Deep learning for classification on the MNIST dataset

Table of Contents

Prepare the dataset

In this example, we will perform image classification using deep learning (from scratch) on the MNIST dataset. The MNIST dataset is a set of handwritten digits categorized 0-9 and is available at http://yann.lecun.com/exdb/mnist/.

```
clear
addpath(genpath(pwd))
```

The following line will download (if necessary) and prepare the dataset to use in MATLAB.

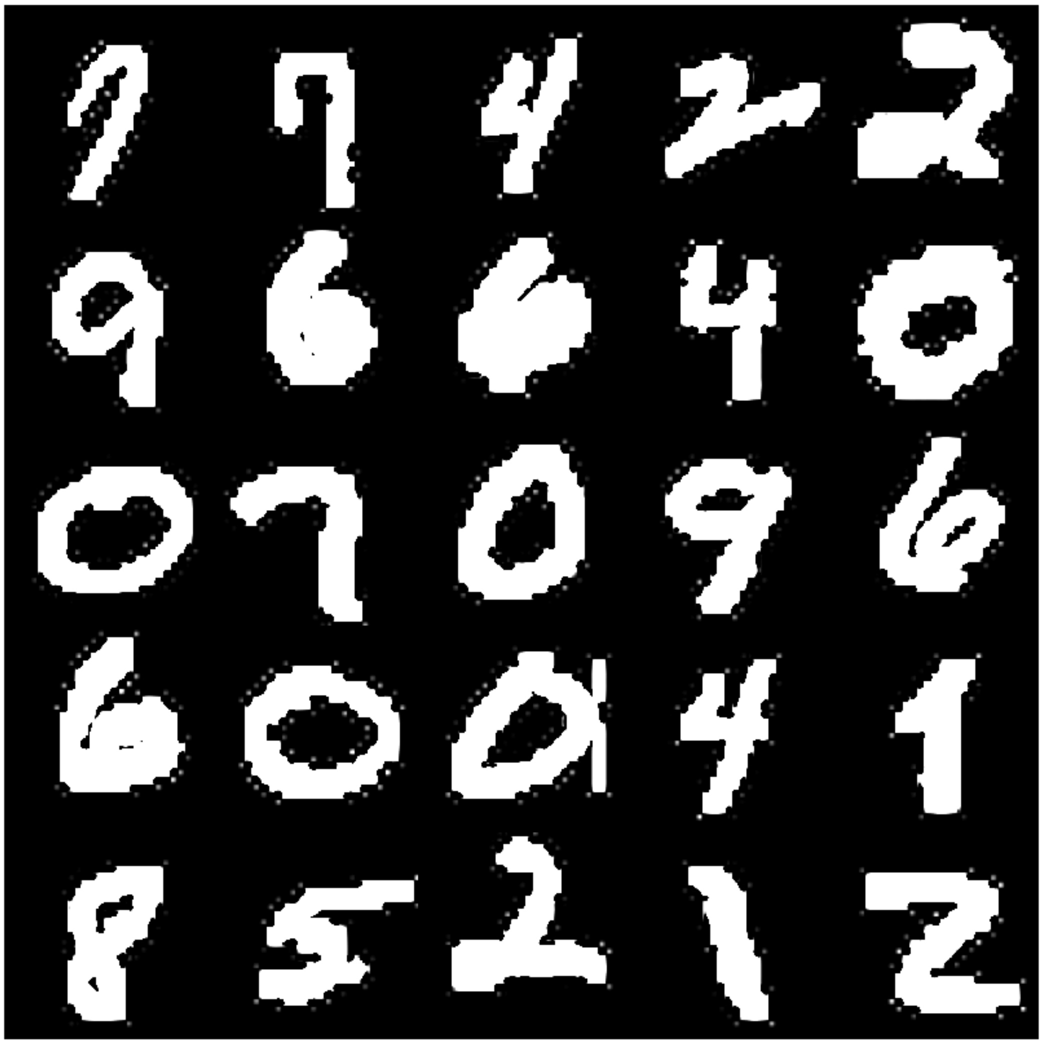
```
[imgDataTrain, labelsTrain, imgDataTest, labelsTest] = prepareData;
```

```
Preparing MNIST data...
MNIST data preparation complete.
```

Let's look at a few of the images

We will pick a random sampling of the images to take a closer look. Run this section a few times to see different picture montages. "viewDigitMontage" is a custom function found in the "03-HelperFunctions" folder.

```
viewDigitMontage(labelsTrain, imgDataTrain)
```



How do we classify a digit?

