THE MNIST DATABASE of handwritten digits

Yann LeCun, Courant Institute, NYU
Corinna Cortes, Google Labs, New York
Christopher J.C. Burges, Microsoft Research, Redmond

The MNIST database of handwritten digits, available from this page, has a training set of 60,000 examples, and a test set of 10,000 examples.

It is a subset of a larger set available from NIST.

The digits have been size-normalized and centered in a fixed-size image.

It is a good database for people who want to try learning techniques and pattern recognition methods on real-world data while spending minimal efforts on preprocessing and formatting.

Four files are available on this site:

```
train-images-idx3-ubyte.gz    :    training set images (9912422 bytes)
train-labels-idx1-ubyte.gz    :    training set labels (28881 bytes)
t10k-images-idx3-ubyte.gz     :    test set images     (1648877 bytes)
t10k-labels-idx1-ubyte.gz     :    test set labels     (4542 bytes)
```

以上四个文档为压缩文档，要先解压，为如下四个文档。
please note that your browser may uncompress these files without telling you.

There are 4 files:(**binary files，二进制文档，注意：没有扩展名**)

```
train-images-idx3-ubyte   :  training set images
train-labels-idx1-ubyte   :  training set labels
t10k-images-idx3-ubyte    :  test set images
t10k-labels-idx1-ubyte    :  test set labels
```

If the files you downloaded have a larger size than the above, they have been uncompressed by your browser.

Simply rename them to remove the .gz extension.

Some people have asked me "my application can't open your image files".

These files are not in any standard image format.

You have to write your own (very simple) program to read them.

The file format is described at the bottom of this page.

The original black and white (bilevel) images from NIST were size normalized to fit in a 20x20 pixel box while preserving their aspect ratio.

The resulting images contain grey levels as a result of the anti-aliasing technique used by the normalization algorithm.

the images were centered in a 28x28 image by computing the center of mass of the pixels, and translating the image so as to position this point at the center of the 28x28 field.

With some classification methods (particuarly template-based methods, such as SVM and K-nearest neighbors), the error rate improves when the digits are centered by bounding box rather than center of mass.

If you do this kind of pre-processing, you should report it in your publications.

The MNIST database was constructed from NIST's Special Database 3 and Special Database 1 which contain binary images of handwritten digits.

NIST originally designated SD-3 as their training set and SD-1 as their test set.

However, SD-3 is much cleaner and easier to recognize than SD-1.

The reason for this can be found on the fact that SD-3 was collected among Census Bureau employees, while SD-1 was collected among high-school students.

Drawing sensible conclusions from learning experiments requires that the result be independent of the choice of training set and test among the complete set of samples.

Therefore it was necessary to build a new database by mixing NIST's datasets.

The MNIST training set is composed of 30,000 patterns from SD-3 and 30,000 patterns from SD-1.

Our test set was composed of 5,000 patterns from SD-3 and 5,000 patterns from SD-1.

The 60,000 pattern training set contained examples from approximately 250 writers.

We made sure that the sets of writers of the training set and test set were disjoint.

SD-1 contains 58,527 digit images written by 500 different writers.

In contrast to SD-3, where blocks of data from each writer appeared in sequence, the data in SD-1 is scrambled.

Writer identities for SD-1 is available and we used this information to unscramble the writers.

We then split SD-1 in two: characters written by the first 250 writers went into our new training set.

The remaining 250 writers were placed in our test set.

Thus we had two sets with nearly 30,000 examples each.

The new training set was completed with enough examples from SD-3, starting at pattern # 0, to make a full set of 60,000 training patterns.

Similarly, the new test set was completed with SD-3 examples starting at pattern # 35,000 to make a full set with 60,000 test patterns.

Only a subset of 10,000 test images (5,000 from SD-1 and 5,000 from SD-3) is available on this site.

The full 60,000 sample training set is available.

Many methods have been tested with this training set and test set. Here are a few examples. Details about the methods are given in an upcoming paper. Some of those experiments used a version of the database where the input images where deskewed (by computing the principal axis of the shape that is closest to the vertical, and shifting the lines so as to make it vertical). In some other experiments, the training set was augmented with artificially distorted versions of the original training samples. The distortions are random combinations of shifts, scaling, skewing, and compression.

