

# Pixel Similarity & Co

How to categorize images without Neural Networks

# 1

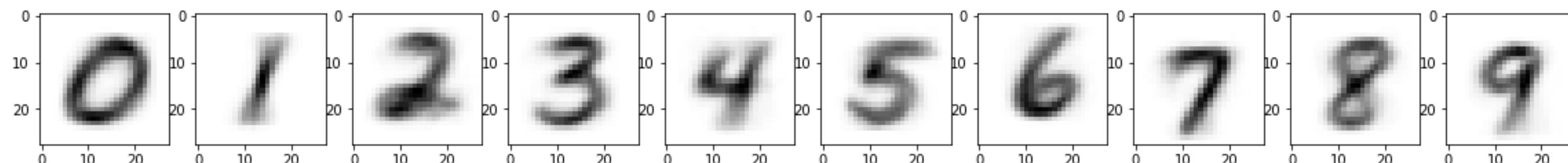
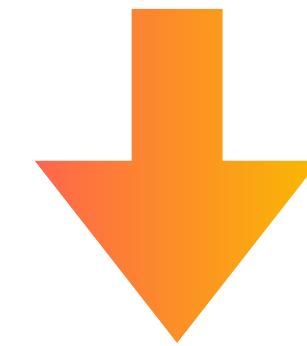
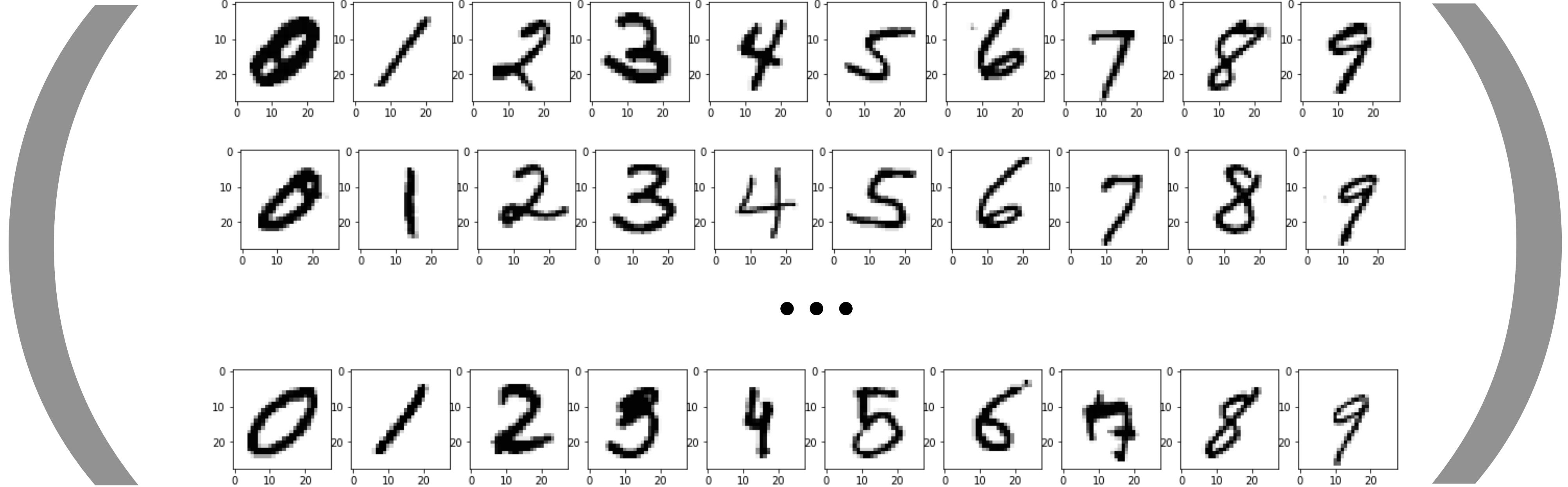
## Pixel Similarity & Co

Overview of this basic approach:

- What is the reasoning and steps?
- Describe what the distance means

# "Ideal Images" as Target

avg



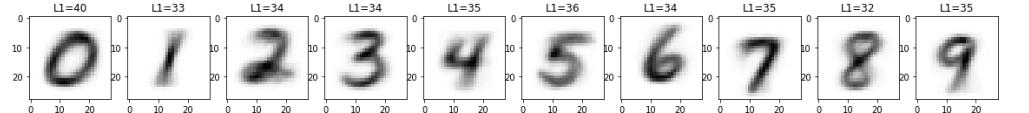
# "Ideal Images" as Target

avg

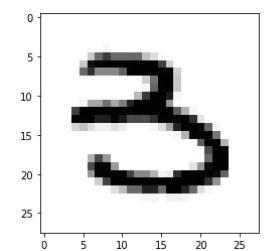
"ideal" 2 astensor

# "L1 Norm" Algo

ideal\_digits



img

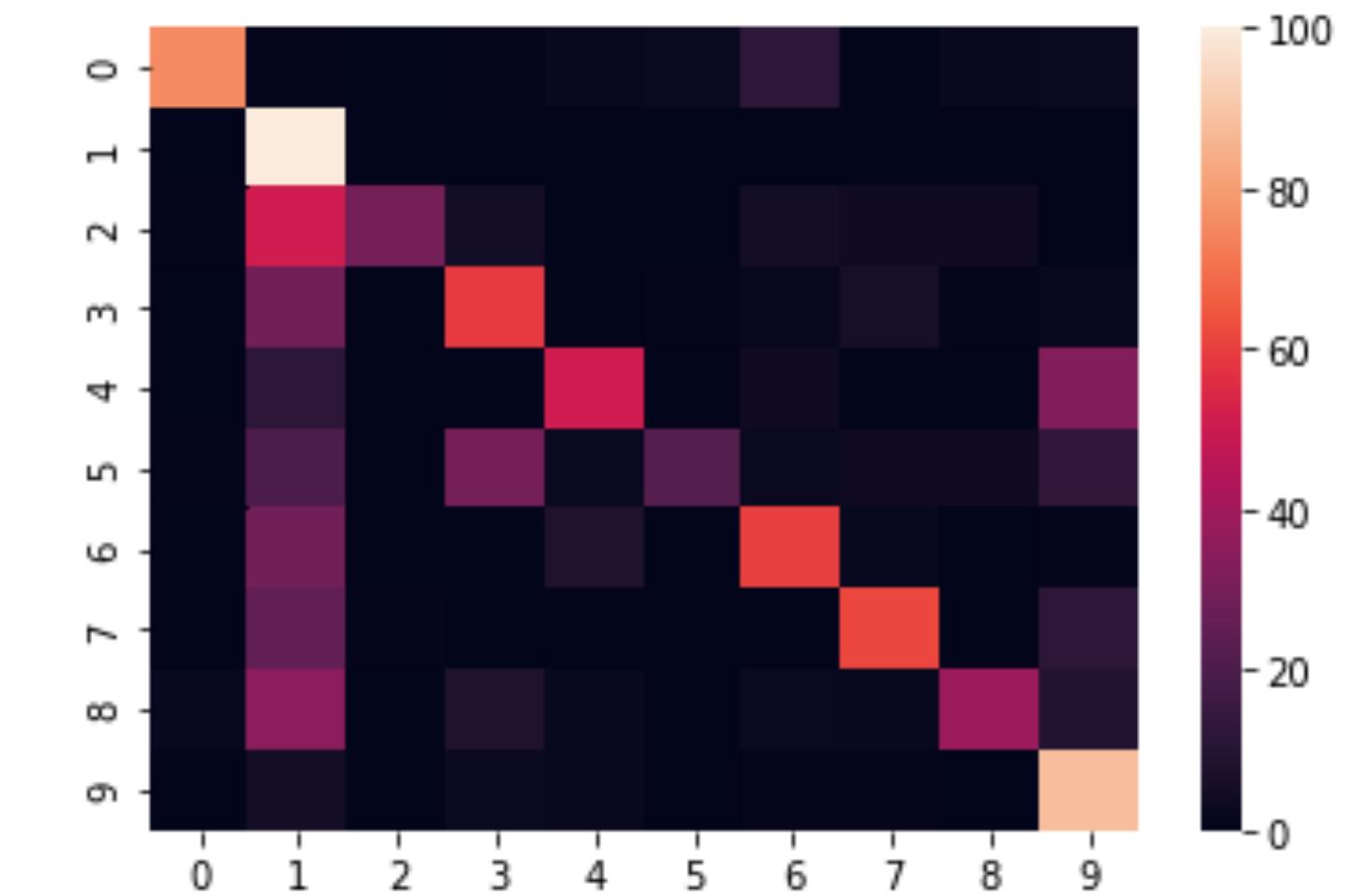


```
# Example: L1 for img
l1=[(img - digit).abs().mean() for digit in ideal_digits]
l1
```

```
[tensor(44.2404),
 tensor(44.0219),
 tensor(46.1260),
 tensor(41.8001),
 tensor(46.5060),
 tensor(42.7080),
 tensor(44.3475),
 tensor(45.5552),
 tensor(45.4974),
 tensor(45.0498)]
```

```
# index of smallest element is digit number
np.argmin(l1)
```

