```
# pip install fastbook
```

```
from fastai.vision.all import *
from fastbook import *
matplotlib.rc('image', cmap='Greys')
```

DataSet MNIST

contains data type -> handwritten images of number .png

```
path = untar_data(URLs.MNIST_SAMPLE)
```

100.14% [3219456/3214948 00:00<00:00]

MNIST dataset follows a common layout for machine learning datasets: separate folders for the training set and the validation set (and/or test set)

```
Path.BASE_PATH=path
path.ls()
```

```
(#3) [Path('valid'),Path('train'),Path('labels.csv')]
```

```
#inside the training set
print((path/'train').ls())
```

```
[Path('train/7'), Path('train/3')]
```

Notes

- 3s and 7s folders - in ML we say
- '3' and '7' are labels/ targets

```
#in folders
three=(path/'train'/'3').ls().sorted() #return lists of asked dataset [.png]
seven=(path/'train'/'7').ls().sorted()
print("Length of 3s, 7s : ",len(three),len(seven))
```

```
Length of 3s, 7s :  6131 6265
```

HOW DATA LOOKS

Notes

- Image class from PIL Python Image Library used

```
image3_path=three[1]
image3=Image.open(image3_path)
image3
```

Represent in an array **Notes**

- Array (numpy) works on cpu
- tensor works on gpu
- So preferably use tensor for deep learning tasks

```
tensor(image3)[4:10,4:10]
```

```
tensor([[  0,   0,   0,   0,   0,   0],
        [  0,   0,   0,   0,   0,  29],
        [  0,   0,   0,  48, 166, 224],
        [  0,  93, 244, 249, 253, 187],
        [  0, 107, 253, 253, 230,  48],
        [  0,   3,  20,  20,  15,   0]], dtype=torch.uint8)
```

Notes

- *The 4:10 indicates we requested the rows from index 4 (included) to 10 (not included) and the same for the columns. NumPy indexes from top to bottom and left to right, so this section is located in the top-left corner of the image.*
- *We can slice the array to pick just the part with the top of the digit in it, and then use a Pandas DataFrame to color-code the values using a gradient, which shows us clearly how the image is created from the pixel values:*

```
image3_tensor=tensor(image3)
df=pd.DataFrame(image3_tensor[4:50,4:22])
df
```

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
1	0	0	0	0	0	29	150	195	254	255	254	176	193	150	96	0	0	
2	0	0	0	48	166	224	253	253	234	196	253	253	253	253	233	0	0	
3	0	93	244	249	253	187	46	10	8	4	10	194	253	253	233	0	0	
4	0	107	253	253	230	48	0	0	0	0	0	192	253	253	156	0	0	
5	0	3	20	20	15	0	0	0	0	0	43	224	253	245	74	0	0	
6	0	0	0	0	0	0	0	0	0	0	249	253	245	126	0	0	0	
7	0	0	0	0	0	0	0	14	101	223	253	248	124	0	0	0	0	
8	0	0	0	0	0	11	166	239	253	253	253	187	30	0	0	0	0	
9	0	0	0	0	0	16	248	250	253	253	253	253	232	213	111	2	0	
10	0	0	0	0	0	0	0	43	98	98	208	253	253	253	253	187	22	
11	0	0	0	0	0	0	0	0	0	0	9	51	119	253	253	253	76	
12	0	0	0	0	0	0	0	0	0	0	0	0	1	183	253	253	139	
13	0	0	0	0	0	0	0	0	0	0	0	0	0	182	253	253	104	
14	0	0	0	0	0	0	0	0	0	0	0	0	85	249	253	253	36	
15	0	0	0	0	0	0	0	0	0	0	0	60	214	253	253	173	11	
16	0	0	0	0	0	0	0	0	0	0	98	247	253	253	226	9	0	
17	0	0	0	0	0	0	0	0	42	150	252	253	253	233	53	0	0	
18	0	0	42	115	42	60	115	159	240	253	253	250	175	25	0	0	0	
19	0	0	187	253	253	253	253	253	253	253	197	86	0	0	0	0	0	
20	0	0	103	253	253	253	253	253	232	67	1	0	0	0	0	0	0	

Color Codes for Better Understanding

22	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

```
df.style.set_properties(**{'font-size':'6pt'}).background_gradient('Greys')
```