CLASSIFIER	PREPROCESSING	TEST ERROR RATE (%)	Reference
Linear Classifiers			
linear classifier (1-layer NN)	none	12.0	LeCun et al. 1998
linear classifier (1-layer NN)	deskewing	8.4	LeCun et al. 1998
pairwise linear classifier	deskewing	7.6	LeCun et al. 1998
K-Nearest Neighbors			
K-nearest-neighbors, Euclidean (L2)	none	5.0	LeCun et al. 1998
K-nearest-neighbors, Euclidean (L2)	none	3.09	Kenneth Wilder, U. Chicago
K-nearest-neighbors, L3	none	2.83	Kenneth Wilder, U. Chicago
K-nearest-neighbors, Euclidean (L2)	deskewing	2.4	LeCun et al. 1998
K-nearest-neighbors, Euclidean (L2)	deskewing, noise removal, blurring	1.80	Kenneth Wilder, U. Chicago
K-nearest-neighbors, L3	deskewing, noise removal, blurring	1.73	Kenneth Wilder, U. Chicago
K-nearest-neighbors, L3	deskewing, noise removal, blurring, 1 pixel shift	1.33	Kenneth Wilder, U. Chicago
K-nearest-neighbors, L3	deskewing, noise removal, blurring, 2 pixel shift	1.22	Kenneth Wilder, U. Chicago
K-NN with non-linear deformation (IDM)	shiftable edges	0.54	Keysers et al. IEEE PAMI 2007
K-NN with non-linear deformation (P2DHMDM)	shiftable edges	0.52	Keysers et al. IEEE PAMI 2007

K-NN, Tangent Distance	subsampling to 16x16 pixels	1.1	LeCun et al. 1998
K-NN, shape context matching	shape context feature extraction	0.63	Belongie et al. IEEE PAMI 2002

Boosted Stumps

boosted stumps	none	7.7	Kegl et al., ICML 2009
products of boosted stumps (3 terms)	none	1.26	Kegl et al., ICML 2009
boosted trees (17 leaves)	none	1.53	Kegl et al., ICML 2009
stumps on Haar features	Haar features	1.02	Kegl et al., ICML 2009
product of stumps on Haar f.	Haar features	0.87	Kegl et al., ICML 2009

Non-Linear Classifiers

40 PCA + quadratic classifier	none	3.3	LeCun et al. 1998
1000 RBF + linear classifier	none	3.6	LeCun et al. 1998

SVMs

SVM, Gaussian Kernel	none	1.4	
SVM deg 4 polynomial	deskewing	1.1	LeCun et al. 1998
Reduced Set SVM deg 5 polynomial	deskewing	1.0	LeCun et al. 1998
Virtual SVM deg-9 poly [distortions]	none	0.8	LeCun et al. 1998
Virtual SVM, deg-9 poly, 1-pixel jittered	none	0.68	DeCoste and Scholkopf, MLJ 2002
Virtual SVM, deg-9 poly, 1-pixel jittered	deskewing	0.68	DeCoste and Scholkopf, MLJ 2002
Virtual SVM, deg-9 poly, 2-pixel jittered	deskewing	0.56	DeCoste and Scholkopf, MLJ 2002

Neural Nets

2-layer NN, 300 hidden units, mean square error	none	4.7	LeCun et al. 1998
2-layer NN, 300 HU, MSE, [distortions]	none	3.6	LeCun et al. 1998
2-layer NN, 300 HU	deskewing	1.6	LeCun et al. 1998
2-layer NN, 1000 hidden units	none	4.5	LeCun et al. 1998
2-layer NN, 1000 HU, [distortions]	none	3.8	LeCun et al. 1998
3-layer NN, 300+100 hidden units	none	3.05	LeCun et al. 1998
3-layer NN, 300+100 HU [distortions]	none	2.5	LeCun et al. 1998
3-layer NN, 500+150 hidden units	none	2.95	LeCun et al. 1998
3-layer NN, 500+150 HU [distortions]	none	2.45	LeCun et al. 1998
3-layer NN, 500+300 HU, softmax, cross entropy, weight decay	none	1.53	Hinton, unpublished, 2005
2-layer NN, 800 HU, Cross-Entropy Loss	none	1.6	Simard et al., ICDAR 2003
2-layer NN, 800 HU, cross-entropy [affine distortions]	none	1.1	Simard et al., ICDAR 2003
2-layer NN, 800 HU, MSE [elastic distortions]	none	0.9	Simard et al., ICDAR 2003
2-layer NN, 800 HU, cross-entropy [elastic distortions]	none	0.7	Simard et al., ICDAR 2003
NN, 784-500-500-2000-30 + nearest neighbor, RBM + NCA training [no distortions]	none	1.0	Salakhutdinov and Hinton, AI-Stats 2007
6-layer NN 784-2500-2000-1500-1000-500-10 (on GPU) [elastic distortions]	none	0.35	Ciresan et al. Neural Computation 10, 2010 and arXiv 1003.0358, 2010
committee of 25 NN 784-800-10 [elastic distortions]	width normalization, deslanting	0.39	Meier et al. ICDAR 2011