3



Classification report for classifier:				
	precision	recall	f1-score	support
0	0.95	0.76	0.84	100
1	0.33	1.00	0.49	100
2	0.97	0.30	0.46	100
3	0.56	0.59	0.57	100
4	0.75	0.51	0.61	100
5	0.85	0.22	0.35	100
6	0.67	0.60	0.63	100
7	0.77	0.62	0.69	100
8	0.78	0.39	0.52	100
9	0.55	0.88	0.67	100
accuracy			0.59	1000
macro avg	0.72	0.59	0.58	1000
weighted avg	0.72	0.59	0.58	1000

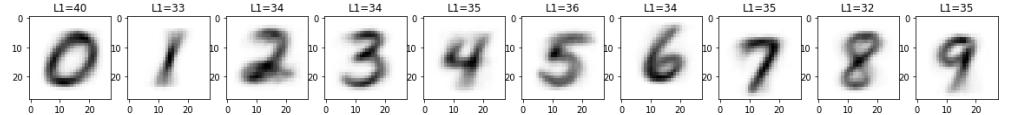
```
confusion_matrix(true, pred)
```

```
array([[ 76,    1,    0,    1,    2,    3,   12,    0,    2,    3],
       [  0, 100,    0,    0,    0,    0,    0,    0,    0,    0],
       [  1,   51,   30,    5,    0,    0,    5,    4,    4,    0],
       [  0,   29,    0,   59,    0,    1,    2,    6,    1,    2],
       [  0,   12,    0,    0,   51,    0,    4,    0,    0,   33],
       [  1,   20,    0,   30,    3,   22,    3,    4,    4,   13],
       [  0,   29,    0,    0,    8,    0,   60,    2,    0,    1],
       [  0,   25,    1,    0,    0,    0,    0,   62,    0,   12],
       [  2,   35,    0,    8,    2,    0,    3,    2,   39,    9],
       [  0,    5,    0,    3,    2,    0,    1,    1,    0,   88]])
```

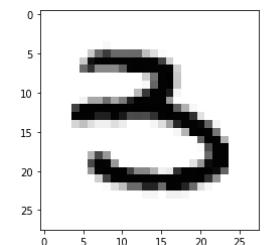
💡 The L1-Norm Algorithm brings solid results with an overall accuracy-value of 59%.