We'll start with the end goal first, and use a model already trained to classify images with MNIST, so we can see what we're trying to achieve.

```
load MNISTModel
```

Note: You can execute this section of the script multiple times and get a random classification image each time.

```
% Predict the class of an image
randIndx = randi(numel(labelsTest));
img = imgDataTest(:,:,1,randIndx);
actualLabel = labelsTest(randIndx);

predictedLabel = classify(net,img);
imshow(img);
title(['Predicted: ' char(predictedLabel) ', Actual: ' char(actualLabel)])
```

Predicted: 6, Actual: 6

Prepare the CNN

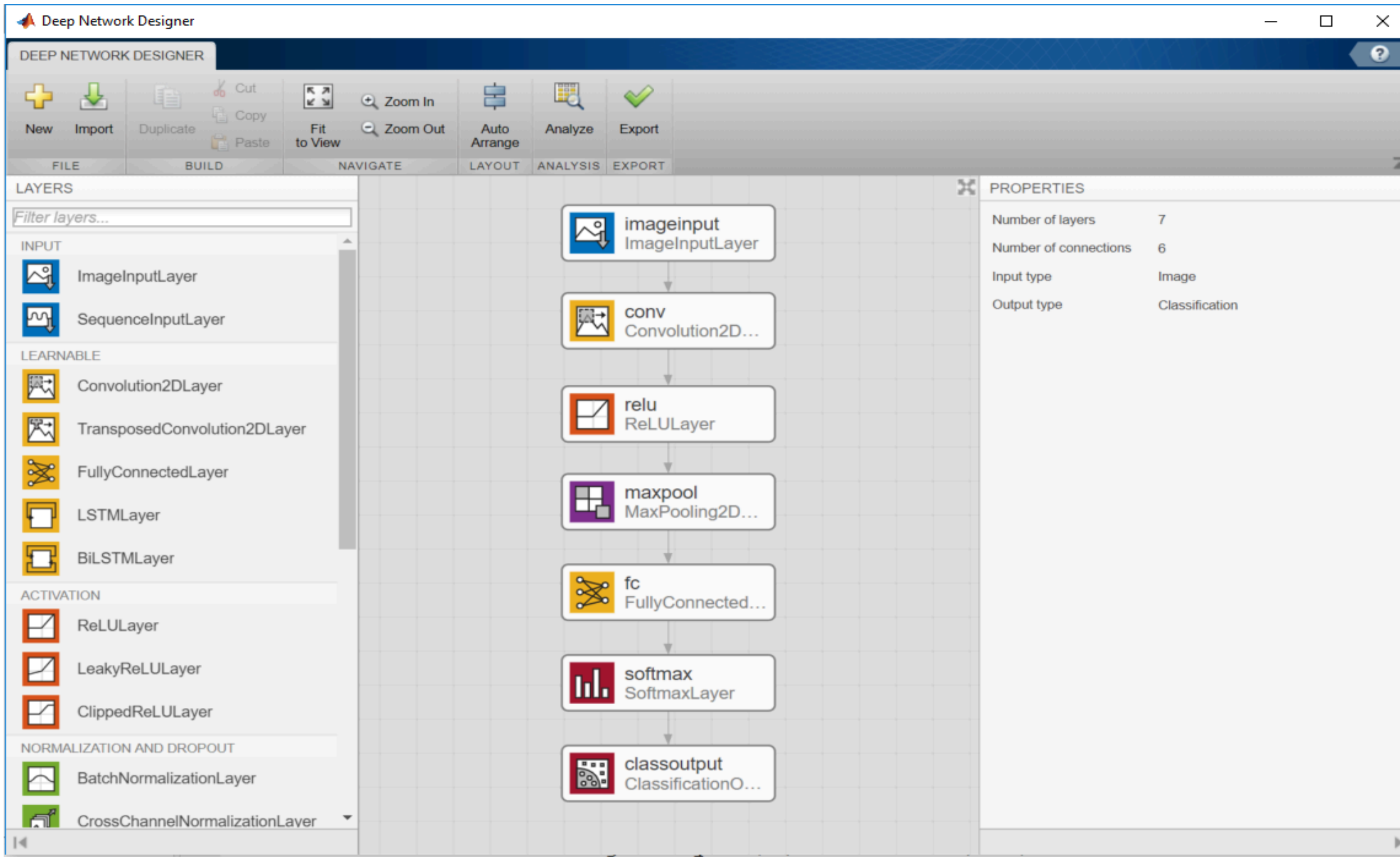
We found an example of a convolutional neural network (CNN) in the documentation. One of the simplest possible convnets, it contains one convolutional layer, one ReLU, one pooling layer, and one fully connected layer. Let's utilize it as our first attempt at classifying these images.

```
layers = [  imageInputLayer([28 28 1])
            convolution2dLayer(5,20)
            reluLayer
            maxPooling2dLayer(2, 'Stride', 2)
            fullyConnectedLayer(10)
            softmaxLayer
            classificationLayer   ]
```

```
layers = 
  7x1 Layer array with layers:

     1   ''   Image Input             28x28x1 images with 'zerocenter' normalization
     2   ''   Convolution             20 5x5 convolutions with stride [1  1] and padding [0  0  0  0]
     3   ''   ReLU                    ReLU
     4   ''   Max Pooling             2x2 max pooling with stride [2  2] and padding [0  0  0  0]
     5   ''   Fully Connected         10 fully connected layer
     6   ''   Softmax                 softmax
     7   ''   Classification Output   crossentropyex
```