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	29	150	195	254	255	254	176	193	150	96	0	0	0
2	0	0	0	48	166	224	253	253	234	196	253	253	253	253	233	0	0	0
3	0	93	244	249	253	187	46	10	8	4	10	194	253	253	233	0	0	0
4	0	107	253	253	230	48	0	0	0	0	0	192	253	253	156	0	0	0
5	0	3	20	20	15	0	0	0	0	0	43	224	253	245	74	0	0	0
6	0	0	0	0	0	0	0	0	0	0	249	253	245	126	0	0	0	0
7	0	0	0	0	0	0	0	14	101	223	253	248	124	0	0	0	0	0
8	0	0	0	0	0	11	166	239	253	253	253	187	30	0	0	0	0	0
9	0	0	0	0	0	16	248	250	253	253	253	253	232	213	111	2	0	0
10	0	0	0	0	0	0	0	43	98	98	208	253	253	253	253	187	22	0
11	0	0	0	0	0	0	0	0	0	0	9	51	119	253	253	253	76	0
12	0	0	0	0	0	0	0	0	0	0	0	0	1	183	253	253	139	0
13	0	0	0	0	0	0	0	0	0	0	0	0	0	182	253	253	104	0
14	0	0	0	0	0	0	0	0	0	0	0	0	85	249	253	253	36	0
15	0	0	0	0	0	0	0	0	0	0	0	60	214	253	253	173	11	0
16	0	0	0	0	0	0	0	0	0	0	98	247	253	253	226	9	0	0
17	0	0	0	0	0	0	0	0	42	150	252	253	253	233	53	0	0	0

SEVEN

```
image7_path=seven[0]
image7=Image.open(image7_path)
image7_tensor=tensor(image7)
df=pd.DataFrame(image7_tensor[4:50,4:23])
df.style.set_properties(**{'font-size':'6pt'}).background_gradient('Greys')
```

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	21	51	213	254	252	252	252	254	252	252	252	254	252	252	252	255	252	100	0
4	161	250	250	252	250	250	250	252	250	250	250	252	250	250	250	252	250	100	0
5	250	250	250	252	189	190	250	252	250	250	250	252	250	250	250	252	189	40	0
6	130	250	250	49	29	30	49	49	49	49	49	49	49	170	250	252	149	0	0
7	0	0	0	0	0	0	0	0	0	0	0	11	132	252	252	244	121	0	0
8	0	0	0	0	0	0	0	0	0	0	0	51	250	250	250	202	0	0	0
9	0	0	0	0	0	0	0	0	0	0	0	172	250	250	250	80	0	0	0
10	0	0	0	0	0	0	0	0	0	0	0	252	250	250	250	0	0	0	0
11	0	0	0	0	0	0	0	0	0	31	213	254	252	252	49	0	0	0	0
12	0	0	0	0	0	0	0	0	0	151	250	252	250	250	49	0	0	0	0
13	0	0	0	0	0	0	0	0	0	151	250	252	250	159	20	0	0	0	0
14	0	0	0	0	0	0	0	0	0	151	250	252	250	100	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	152	252	254	252	100	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0	151	250	252	250	100	0	0	0	0	0
17	0	0	0	0	0	0	0	0	0	151	250	252	250	100	0	0	0	0	0
18	0	0	0	0	0	0	0	0	0	151	250	252	250	221	40	0	0	0	0
19	0	0	0	0	0	0	0	0	0	153	252	255	252	252	49	0	0	0	0

First Method: Pixel Similarity

find the average pixel value for every pixel of the 3s, then do the same for the 7s. This will give us two group averages, defining what we might call the "ideal" 3 and 7. Then, to classify an image as one digit or the other, we see which of these two ideal digits the image is most similar to

Get Avg of pixel vales for each of our two groups

Notes

- tensor containing all of our 3s stacked together

```
three_tensors=[tensor(Image.open(img)) for img in three]
```

```
seven_tensors=[tensor(Image.open(img)) for img in seven]
len(three_tensors),len(seven_tensors)
```

```
(6131, 6265)
```

Python list comprehension to create a plain list of the single image tensors.

Since we now have tensors use fastai's show_image function to display it:

```
show_image(three_tensors[1])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f431a86f350>
```

combine all the images in this list into a single three-dimensional tensor.he most common way to describe such a tensor is to call it a rank-3 tensor. We often need to stack up individual tensors in a collection into a single tensor. Unsurprisingly, PyTorch comes with a function called stack that we can use for this purpose.

Generally when images are floats, the pixel values are expected to be between 0 and 1, so we will also divide by 255

Double-click (or enter) to edit

```
stacked_three=torch.stack(three_tensors).float()/255
stacked_seven=torch.stack(seven_tensors).float()/255
stacked_three.shape
```

```
torch.Size([6131, 28, 28])
```

Perhaps the most important attribute of a tensor is its shape. This tells you the length of each axis. In this case, we can see that we have 6,131 images, each of size 28×28 pixels. first axis is the number of images, the second is the height, and the third is the width

The length of a tensor's shape is its rank:

```
len(stacked_three.shape)
```

```
3
```

we can compute what the ideal 3 looks like. We calculate the mean of all the image tensors by taking the mean along dimension 0 of our stacked, rank-3 tensor. This is the dimension that indexes over all the images.

