828L HW 2

Julian Vanecek

October 2018

1 General

In general,dropout on dense and max pooling layers seems to be good, Batch normalization on every layer works pretty well and Nesterov momentum helps the model converge quicker.

2 MNIST

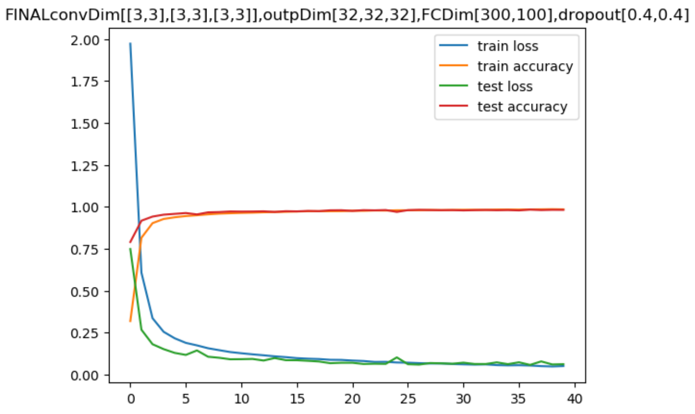


Figure 1: performance of my model

My MNIST architecture has three [3,3] conv layers with 32 channel outputs and max poolings, followed by a fully connected network with two layers of 300 and 100 dimensions and dropout of 0.4. I tried many other architectures in a grid search, and this one won out. I tried fewer conv layers with higher dimensions, differing filter sizes, differing feature sizes. I think the current one makes sense because the small filters find very local patterns, but stacking layers upon each other find patterns of these local patterns. The 3 [3,3] layers have as

large of a window as a single [7,7] layers, but with more parameters available to tune. Dropout reduced overfitting. The loss is cross entropy because we are doing softmax classification.

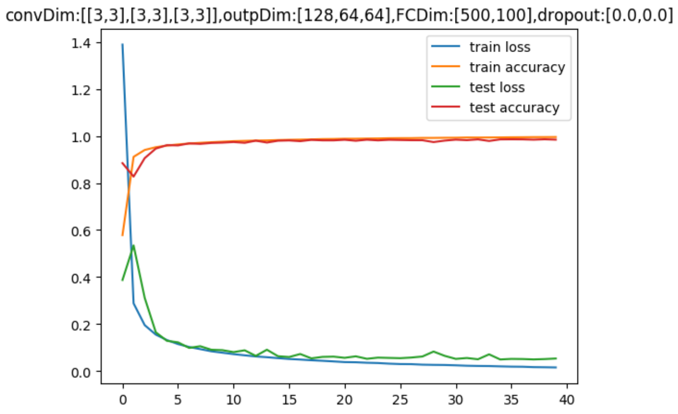
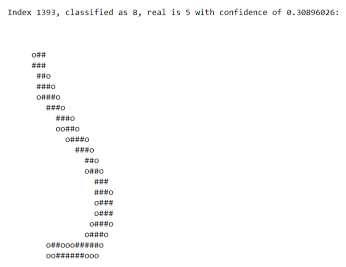


Figure 2: This is a model with no dropout. It begins to overfit. Adding dropout reduces overfitting.



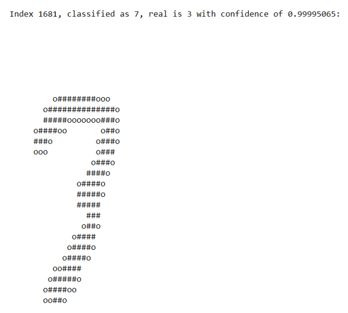
```
Index 1393, classified as 8, real is 5 with confidence of 0.30896026:


o##
###
 ##o
 ###o
 o###o
   ###o
     ###o
     oo##o
       o###o
         ###o
           ##o
           o##o
             ###
             ###o
             o###
             o###
            o###o
           o###o
   o##ooo#####o
   oo######ooo
```

Figure 3: Incorrectly classified a 5 as an 8 with only 30% confidence of choice.

My model ended with 0.9817 test accuracy and got above 95% accuracy after 4 epochs. The final loss is 0.0625.

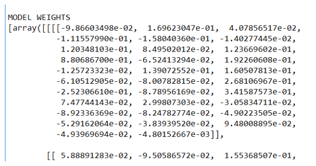
Two examples of incorrectly classified items. The first is with no certainty of which number it was choosing, the second was totally certain of its choice but got it wrong.