Deep Network Designer

You can also create networks using the Deep Network Designer. This app provides a drag-and-drop interface, which lets you define layer architecture, layer parameters, and layer connections. After creating a network, you can check it for errors then export it to the workspace. Try to create the network we defined above!



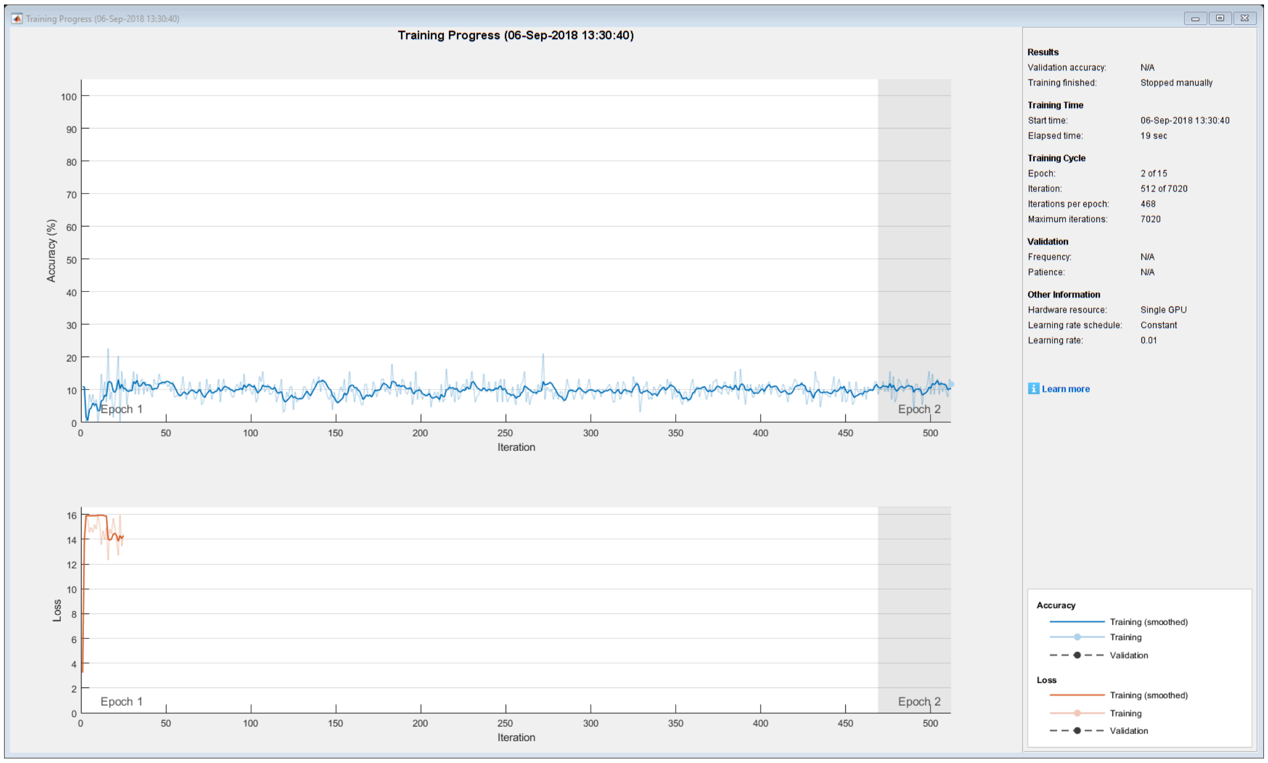
Number of layers	7
Number of connections	6
Input type	Image
Output type	Classification

```
% deepNetworkDesigner
```

Attempt 1: Set training options and train the network

Let's use default values and try training this network.

```
options = trainingOptions( 'sgdm',...
    'Plots', 'training-progress',...
    'MaxEpochs',10);

net = trainNetwork(imgDataTrain, labelsTrain, layers, options);
```



Results

Validation accuracy:	N/A
Training finished:	Stopped manually

Training Time

Start time:	06-Sep-2018 13:30:40
Elapsed time:	19 sec

Training Cycle

Epoch:	2 of 15
Iteration:	512 of 7020
Iterations per epoch:	468
Maximum iterations:	7020

Validation

Frequency:	N/A
Patience:	N/A

Other Information

Hardware resource:	Single GPU
Learning rate schedule:	Constant
Learning rate:	0.01