In other words, for every pixel position, this will compute the average of that pixel over all images. The result will be one value for every pixel position, or a single image. Here it is:

Ideal Three

```
mean3 = stacked_three.mean(0)
show_image(mean3);
```

Ideal Seven

```
mean7=stacked_seven.mean(0)
show_image(mean7)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f431a1b0990>
```

PICKING TEST IMAGE

- from validation set

```
path.ls()
```

```
(#3) [Path('valid'),Path('train'),Path('labels.csv')]
```

```
(path/'valid').ls()
```

```
(#2) [Path('valid/7'),Path('valid/3')]
```

```
(path/'valid'/'3').ls()
```

```
(#1010)
```

```
[Path('valid/3/3834.png'),Path('valid/3/5248.png'),Path('valid/3/8115.png'),Path
```

NOT FROM SAME DATA SET

```
# test_image=stacked_three[1]
# show_image(test_image)
```

FROM VALIDATION SET

```
testingPath=(path/'valid'/'3').ls().sorted()
testing_tensors_three=[tensor(Image.open(img)) for img in testingPath]
stacked_test_three=torch.stack(testing_tensors_three).float()/255
test_image=stacked_test_three[1]
show_image(test_image)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f431a074d50>
```

Double-click (or enter) to edit

Calculating Erros

Notes

- mean absolute difference Take the mean of the absolute value of differences
- Take the mean of the square of differences RMSE

FOR 3

```
dist_three_abs=(test_image - mean3).abs().mean()
dist_three_sqrt=((test_image-mean3)**2).mean().sqrt()
dist_three_abs,dist_three_sqrt
```

```
(tensor(0.1623), tensor(0.2883))
```

FOR 7

```
dist_seven_abs=(test_image - mean7).abs().mean()
dist_seven_sqrt=((test_image - mean7)**2).mean().sqrt()
dist_seven_abs,dist_seven_sqrt
```

```
(tensor(0.1958), tensor(0.3523))
```

- **2nd Method Using PyTorch Funtions**

```
mad7=F.l1_loss(test_image.float(),mean7)
mse7=F.mse_loss(test_image,mean7).sqrt()
mad7,mse7
```

```
(tensor(0.1958), tensor(0.3523))
```

TESTING

```
print('YOUR IMAGE : ')
show_image(test_image)
```

```
YOUR IMAGE :
<matplotlib.axes._subplots.AxesSubplot at 0x7f431a048fd0>
```

```
if(dist_three_abs<mad7 and dist_three_sqrt<mse7) :
   print("Image Classified As 3")
else:
  print("Image Classified As 7")
```

```
Image Classified As 3
```

Designing Metrixs Using Broadcasting

```
pip install torch
```

```
valid_three_tensors=torch.stack([tensor(Image.open(img)) for img in (path/'valid'/'3'
valid_seven_tensors=torch.stack([tensor(Image.open(img))for img in (path/'valid'/'7').
```

the pixel values are expected to be between 0 and 1, so we will also divide by 255

```
valid_three_tensors=valid_three_tensors.float()/255
valid_seven_tensors=valid_seven_tensors.float()/255
valid_seven_tensors.shape,valid_three_tensors.shape
```

```
(torch.Size([1028, 28, 28]), torch.Size([1010, 28, 28]))
```

Funtion - Returns Mean Abs Error

```
def mnist_distance(a,b): return (a-b).abs().mean((-1,-2))
mnist_distance(test_image, mean3)
```

```
tensor(0.1623)
```

```
valid_three_dist = mnist_distance(valid_three_tensors, mean3)
valid_three_dist, valid_three_dist.shape
```

```
(tensor([0.1210, 0.1287, 0.1781,  ..., 0.1324, 0.1287, 0.1423]),
 torch.Size([1010]))
```

returned distance for every image in a vector = size 1010

when it tries to perform a simple subtraction operation between two tensors of different ranks, will use broadcasting. That is, it will automatically expand the tensor with the smaller rank to have the same size as the one with the larger rank. Broadcasting is an important capability that makes tensor code much easier to write.

```
(valid_three_tensors-mean3).shape
```

```
torch.Size([1010, 28, 28])
```

Testing

```
def is_3(x): return mnist_distance(x,mean3) < mnist_distance(x,mean7)
```

```
is_3(valid_three_tensors),is_3(valid_three_tensors).float()
```

```
(tensor([True, True, True,  ..., True, True, True]),
 tensor([1., 1., 1.,  ..., 1., 1., 1.]))
```

Accuracy

```
accuracy_three= is_3(valid_three_tensors).float().mean()
accuracy_seven= (1- is_3(valid_seven_tensors).float()).mean()

accuracy_three,accuracy_seven,(accuracy_three+accuracy_seven)/2
```

```
(tensor(0.9168), tensor(0.9854), tensor(0.9511))
```