deep convex net, unsup pre-training [no distortions]	none	0.83	Deng et al. Interspeech 2010
Convolutional nets			
Convolutional net LeNet-1	subsampling to 16x16 pixels	1.7	LeCun et al. 1998
Convolutional net LeNet-4	none	1.1	LeCun et al. 1998
Convolutional net LeNet-4 with K-NN instead of last layer	none	1.1	LeCun et al. 1998
Convolutional net LeNet-4 with local learning instead of last layer	none	1.1	LeCun et al. 1998
Convolutional net LeNet-5, [no distortions]	none	0.95	LeCun et al. 1998
Convolutional net LeNet-5, [huge distortions]	none	0.85	LeCun et al. 1998
Convolutional net LeNet-5, [distortions]	none	0.8	LeCun et al. 1998
Convolutional net Boosted LeNet-4, [distortions]	none	0.7	LeCun et al. 1998
Trainable feature extractor + SVMs [no distortions]	none	0.83	Lauer et al., Pattern Recognition 40-6, 2007
Trainable feature extractor + SVMs [elastic distortions]	none	0.56	Lauer et al., Pattern Recognition 40-6, 2007
Trainable feature extractor + SVMs [affine distortions]	none	0.54	Lauer et al., Pattern Recognition 40-6, 2007
unsupervised sparse features + SVM, [no distortions]	none	0.59	Labusch et al., IEEE TNN 2008
Convolutional net, cross-entropy [affine distortions]	none	0.6	Simard et al., ICDAR 2003
Convolutional net, cross-entropy [elastic distortions]	none	0.4	Simard et al., ICDAR 2003
large conv. net, random features [no distortions]	none	0.89	Ranzato et al., CVPR 2007
large conv. net, unsup features [no distortions]	none	0.62	Ranzato et al., CVPR 2007
large conv. net, unsup pretraining [no distortions]	none	0.60	Ranzato et al., NIPS 2006
large conv. net, unsup pretraining [elastic distortions]	none	0.39	Ranzato et al., NIPS 2006
large conv. net, unsup pretraining [no distortions]	none	0.53	Jarrett et al., ICCV 2009
large/deep conv. net, 1-20-40-60-80-100-120-120-10 [elastic distortions]	none	0.35	Ciresan et al. IJCAI 2011
committee of 7 conv. net, 1-20-P-40-P-150-10 [elastic distortions]	width normalization	0.27 +-0.02	Ciresan et al. ICDAR 2011
committee of 35 conv. net, 1-20-P-40-P-150-10 [elastic distortions]	width normalization	0.23	Ciresan et al. CVPR 2012

References

[LeCun et al., 1998a] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. "Gradient-based learning applied to document recognition." Proceedings of the IEEE, 86(11):2278-2324, November 1998. [on-line version]

FILE FORMATS FOR THE MNIST DATABASE

The data is stored in a very simple file format designed for storing vectors and multidimensional matrices.

General info on this format is given at the end of this page, but you don't need to read that to use the data files.

All the integers in the files are stored in the MSB first (high endian) format used by most non-Intel processors.

Users of Intel processors and other low-endian machines must flip the bytes of the header.

There are 4 files:(**binary files**，二进制文档，注意：没有扩展名)

```
train-images-idx3-ubyte   :  training set images
train-labels-idx1-ubyte   :  training set labels
t10k-images-idx3-ubyte    :  test set images
t10k-labels-idx1-ubyte    :  test set labels
```

The training set contains 60000 examples, and the test set 10000 examples.

The first 5000 examples of the test set are taken from the original NIST training set.

The last 5000 are taken from the original NIST test set.

The first 5000 are cleaner and easier than the last 5000.

TRAINING SET LABEL FILE
(train-labels-idx1-ubyte --> **train-labels.idx1-ubyte**):

```
[offset]  [type]           [value]          [description]
0000       32 bit integer   0x00000801(2049)  magic number (MSB first)
0004       32 bit integer   60000            number of items
0008       unsigned byte    ??                label (0 to 9)
0009       unsigned byte    ??                label (0 to 9)
........
xxxx       unsigned byte    ??                label (0 to 9)
```

The labels values are 0 to 9.