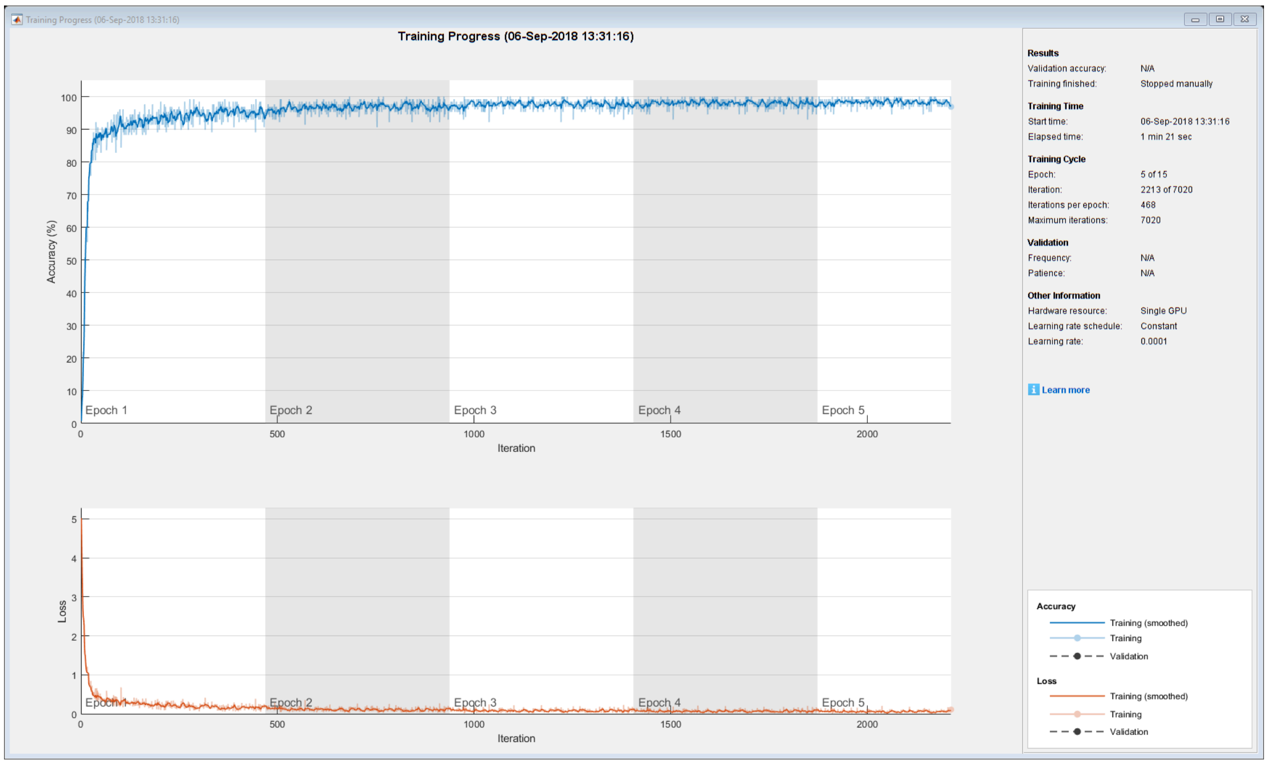
Learn more

```
Training on single GPU.
Initializing image normalization.
|=======================================================================================|
|  Epoch  |  Iteration  |  Time Elapsed  |  Mini-batch  |  Mini-batch  |  Base Learning  |
|         |             |   (hh:mm:ss)   |   Accuracy   |     Loss     |      Rate       |
|=======================================================================================|
|       1 |           1 |       00:00:01 |       10.94% |       3.2426 |          0.0100 |
|       1 |          50 |       00:00:03 |       12.50% |          NaN |          0.0100 |
|       1 |         100 |       00:00:05 |        9.38% |          NaN |          0.0100 |
|       1 |         150 |       00:00:06 |        7.03% |          NaN |          0.0100 |
|       1 |         200 |       00:00:08 |        8.59% |          NaN |          0.0100 |
|       1 |         250 |       00:00:10 |        8.59% |          NaN |          0.0100 |
|       1 |         300 |       00:00:12 |       10.16% |          NaN |          0.0100 |
|       1 |         350 |       00:00:13 |        7.81% |          NaN |          0.0100 |
|       1 |         400 |       00:00:15 |       10.16% |          NaN |          0.0100 |
|       1 |         450 |       00:00:17 |        6.25% |          NaN |          0.0100 |
|       2 |         500 |       00:00:18 |        8.59% |          NaN |          0.0100 |
|       2 |         512 |       00:00:19 |       11.72% |          NaN |          0.0100 |
|=======================================================================================|
```