# "L2 Norm" Algo

ideal\_digits



img

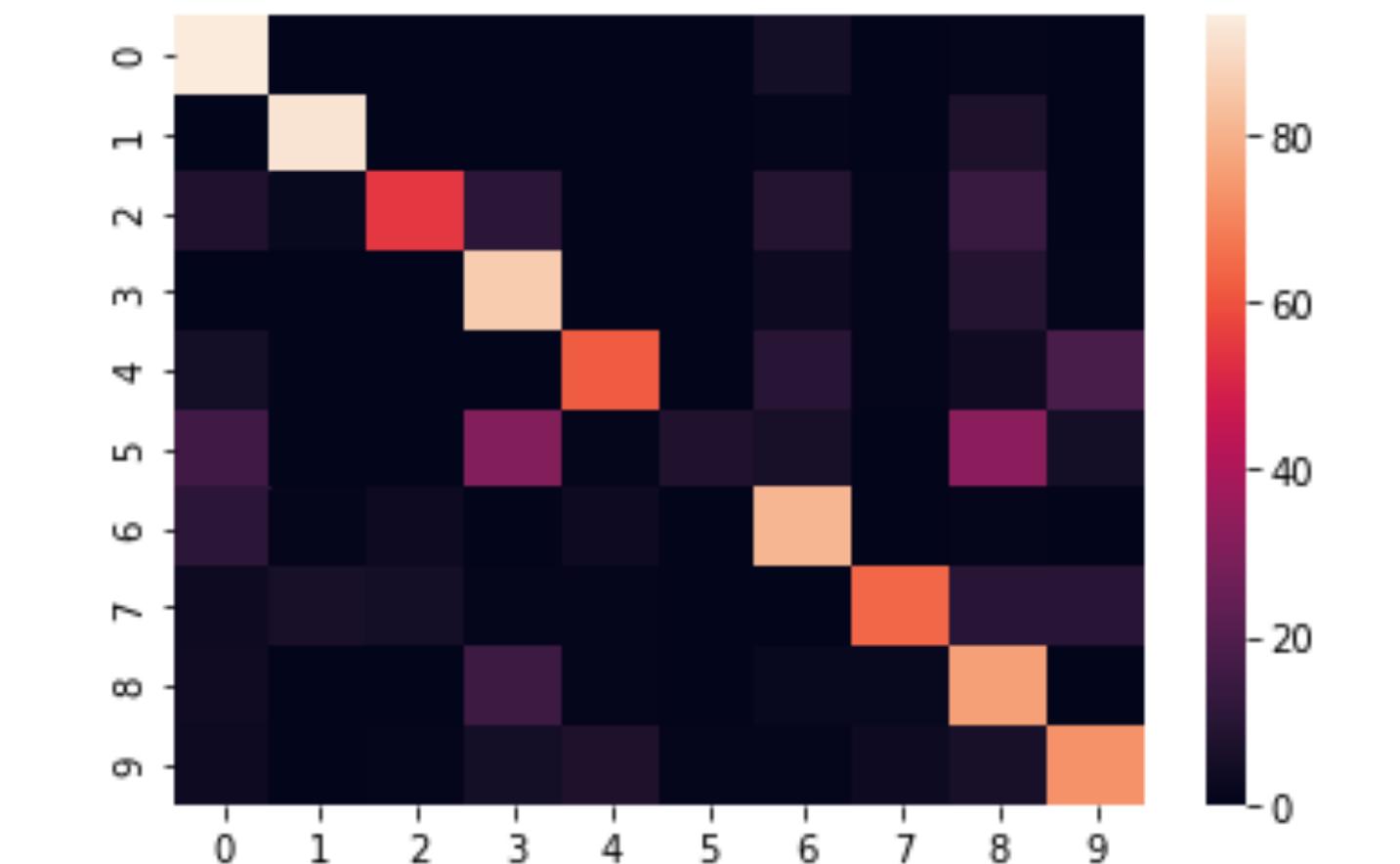


```
# Example: L2 for img
l2=[((img - digit)**2).mean().sqrt() for digit in ideal_digits]
l2
```

```
[tensor(83.5524),
 tensor(91.7463),
 tensor(88.1979),
 tensor(83.2467),
 tensor(90.5193),
 tensor(85.2805),
 tensor(86.3737),
 tensor(90.3234),
 tensor(87.1980),
 tensor(89.5686)]
```

```
# index of smallest element is digit number
np.argmin(l2)
```

3



Classification report for classifier:

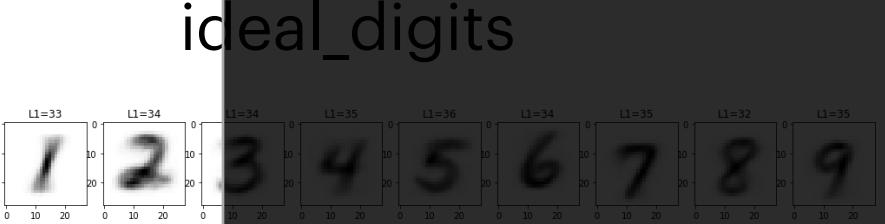
	precision	recall	f1-score	support
0	0.65	0.94	0.77	100
1	0.91	0.92	0.92	100
2	0.86	0.55	0.67	100
3	0.58	0.86	0.69	100
4	0.83	0.62	0.71	100
5	0.89	0.08	0.15	100
6	0.69	0.81	0.74	100
7	0.89	0.64	0.74	100
8	0.47	0.76	0.58	100
9	0.68	0.73	0.71	100
accuracy			0.69	1000
macro avg	0.74	0.69	0.67	1000
weighted avg	0.74	0.69	0.67	1000

```
confusion_matrix(true, pred)
```