TRAINING SET IMAGE FILE
(train-images-idx3-ubyte --> **train-images.idx3-ubyte**):

```
[offset]  [type]           [value]           [description]
0000       32 bit integer   0x00000803(2051)   magic number
0004       32 bit integer   60000              number of images
0008       32 bit integer   28                number of rows
0012       32 bit integer   28                number of columns
0016       unsigned byte    ??                pixel (0 to 255)
0017       unsigned byte    ??                pixel (0 to 255)
........
xxxx       unsigned byte    ??                pixel (0 to 255)
```

Pixels are organized row-wise. Pixel values are 0 to 255. 0 means background (white), 255 means foreground (black).

TEST SET LABEL FILE

```
(t10k-labels-idx1-ubyte  -->  t10k-labels.idx1-ubyte):

[offset]  [type]           [value]           [description]
0000      32 bit integer   0x00000801(2049)  magic number (MSB first)
0004      32 bit integer   10000             number of items
0008      unsigned byte    ??                 label
0009      unsigned byte    ??                 label
........
xxxx      unsigned byte    ??                 label
```

The labels values are 0 to 9.

TEST SET IMAGE FILE

```
(t10k-images-idx3-ubyte  -->  t10k-images.idx3-ubyte):
[offset]  [type]           [value]            [description]
0000      32 bit integer   0x00000803(2051)   magic number
0004      32 bit integer   60000              number of images
0008      32 bit integer   28                number of rows
0012      32 bit integer   28                number of columns
0016      unsigned byte    ??                pixel (0 to 255)
0017      unsigned byte    ??                pixel (0 to 255)
........
xxxx      unsigned byte    ??                pixel (0 to 255)

[offset]  [type]           [value]            [description]
0000      32 bit integer   0x00000803(2051)   magic number
0004      32 bit integer   10000              number of images
0008      32 bit integer   28                number of rows
0012      32 bit integer   28                number of columns
0016      unsigned byte    ??                pixel
0017      unsigned byte    ??                pixel
........
xxxx      unsigned byte    ??                pixel
```

Pixels are organized row-wise. Pixel values are 0 to 255. 0 means background (white), 255 means foreground (black).

THE IDX FILE FORMAT

the IDX file format is a simple format for vectors and multidimensional matrices of various numerical types.
The basic format is

magic number
size in dimension 0
size in dimension 1

size in dimension 2
.....
size in dimension N
data

The **magic number** is an integer (MSB first). The first 2 bytes are always 0.

The third byte codes the type of the data:
0x08: unsigned byte
0x09: signed byte
0x0B: short (2 bytes)
0x0C: int (4 bytes)
0x0D: float (4 bytes)
0x0E: double (8 bytes)

The 4-th byte codes the number of dimensions of the vector/matrix: 1 for vectors, 2 for matrices....

The sizes in each dimension are 4-byte integers (MSB first, high endian, like in most non-Intel processors).

The data is stored like in a C array, i.e. the index in the last dimension changes the fastest.

Happy hacking.

The digit images in the MNIST set were originally selected and experimented with by Chris Burges and Corinna Cortes using bounding-box normalization and centering. Yann LeCun's version which is provided on this page uses centering by center of mass within in a larger window.

Yann LeCun, Professor
The Courant Institute of Mathematical Sciences
New York University

Corinna Cortes, Research Scientist
Google Labs, New York
corinna at google dot com