Attempt 2: Change the learning rate

That didn't work too well. Let's try changing one of the main parameters that affects neural network training: the **learning rate.**

```
options = trainingOptions('sgdm',...
    'Plots', 'training-progress',...
    'MaxEpochs',10,...
    'InitialLearnRate', 0.0001);

net = trainNetwork(imgDataTrain, labelsTrain, layers, options);
```



Results
Validation accuracy: N/A
Training finished: Stopped manually

Training Time
Start time: 06-Sep-2018 13:31:16
Elapsed time: 1 min 21 sec

Training Cycle
Epoch: 5 of 15
Iteration: 2213 of 7020
Iterations per epoch: 468
Maximum iterations: 7020

Validation
Frequency: N/A
Patience: N/A

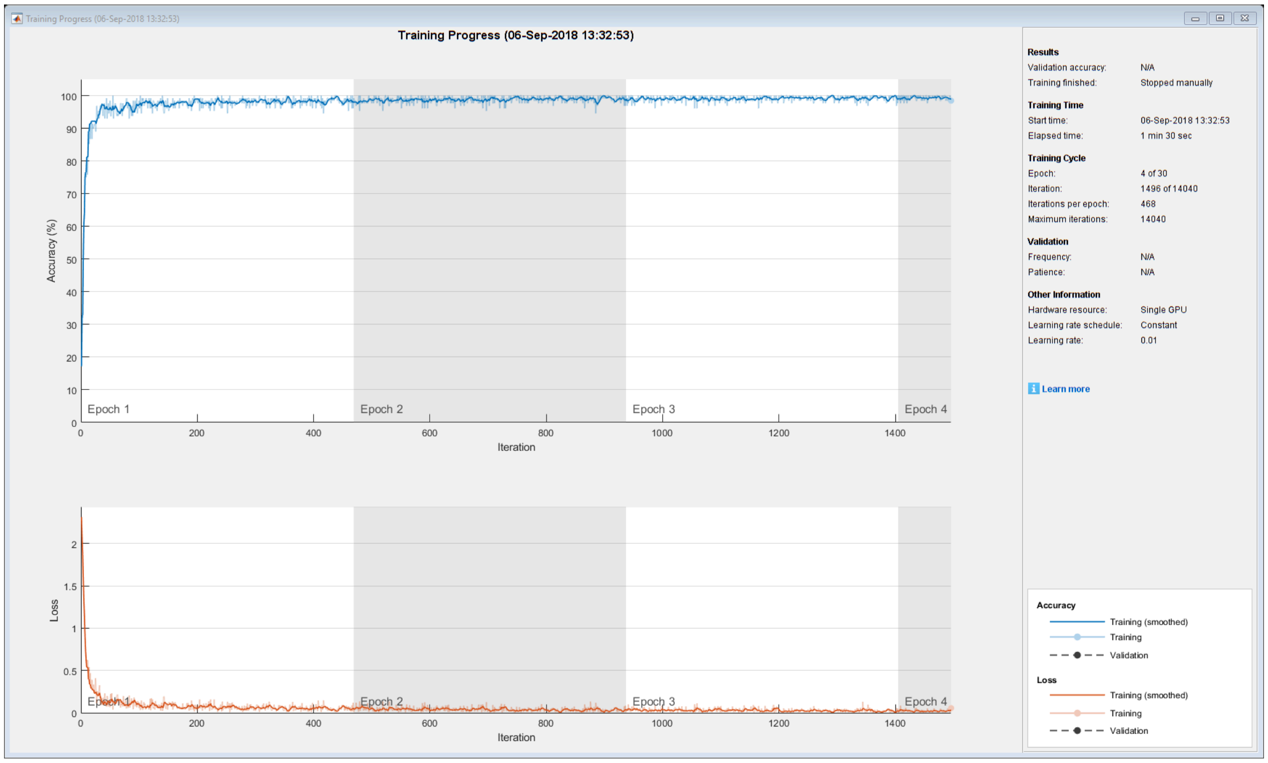
Other Information
Hardware resource: Single GPU
Learning rate schedule: Constant
Learning rate: 0.0001

```
Training on single GPU.
Initializing image normalization.
|========================================================================================|
|  Epoch  |  Iteration  |  Time Elapsed  |  Mini-batch  |  Mini-batch  |  Base Learning  |
|         |             |   (hh:mm:ss)   |   Accuracy   |     Loss     |      Rate       |
|========================================================================================|
|       1 |           1 |       00:00:01 |        0.78% |       5.0161 |      1.0000e-04 |
|       1 |          50 |       00:00:03 |       88.28% |       0.3062 |      1.0000e-04 |
|       1 |         100 |       00:00:05 |       92.19% |       0.2768 |      1.0000e-04 |
|       1 |         150 |       00:00:07 |       90.63% |       0.3044 |      1.0000e-04 |
|       1 |         200 |       00:00:08 |       94.53% |       0.2589 |      1.0000e-04 |
|       1 |         250 |       00:00:10 |       94.53% |       0.2127 |      1.0000e-04 |
|       1 |         300 |       00:00:12 |       93.75% |       0.1443 |      1.0000e-04 |
|       1 |         350 |       00:00:14 |       98.44% |       0.1178 |      1.0000e-04 |
|       1 |         400 |       00:00:16 |       97.66% |       0.0685 |      1.0000e-04 |
|       1 |         450 |       00:00:18 |       94.53% |       0.1546 |      1.0000e-04 |
|       2 |         500 |       00:00:20 |       96.09% |       0.1547 |      1.0000e-04 |
|       2 |         550 |       00:00:22 |       96.09% |       0.0892 |      1.0000e-04 |
|       2 |         600 |       00:00:23 |       99.22% |       0.0385 |      1.0000e-04 |
|       2 |         650 |       00:00:25 |       98.44% |       0.0589 |      1.0000e-04 |
|       2 |         700 |       00:00:27 |       97.66% |       0.0884 |      1.0000e-04 |
|       2 |         750 |       00:00:29 |       96.09% |       0.1464 |      1.0000e-04 |
|       2 |         800 |       00:00:30 |       96.88% |       0.1206 |      1.0000e-04 |