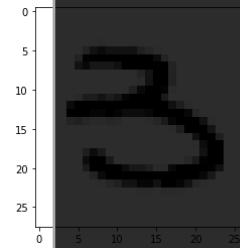
```
array([[94,  0,  0,  0,  0,  0,  5,  0,  1,  0],
       [ 0, 92,  0,  0,  0,  0,  1,  0,  7,  0],
       [ 8,  2, 55, 11,  0,  0,  9,  1, 14,  0],
       [ 0,  0,  0, 86,  0,  0,  3,  1,  9,  1],
       [ 5,  0,  0,  0, 62,  0, 10,  1,  4, 18],
       [16,  0,  0, 31,  1,  8,  6,  0, 33,  5],
       [11,  1,  3,  0,  3,  0, 81,  0,  1,  0],
       [ 3,  6,  5,  1,  1,  0,  0, 64, 10, 10],
       [ 4,  0,  0, 15,  1,  0,  2,  2, 76,  0],
       [ 3,  0,  1,  5,  7,  1,  1,  3,  6, 73]])
```

💡 The L2-Norm Algorithm brings even better results with an overall accuracy-value of 69%.

# "L2 Norm" Algo



img

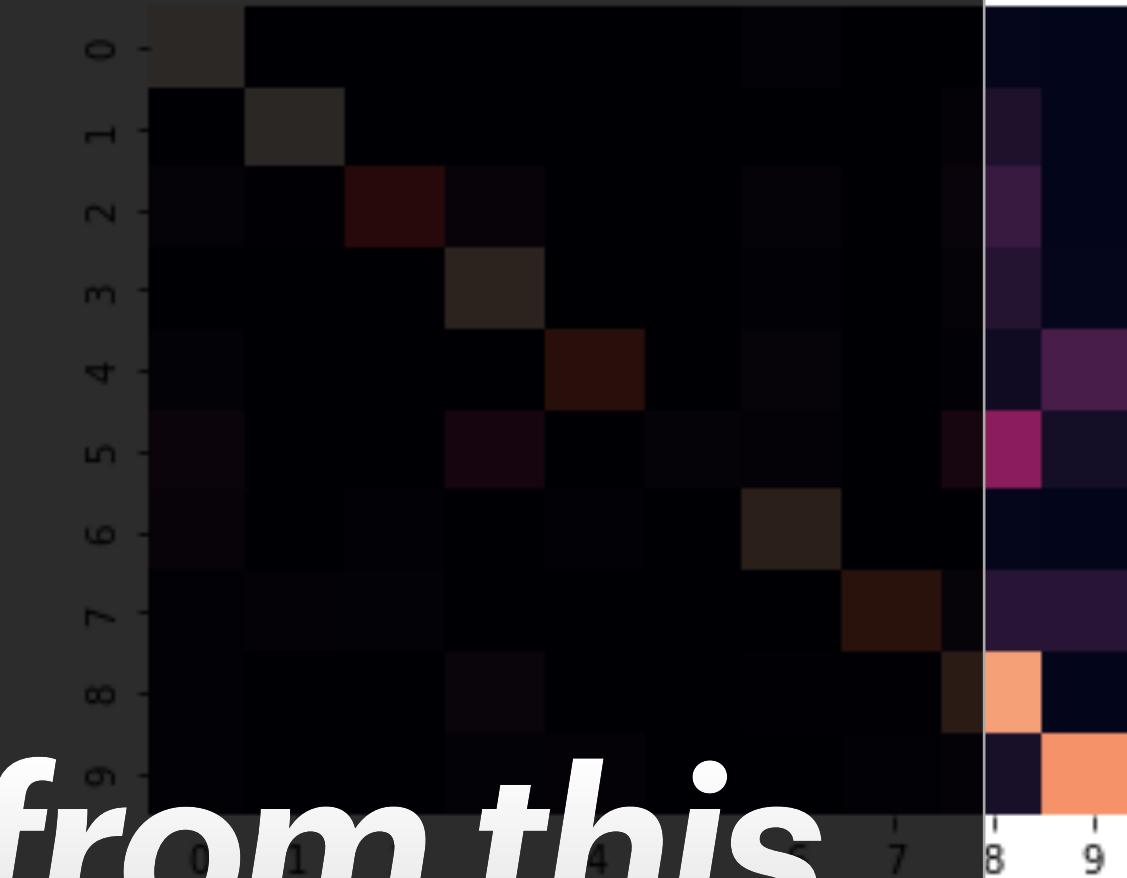


```
# Example: L2 for img
l2=[((img - digit)**2).mean().sqrt() for digit in ideal_digits]
l2

[tensor(83.5524),
 tensor(91.7463),
 tensor(88.1979),
 tensor(83.2467),
 tensor(90.5193),
 tensor(85.2805),
 tensor(86.3737),
 tensor(90.3234),
 tensor(87.1980),
 tensor(89.5686])

# index of smallest element is digit number
np.argmin(l2)
```