C code for the MNIST database

```c
main()
{
    // mnist train iamges 60000, // 784 = (28 * 28)
    int size_mnist_train_images = sizeof(unsigned char) * 784 * 60000;

    unsigned char * mnist_train_images =
                    (unsigned char *) malloc(size_mnist_train_images);

    // mnist test images 10000, // 784 = (28 * 28)
    int size_mnist_test_images = sizeof(unsigned char) * 784 * 10000;

    unsigned char * mnist_test_images =
                    (unsigned char *) malloc(size_mnist_test_images);

    // mnist train labels 60000
    int size_mnist_train_labels = sizeof(int) * 1 * 60000;

    int * mnist_train_labels = (int *) malloc(size_mnist_train_labels);

    // mnist test labels 60000
    int size_mnist_test_labels = sizeof(int) * 1 * 10000;

    int * mnist_test_labels = (int *) malloc(size_mnist_test_labels);

    // read data to data set, call func
    read_mnist_data(
                    mnist_train_images,
                    mnist_test_images,
                    mnist_train_labels,
                    mnist_test_labels);

    // free
    free(mnist_train_images);
    free(mnist_test_images);
    free(mnist_train_labels);
    free(mnist_test_labels);
}
```

```
/* 2018-08-24 */
void read_mnist_data(
                    unsigned char * mnist_train_images,
                    unsigned char * mnist_test_images,
                    int * mnist_train_labels,
                    int * mnist_test_labels)
{
   ////////////////////////////////////////////////////////////////////////
   // open input file
   ////////////////////////////////////////////////////////////////////////

   // "rb" : read only for binary file

   FILE * fptr_train_images =
              fopen("D:\\VideoData2016\\train-images.idx3-ubyte", "rb");

   FILE * fptr_train_labels =
              fopen("D:\\VideoData2016\\train-labels.idx1-ubyte", "rb");

   FILE * fptr_test_images  =
              fopen("D:\\VideoData2016\\t10k-images.idx3-ubyte",  "rb");

   FILE * fptr_test_labels  =
              fopen("D:\\VideoData2016\\t10k-labels.idx1-ubyte",  "rb");


   ////////////////////////////////////////////////////////////////////////
   // open output file
   ////////////////////////////////////////////////////////////////////////

   // "w+" : clean before write for text file

   FILE * fptr_writeout_train_data =
              fopen("D:\\VideoData2016\\mnist_train_data.show", "w+");

   FILE * fptr_writeout_test_data  =
              fopen("D:\\VideoData2016\\mnist_test_data.show",  "w+");


   ////////////////////////////////////////////////////////////////////////
   // (1) read data from fptr_train_images
   ////////////////////////////////////////////////////////////////////////

   unsigned char ch;
   int char_cursor;
```

```
char_cursor = 0;

while(!feof(fptr_train_images) && char_cursor < 16 )
//use feof : in <stdio.h>
{
    ch = (unsigned char) fgetc(fptr_train_images);
    //use fgetc : in <stdio.h>

    char_cursor = char_cursor + 1;

    printf("%5d",ch);
}

int char_num;
int char_total;

char_num = 0;
char_total = 60000 * 28 * 28;

while(!feof(fptr_train_images) && char_num < char_total)
//use feof : in <stdio.h>
{
    ch = (unsigned char) fgetc(fptr_train_images);
    //use fgetc : in <stdio.h>

    mnist_train_images[char_num] = ch;

    //printf("%4d",ch);

    char_num = char_num + 1;
}


//////////////////////////////////////////////////////////////////////
// (2) read data from fptr_test_images
//////////////////////////////////////////////////////////////////////

//unsigned char ch;

char_cursor = 0;

while(!feof(fptr_test_images) && char_cursor < 16 )
//use feof : in <stdio.h>
{
    ch = (unsigned char) fgetc(fptr_test_images);
    //use fgetc : in <stdio.h>
    char_cursor = char_cursor + 1;

    printf("%5d",ch);
}

char_num = 0;
char_total = 10000 * 28 * 28;

while(!feof(fptr_test_images) && char_num < char_total)
//use feof : in <stdio.h>
{
    ch = (unsigned char) fgetc(fptr_test_images);
     //use fgetc : in <stdio.h>

    mnist_test_images[char_num] = ch;

    //printf("%4d",ch);

    char_num = char_num + 1;
}


//////////////////////////////////////////////////////////////////////
// (3) read data from fptr_train_labels
//////////////////////////////////////////////////////////////////////

//unsigned char ch;

char_cursor = 0;

while(!feof(fptr_train_labels) && char_cursor < 8 )
//use feof : in <stdio.h>
{
    ch = (unsigned char) fgetc(fptr_train_labels);
     //use fgetc : in <stdio.h>

    char_cursor = char_cursor + 1;

    printf("%5d",ch);
}

char_num = 0;
char_total = 60000 * 1;

while(!feof(fptr_train_labels) && char_num < char_total)
```

```
//use feof : in <stdio.h>
{
    ch = (unsigned char) fgetc(fptr_train_labels);
     //use fgetc : in <stdio.h>

    mnist_train_labels[char_num] = (int) ch;

    //printf("%4d",ch);

    char_num = char_num + 1;
}


////////////////////////////////////////////////////////////////////////
// (4) read data from fptr_test_labels
////////////////////////////////////////////////////////////////////////

//unsigned char ch;

char_cursor = 0;

while(!feof(fptr_test_labels) && char_cursor < 8 )
//use feof : in <stdio.h>
{
    ch = (unsigned char) fgetc(fptr_test_labels);
     //use fgetc : in <stdio.h>

    char_cursor = char_cursor + 1;

    printf("%5d",ch);
}

char_num = 0;
char_total = 10000 * 1;

while(!feof(fptr_test_labels) && char_num < char_total)
//use feof : in <stdio.h>
{
    ch = (unsigned char) fgetc(fptr_test_labels);
     //use fgetc : in <stdio.h>

    mnist_test_labels[char_num] = (int) ch;

    //printf("%4d",ch);

    char_num = char_num + 1;
```

```
}


//////////////////////////////////////////////////////////////////////////
// write train data
//////////////////////////////////////////////////////////////////////////

for (int image = 0; image < 60000; image ++)
{
    printf("%10d",image);

    fprintf(fptr_writeout_train_data, "%4d", mnist_train_labels[image]);

    fprintf(fptr_writeout_train_data, "\n");

    for (int i = 0; i < 28; i++)
    {
        for (int j = 0; j < 28; j++)
        {
            // 784 = 28 * 28
            fprintf(
                    fptr_writeout_train_data,
                    "%4d",
                    mnist_train_images[image * 784 + i * 28 + j]);
        } // end for j

        fprintf(fptr_writeout_train_data, "\n");

    }// end for i

}// end for image


//////////////////////////////////////////////////////////////////////////
// write test data
//////////////////////////////////////////////////////////////////////////

for (int image = 0; image < 10000; image ++)
{
    printf("%10d",image);

    fprintf(fptr_writeout_test_data, "%4d", mnist_test_labels[image]);

    fprintf(fptr_writeout_test_data, "\n");

    for (int i = 0; i < 28; i++)
```

```
        {
            for (int j = 0; j < 28; j++)
            {
                // 784 = 28 * 28
                fprintf(
                        fptr_writeout_test_data,
                        "%4d",
                        mnist_test_images[image * 784 + i * 28 + j]);
            } // end for j

            fprintf(fptr_writeout_test_data, "\n");

        }// end for i

    }// end for image


    ////////////////////////////////////////////////////////////////////////
    // close file
    ////////////////////////////////////////////////////////////////////////

    fclose(fptr_train_images);
    fclose(fptr_train_labels);
    fclose(fptr_test_images );
    fclose(fptr_test_labels );

    fclose(fptr_writeout_train_data);
    fclose(fptr_writeout_test_data );

}
```