|       2 |         850 |       00:00:32 |       95.31% |       0.1005 |      1.0000e-04 |
|       2 |         900 |       00:00:34 |       96.88% |       0.0905 |      1.0000e-04 |
|       3 |         950 |       00:00:36 |       97.66% |       0.0721 |      1.0000e-04 |
|       3 |        1000 |       00:00:37 |       96.88% |       0.0797 |      1.0000e-04 |
|       3 |        1050 |       00:00:39 |       98.44% |       0.0481 |      1.0000e-04 |
|       3 |        1100 |       00:00:41 |       96.09% |       0.0633 |      1.0000e-04 |
|       3 |        1150 |       00:00:43 |       95.31% |       0.1402 |      1.0000e-04 |
|       3 |        1200 |       00:00:45 |       98.44% |       0.1057 |      1.0000e-04 |
|       3 |        1250 |       00:00:46 |       97.66% |       0.1229 |      1.0000e-04 |
|       3 |        1300 |       00:00:48 |       98.44% |       0.0770 |      1.0000e-04 |
|       3 |        1350 |       00:00:50 |       96.09% |       0.1004 |      1.0000e-04 |
|       3 |        1400 |       00:00:52 |       97.66% |       0.0709 |      1.0000e-04 |
|       4 |        1450 |       00:00:54 |       99.22% |       0.0348 |      1.0000e-04 |
|       4 |        1500 |       00:00:55 |       98.44% |       0.0381 |      1.0000e-04 |
|       4 |        1550 |       00:00:57 |       98.44% |       0.0619 |      1.0000e-04 |
|       4 |        1600 |       00:00:59 |       97.66% |       0.0441 |      1.0000e-04 |
|       4 |        1650 |       00:01:01 |       98.44% |       0.0478 |      1.0000e-04 |
|       4 |        1700 |       00:01:02 |       97.66% |       0.0851 |      1.0000e-04 |
|       4 |        1750 |       00:01:04 |       96.09% |       0.1226 |      1.0000e-04 |
|       4 |        1800 |       00:01:06 |      100.00% |       0.0272 |      1.0000e-04 |
|       4 |        1850 |       00:01:08 |       98.44% |       0.0564 |      1.0000e-04 |
|       5 |        1900 |       00:01:10 |       96.09% |       0.1190 |      1.0000e-04 |
|       5 |        1950 |       00:01:11 |       97.66% |       0.0767 |      1.0000e-04 |
|       5 |        2000 |       00:01:13 |       97.66% |       0.0629 |      1.0000e-04 |
|       5 |        2050 |       00:01:15 |       99.22% |       0.0267 |      1.0000e-04 |
|       5 |        2100 |       00:01:17 |       98.44% |       0.0414 |      1.0000e-04 |
|       5 |        2150 |       00:01:19 |       97.66% |       0.0838 |      1.0000e-04 |
|       5 |        2200 |       00:01:20 |       98.44% |       0.0897 |      1.0000e-04 |
|       5 |        2213 |       00:01:21 |       96.88% |       0.1106 |      1.0000e-04 |
|=========================================================================================|
```