2



-80  
-60  
-40  
-20  
0

**"Remember to step away from this book and jot down some ideas before you move on! "**

0	0.65	0.94	0.77	100
1	0.91	0.92	0.92	100
2	0.86	0.5	0.76	100
3	0.58	0.72	0.59	100
4	0.83	0.62	0.71	100
5	0.89	0.08	0.15	100
6	0.69	0.81	0.74	100
7	0.89	0.64	0.74	100
8	0.47	0.76	0.58	100
9	0.68	0.73	0.71	100
accuracy			0.69	1000
macro avg	0.74	0.69	0.67	1000
weighted avg	0.74	0.69	0.67	1000

confusion\_matrix(true, pred)

```
[1] Chapter:I.3[[ 94, 0, 0, 0, 0, 0, 0,
 [ 0, 92, 0, 0, 0, 0, 0,
 [ 8, 2, 55, 11, 0, 0, 0,
 [ 0, 0, 0, 86, 0, 0, 0,
 [ 5, 0, 0, 0, 62, 0, 0,
 [16, 0, 0, 31, 1, 8, 6, 0, 33, 5],
 [11, 1, 3, 0, 3, 0, 81, 0, 1, 0],
 [ 3, 6, 5, 1, 1, 0, 0, 64, 10, 10],
 [ 4, 0, 0, 15, 1, 0, 2, 2, 76, 0],
 [ 3, 0, 1, 5, 7, 1, 1, 3, 6, 73]])
```

💡 The L2-Norm Algorithm brings even better results with an overall accuracy-value of 69%.

# Find a Value which distincts every image class

## 1. "Average Ink" Value

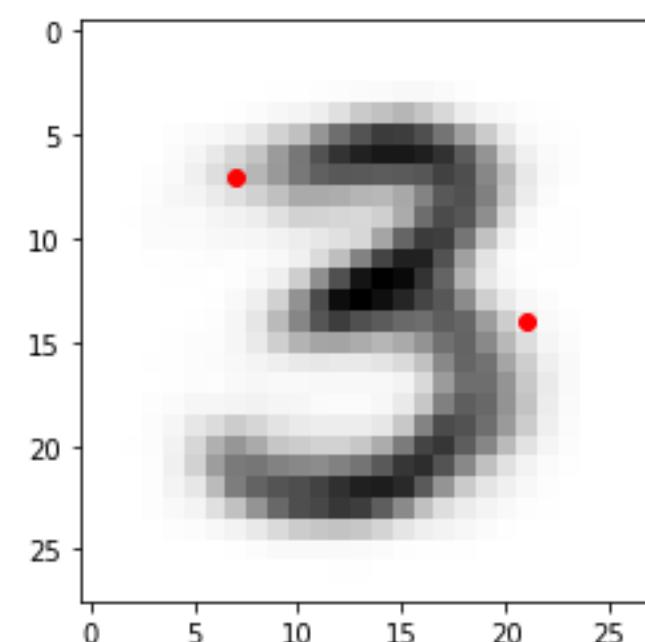
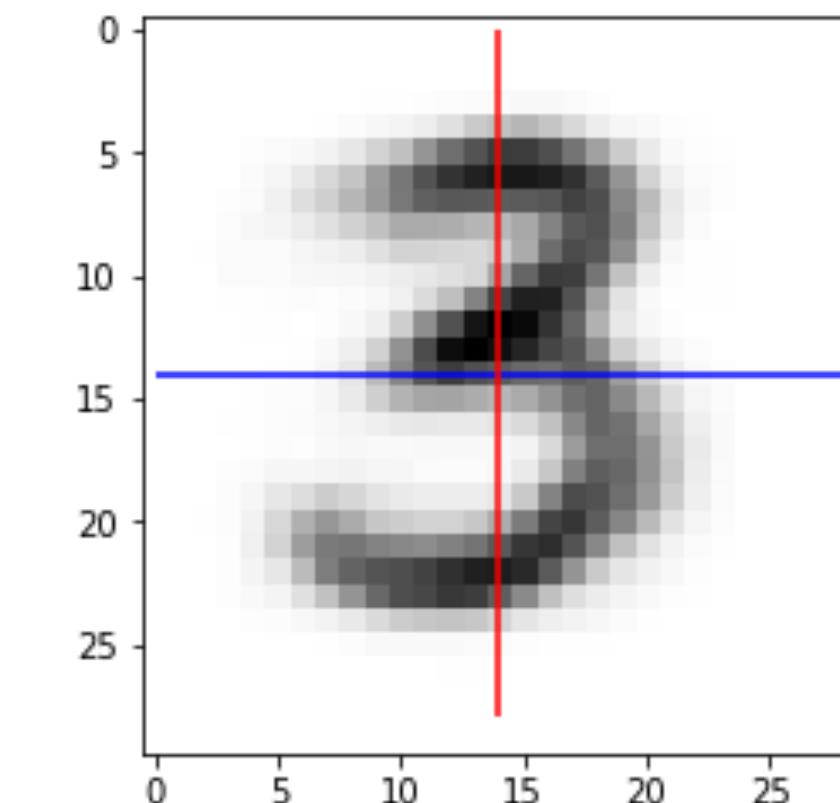
💻 sum of all pixels