mnist_train_data.show 前 29 行数据：

```
   5
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   3  18  18  18 126 136 175  26 166 255 247 127   0   0   0   0
   0   0   0   0   0   0   0   0  30  36  94 154 170 253 253 253 253 253 225 172 253 242 195  64   0   0   0   0
   0   0   0   0   0   0   0  49 238 253 253 253 253 253 253 253 253 251  93  82  82  56  39   0   0   0   0   0
   0   0   0   0   0   0   0  18 219 253 253 253 253 253 198 182 247 241   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0  80 156 107 253 253 205  11   0  43 154   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0  14   1 154 253  90   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0 139 253 190   2   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0  11 190 253  70   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0  35 241 225 160 108   1   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0  81 240 253 253 119  25   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0  45 186 253 253 150  27   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0  16  93 252 253 187   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 249 253 249  64   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0  46 130 183 253 253 207   2   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0  39 148 229 253 253 253 250 182   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0  24 114 221 253 253 253 253 201  78   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0  23  66 213 253 253 253 253 198  81   2   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0  18 171 219 253 253 253 253 195  80   9   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0  55 172 226 253 253 253 253 244 133  11   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0 136 253 253 253 212 135 132  16   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
```

mnist_test_data.show 前 29 行数据：

```
   7
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0  84 185 159 151  60  36   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0 222 254 254 254 254 241 198 198 198 198 198 198 198 198 170  52   0   0   0   0   0   0
   0   0   0   0   0   0  67 114  72 114 163 227 254 225 254 254 254 250 229 254 254 140   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0  17  66  14  67  67  67  59  21 236 254 106   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0  83 253 209  18   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0  22 233 255  83   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 129 254 238  44   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0  59 249 254  62   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 133 254 187   5   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   9 205 248  58   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 126 254 182   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0  75 251 240  57   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0  19 221 254 166   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   3 203 254 219  35   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0  38 254 254  77   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0  31 224 254 115   1   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0 133 254 254  52   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0  61 242 254 254  52   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0 121 254 254 219  40   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0 121 254 207  18   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
```