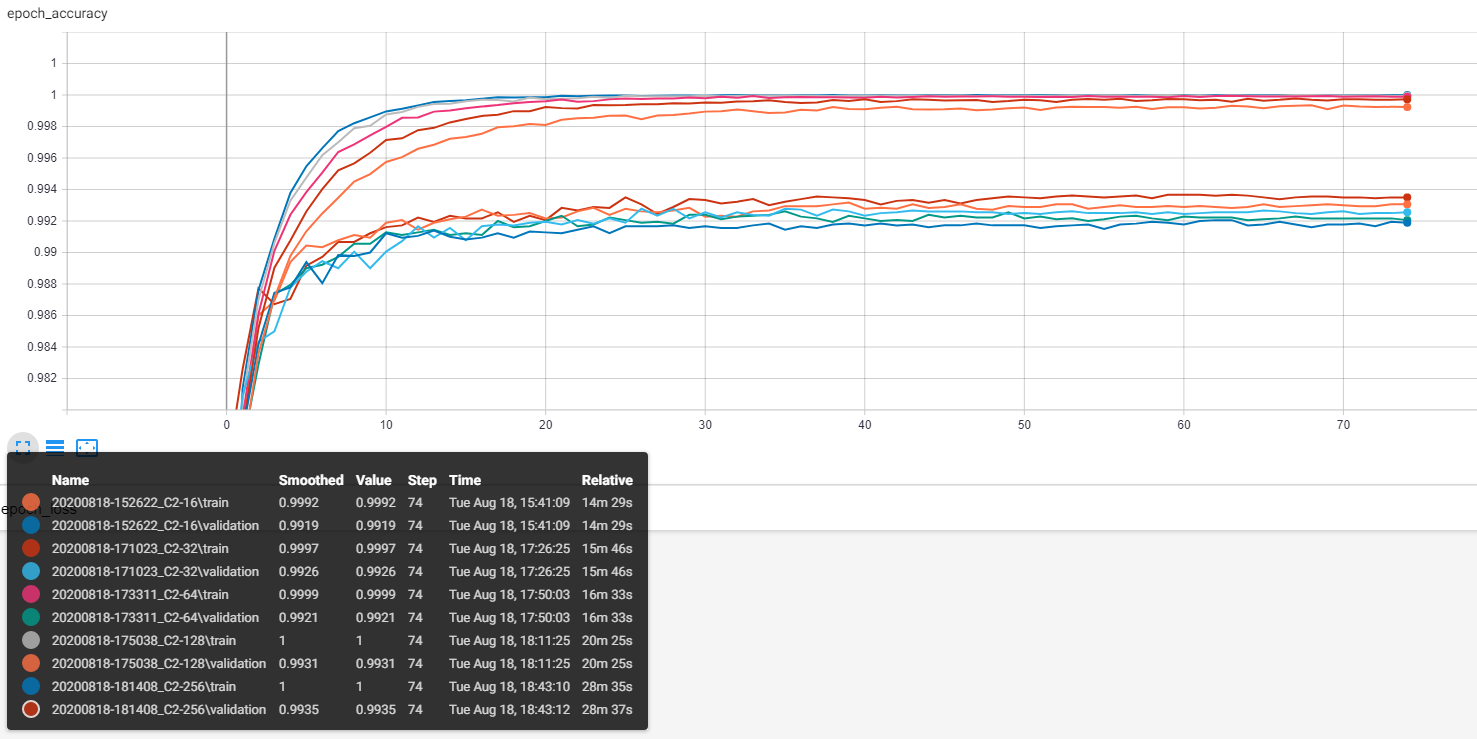
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| C1 - # Filters | 32 | 64 | 128 | 256 | 512 |
| Train Loss | 0.043626 | 0.041960 | 0.038983 | 0.038282 | 0.036689 |
| Train Accuracy | 100% | 100% | 100% | 100% | 100% |
| Test Loss | 0.065591 | 0.063361 | 0.058700 | 0.059314 | 0.056566 |
| Test Accuracy | 99.42% | 99.45% | 99.60% | 99.56% | 99.51% |
| Total Parameters | 443k | 469k | 522k | 629k | 841k |

As we can see, 512 filters will increase the validation accuracy, but in fact, the test data is much better with 128 filters.

From now on I’ll use 128 filters.

* Finding the best number of filters to use at the second convolutional layer (with 128 filters on the first convolutional layer):

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| C2 - # Filters | 16 | 32 | 64 | 128 | 256 |
| Train Loss | 0.034217 | 0.039472 | 0.041440 | 0.034661 | 0.023062 |
| Train Accuracy | 100% | 100% | 100% | 100% | 100% |
| Test Loss | 0.056207 | 0.059474 | 0.061525 | 0.053180 | 0.042047 |
| Test Accuracy | 99.46% | 99.53% | 99.54% | 99.58% | 99.55% |
| Total Parameters | 339k | 522k | 889k | 1.622M | 3M |



256 filters

128 filters

64 filters

32 filters

16 filters

\*validation

Figure 7 - conv2D layer 1 filters comparison

As we can see, there is a huge tradeoff at performance. From layer with 32 filters and 0.5M parameters, to gain more 0.05% accuracy on the test data, we need to increase our model parameter number 3 times!

So I picked the 64 filters which is not bad at performance, and the parameters are not far from 0.5M.

My MNIST state-of-the-art model summary:

Model: "sequential"  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
Layer (type) Output Shape Param #   
=================================================================  
reshape (Reshape) (None, 28, 28, 1) 0   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
conv2d (Conv2D) (None, 24, 24, 128) 3328   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
max\_pooling2d (MaxPooling2D) (None, 12, 12, 128) 0   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
batch\_normalization (BatchNo (None, 12, 12, 128) 512   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
conv2d\_1 (Conv2D) (None, 8, 8, 64) 204864   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
max\_pooling2d\_1 (MaxPooling2 (None, 4, 4, 64) 0   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
flatten (Flatten) (None, 1024) 0   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
batch\_normalization\_1 (Batch (None, 1024) 4096   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
dense (Dense) (None, 512) 524800   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
batch\_normalization\_2 (Batch (None, 512) 2048   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
dropout (Dropout) (None, 512) 0   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
dense\_1 (Dense) (None, 256) 131328   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
batch\_normalization\_3 (Batch (None, 256) 1024   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
dropout\_1 (Dropout) (None, 256) 0   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
dense\_2 (Dense) (None, 64) 16448   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
batch\_normalization\_4 (Batch (None, 64) 256   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
dense\_3 (Dense) (None, 10) 650   
=================================================================  
Total params: 889,354  
Trainable params: 885,386  
Non-trainable params: 3,968  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_