Attempt 3: Change the network architecture

Sometimes, it's not the learning rate that needs changing. Other factors about the model can affect the training as well. Here, instead of playing with hyperparameters such as the learning rate, we can use a more sophisticated CNN in the first place. This one has multiple convolution layers, as well as **batch normalization** layers, which improve the quality and convergence rate of the training.

```
layers = [
    imageInputLayer([28 28])

    convolution2dLayer(3,16,'Padding',1)
    batchNormalizationLayer
    reluLayer

    maxPooling2dLayer(2,'Stride',2)

    convolution2dLayer(3,32,'Padding',1)
    batchNormalizationLayer
    reluLayer

    maxPooling2dLayer(2,'Stride',2)

    convolution2dLayer(3,64,'Padding',1)
    batchNormalizationLayer
    reluLayer

    fullyConnectedLayer(10)
    softmaxLayer
    classificationLayer];

options = trainingOptions('sgdm',...
    'Plots', 'training-progress');

net = trainNetwork(imgDataTrain, labelsTrain, layers, options);
```



Results	
Validation accuracy:	N/A
Training finished:	Stopped manually
Training Time	
Start time:	06-Sep-2018 13:32:53
Elapsed time:	1 min 30 sec
Training Cycle	
Epoch:	4 of 30
Iteration:	1496 of 14040
Iterations per epoch:	468
Maximum iterations:	14040
Validation	
Frequency:	N/A
Patience:	N/A
Other Information	
Hardware resource:	Single GPU
Learning rate schedule:	Constant
Learning rate:	0.01

```
Training on single GPU.
Initializing image normalization.
|========================================================================================|
|  Epoch  |  Iteration  |  Time Elapsed  |  Mini-batch  |  Mini-batch  |  Base Learning  |
|         |             |   (hh:mm:ss)   |   Accuracy   |     Loss     |      Rate       |
|========================================================================================|
|       1 |           1 |       00:00:01 |       17.19% |       2.3120 |          0.0100 |
|       1 |          50 |       00:00:04 |       98.44% |       0.0682 |          0.0100 |
|       1 |         100 |       00:00:07 |       98.44% |       0.0579 |          0.0100 |
|       1 |         150 |       00:00:10 |       96.88% |       0.0994 |          0.0100 |
|       1 |         200 |       00:00:13 |       99.22% |       0.0275 |          0.0100 |
|       1 |         250 |       00:00:16 |       98.44% |       0.0594 |          0.0100 |
|       1 |         300 |       00:00:19 |       99.22% |       0.0216 |          0.0100 |
|       1 |         350 |       00:00:22 |       98.44% |       0.0457 |          0.0100 |
|       1 |         400 |       00:00:25 |       98.44% |       0.0724 |          0.0100 |
|       1 |         450 |       00:00:28 |       98.44% |       0.0740 |          0.0100 |
|       2 |         500 |       00:00:31 |       97.66% |       0.0890 |          0.0100 |
|       2 |         550 |       00:00:34 |      100.00% |       0.0141 |          0.0100 |
|       2 |         600 |       00:00:37 |       99.22% |       0.0137 |          0.0100 |
|       2 |         650 |       00:00:40 |       98.44% |       0.0517 |          0.0100 |
|       2 |         700 |       00:00:43 |      100.00% |       0.0175 |          0.0100 |
|       2 |         750 |       00:00:46 |      100.00% |       0.0142 |          0.0100 |
|       2 |         800 |       00:00:49 |       99.22% |       0.0615 |          0.0100 |
|       2 |         850 |       00:00:52 |       99.22% |       0.0289 |          0.0100 |
|       2 |         900 |       00:00:55 |       99.22% |       0.0212 |          0.0100 |
|       3 |         950 |       00:00:58 |       97.66% |       0.0825 |          0.0100 |
|       3 |        1000 |       00:01:01 |       99.22% |       0.0192 |          0.0100 |
|       3 |        1050 |       00:01:04 |       99.22% |       0.0333 |          0.0100 |
|       3 |        1100 |       00:01:07 |       98.44% |       0.0810 |          0.0100 |
|       3 |        1150 |       00:01:10 |       99.22% |       0.0165 |          0.0100 |
|       3 |        1200 |       00:01:13 |      100.00% |       0.0158 |          0.0100 |
|       3 |        1250 |       00:01:16 |       98.44% |       0.0401 |          0.0100 |
|       3 |        1300 |       00:01:19 |       98.44% |       0.0542 |          0.0100 |
|       3 |        1350 |       00:01:22 |       97.66% |       0.0514 |          0.0100 |
|       3 |        1400 |       00:01:25 |      100.00% |       0.0051 |          0.0100 |
|       4 |        1450 |       00:01:28 |       98.44% |       0.0214 |          0.0100 |
|       4 |        1496 |       00:01:30 |       98.44% |       0.0564 |          0.0100 |
|========================================================================|
```

Classify the test data set

We have a separate set of images in the MNIST dataset set aside specifically for testing. Let's use the test data to test the accuracy of the model we just trained.