And here is a sample of the weights of the system, so you know they are not all zero.



```
Index 1681, classified as 7, real is 3 with confidence of 0.99995065:

    o########ooo
  o##############o
  #####ooooooo###o
o####oo       o##o
###o         o###o
ooo          o###
            o###o
           ####o
         o####o
         #####o
         #####
           ###
          o##o
        o####
       o####o
      o####o
    oo####
   o#####o
  o####oo
  oo##o
```

Figure 4: Incorrectly classified a 3 as a 7 but very 99.9% confident in choice.



```
MODEL WEIGHTS
[array([[[[-9.86603498e-02,  1.69623047e-01,  4.07856517e-02,
           -1.11557990e-01, -1.58040360e-01, -1.40277445e-02,
            1.20348103e-01,  8.49502012e-02,  1.23669602e-01,
            8.80686700e-01, -6.52413294e-02,  1.92260608e-01,
           -1.25723323e-02,  1.39072552e-01,  1.60507813e-01,
           -6.10512905e-02, -8.00782815e-02,  2.68106967e-01,
           -2.52306610e-01, -8.78956169e-02,  3.41587573e-01,
            7.47744143e-02,  2.99807303e-02, -3.05834711e-02,
           -8.92336369e-02, -8.24782774e-02, -4.90223505e-02,
           -5.29162064e-02, -3.83939520e-02,  9.48008895e-02,
           -4.93969694e-02, -4.80152667e-03]],

         [[ 5.88891283e-02, -9.50586572e-02,  1.55368507e-01,
```

Figure 5: sample weights of MNIST model

3 Fashion MNIST

My model has conv dimensions [[3,3],[3,3],[5,5],[5,5]] with only a single max pooling layer between the last and penultimate conv layers (I read a piece from Stanford saying max pooling layers are overrated and should probably be phased out). I increase the filter size because I didn't want more layers to train, but I wanted to reduce the dimensions of the image down to [1,1,channels]. Each [3,3] layer produces 32 features and the [5,5] layers 64 features. The output fits right into the fully connect portion, with 64, 16 and 10 hidden layers, before the softmax layer with 10 options. Each fully connected layer has a dropout of 0.4. Batch normalization is after every layer of the network. The loss is cross entropy because we are doing softmax classification.

This network ended with 0.9254 test set accuracy with 0.31375 loss.

As you can see the model begins to overfit after 40 epochs, and the test accuracy hovers around 92.5% from that point on, with test losses from the high 0.20s into the low 0.30s. Incorrectly classified examples are below.

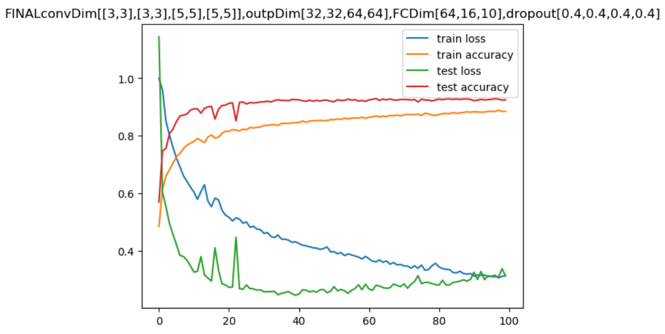


Figure 6: Training results for this model.

```
Index 6474, classified as 3, real is 2 with confidence of 0.3016911:
```

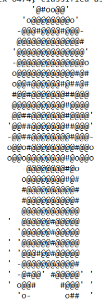


Figure 7: Example incorrectly classified. Not confident in its choice, with 30%.

```
Index 2914, classified as 9, real is 7 with confidence of 0.9999943:
```

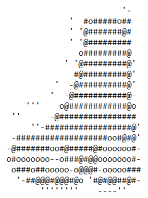


Figure 8: Example correctly classified with near 100% confidence.

```
MODEL WEIGHTS
[array([[[[-4.27923948e-01,  9.62779392e-03, -8.31191123e-01,
          -7.99232483e-01, -1.45344675e-01,  1.05459893e+00,
           1.44753456e-02,  1.15395033e+00, -2.36827657e-01,
           1.41534775e-01,  1.04548849e-01,  7.79607669e-02,
           3.28484505e-01, -1.73104212e-01,  1.04833484e+00,
          -1.68264300e-01,  7.59569108e-01, -1.35882592e+00,
           1.15239310e+00,  1.11014187e+00, -4.69114892e-02,
          -1.78517044e-01, -3.35915610e-02, -6.74681738e-02,
           1.27754807e+00, -4.74786013e-01,  1.35587931e-01,
           4.04229283e-01, -5.09168386e-01, -3.33613902e-01,
          -1.04170449e-01, -9.94194597e-02]],

        [[-3.39428306e-01,  1.53201506e-01,  6.11036777e-01,
           1.03618085e+00, -1.74358133e-02, -1.97285697e-01,
```

Figure 9: Some weights from the model so you know they arent all zero.