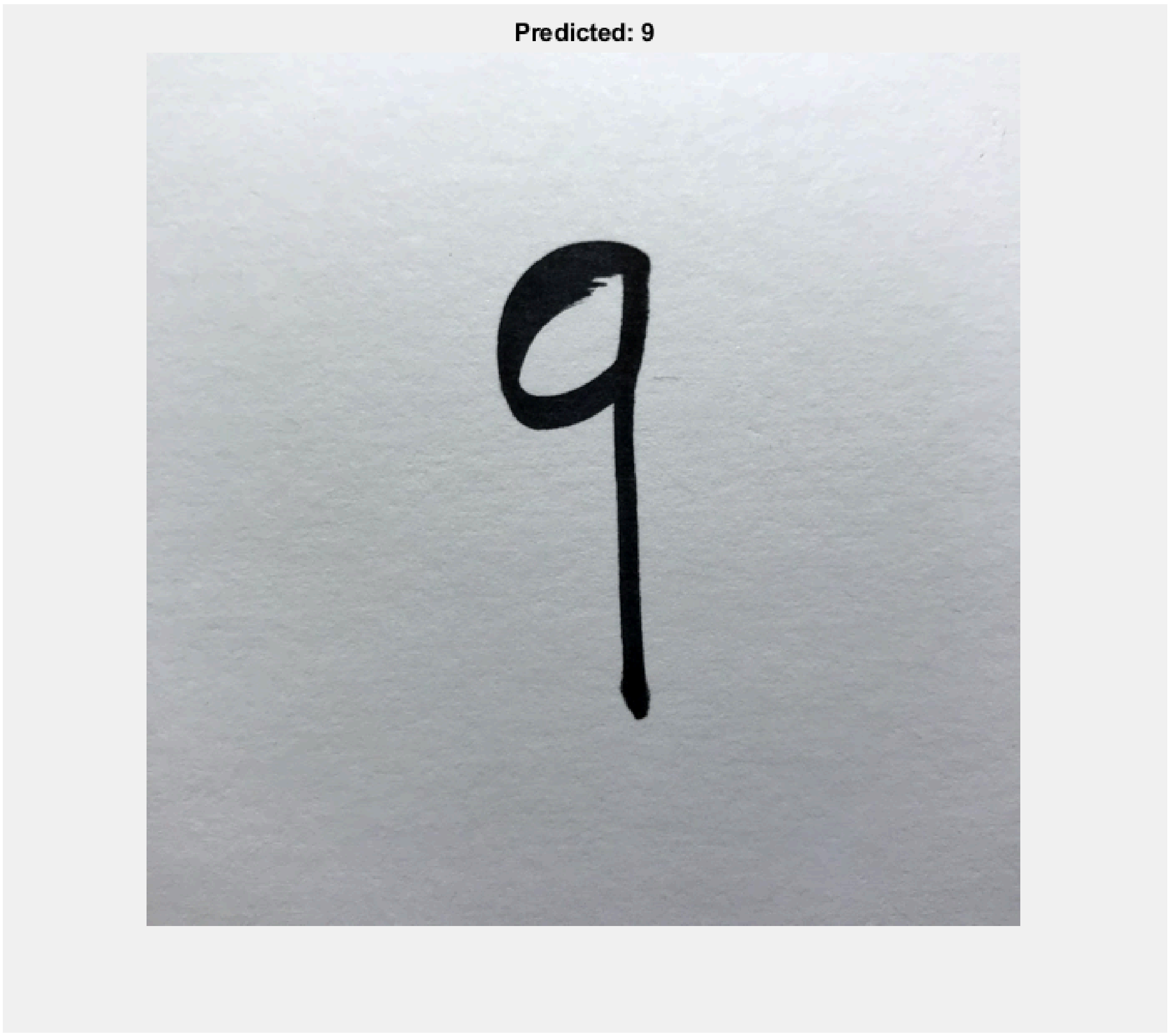
```
predLabelsTest = classify(net,imgDataTest);

testAccuracy = sum(predLabelsTest == labelsTest) / numel(labelsTest)
```

```
testAccuracy = 0.9902
```

Try to classify our own images

Let's try to classify the digits in the "03 - Handwritten Digits" folder

```
imgPath = fullfile('03-HandwrittenDigits');
ids = imageDatastore(imgPath);

f = figure;
f.Visible = 'on';

while hasdata(ids)

    iOriginal = read(ids);
    iFinal = preprocessImage(iOriginal);

    prediction = classify(net,iFinal);

    imshow(imresize(iOriginal,0.5));
    title(['Predicted: ' char(prediction)]);

    pause(1)

end
```



Why change learning rate? (optional aside)

The following app dives into a brief explanation of how the **learning rate** parameter affects training in machine learning algorithms.

```
gradientDescentVisualization
```

Copyright 2019 The MathWorks, Inc.