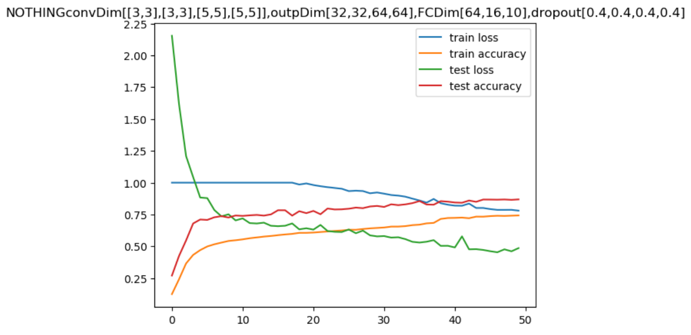


Figure 10: This is the training on the model if we remove momentum, batch normalization and max pooling layers. Test acc. never surpasses 90% over the duration of training.

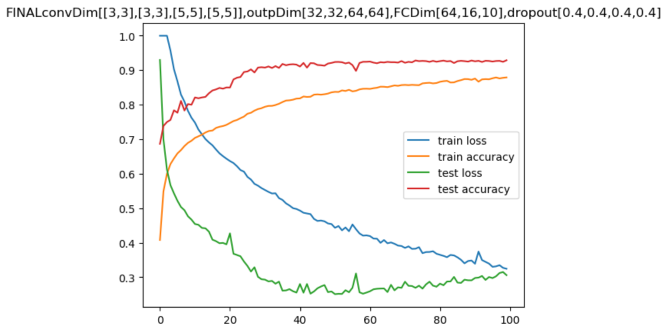


Figure 11: This is the training on the model if we only remove momentum and max pooling layers. It looks pretty good, but it takes twice as long to converge. This needs 27 epochs to get above 90% test acc., in comparisson with 15 for the best model; and converges to 92.5% at 75 epochs instead of 40 epochs.

4 BreastCancer Rates

This was the strangest set. Using grid search over number of hidden layers [1 to 3] and node sizes [10 to 100], it seemed like each model converged to the local minima where ever the initial parameters happened to be. Running it 15 times, the model usually converges with 95-98% test accuracy. One of the fifteen times will converge with about 100% accuracy. I of course chose a model with 100% accuracy for my writeup.

The best model happened to be one with two fully connected layers with 10 nodes each. The loss function is MSE, basically because it's a binary classification with a single node. It uses Nesterov mo-

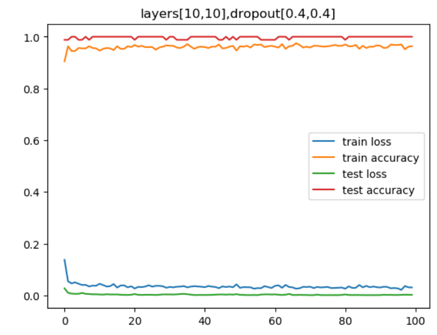
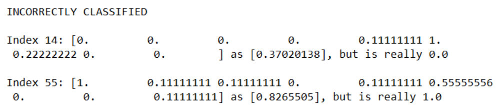
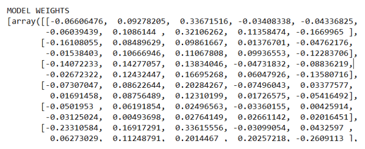


Figure 12: Convergence rates of my feed forward model.



```
INCORRECTLY CLASSIFIED

Index 14: [0.         0.         0.         0.         0.11111111 1.
 0.22222222 0.         0.         ] as [0.37020138], but is really 0.0

Index 55: [1.         0.11111111 0.11111111 0.         0.11111111 0.55555556
 0.         0.         0.11111111] as [0.8265505], but is really 1.0
```

Figure 13: While everything was correctly classified, these are two (correctly) classified inputs with abnormally low confidence.



```
MODEL WEIGHTS
[array([[-0.06606476,  0.09278205,  0.33671516, -0.03408338, -0.04336825,
        -0.06039439,  0.1086144 ,  0.32106262,  0.11358474, -0.1669965 ],
       [-0.16108055,  0.08489629,  0.09861667,  0.01376701, -0.04762176,
        -0.01538403,  0.10666946,  0.11067808,  0.09936553, -0.12283706],
       [-0.14072233,  0.14277057,  0.13834046, -0.04731832, -0.08836219,
        -0.02672322,  0.12432447,  0.16695268,  0.06047926, -0.13580716],
       [-0.07307047,  0.08622644,  0.20284267, -0.07496043,  0.03377577,
         0.01691458,  0.08756489,  0.12310199,  0.01726575, -0.05416492],
       [-0.0501953 ,  0.06191854,  0.02496563, -0.03360155,  0.00425914,
        -0.03125024,  0.00493698,  0.02764149,  0.02661142,  0.02016451],
       [-0.23310584,  0.16917291,  0.33615556, -0.03099054,  0.0432597 ,
         0.06273029,  0.11248791,  0.2014467 ,  0.20257218, -0.2609113 ],
```

Figure 14: Sample model weights so not everything is zero.

mentum, weight decay and batch normalization. I thought adding in momentum would help break over the local minima, but it didn't. Test accuracy ended with 100%. Test loss ended with 0.00128. Train accuracy ended with 0.9632 interestingly enough.