## 2. "Middle Line" Ratio

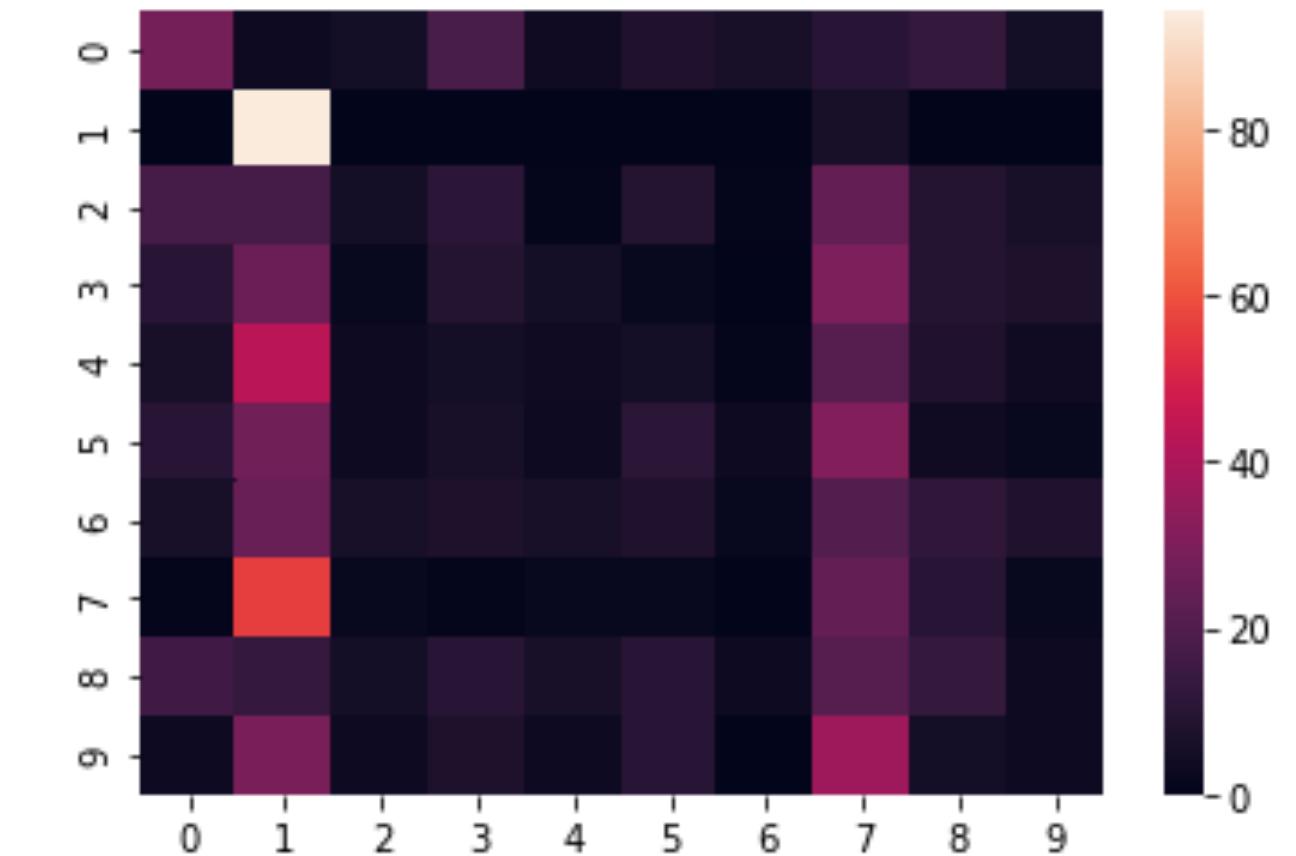
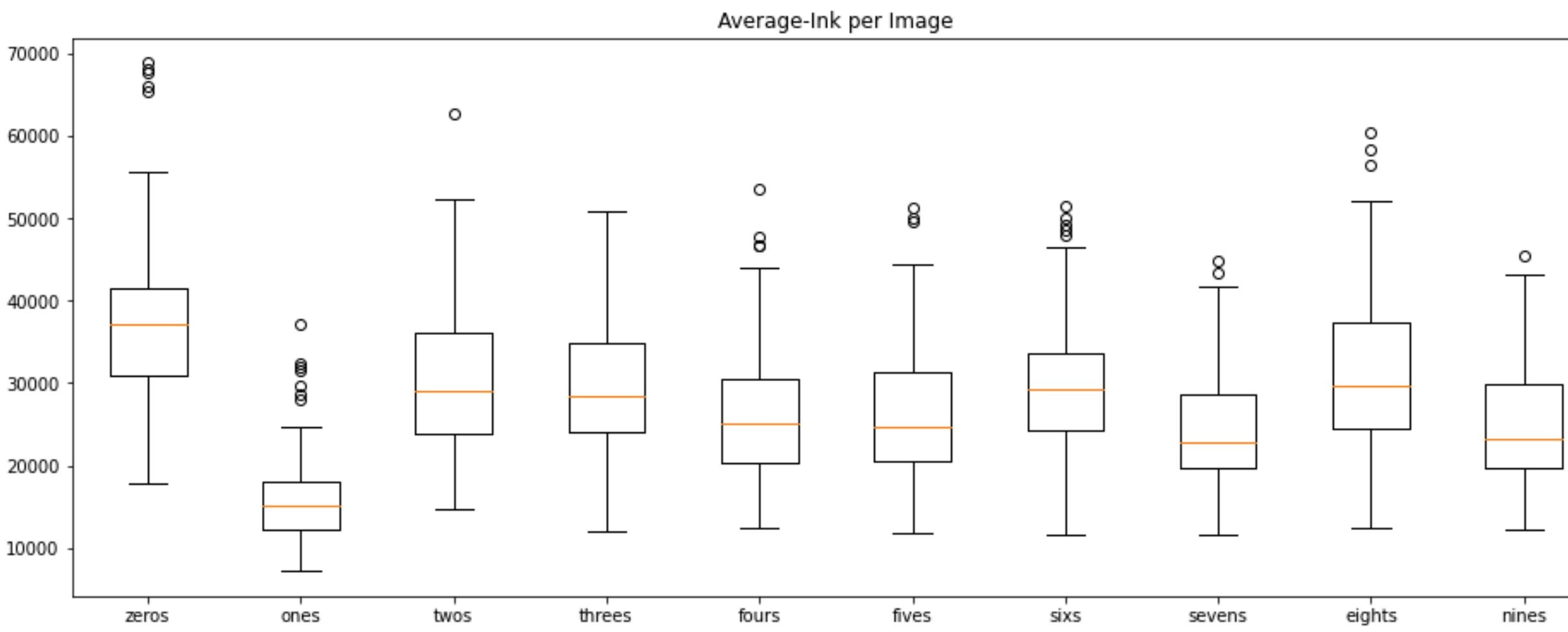
💻 sum of blue line / sum of red line

## 3. "Random Points" Value

💻 sum of pixel value of red points



# "Average Ink" Algo



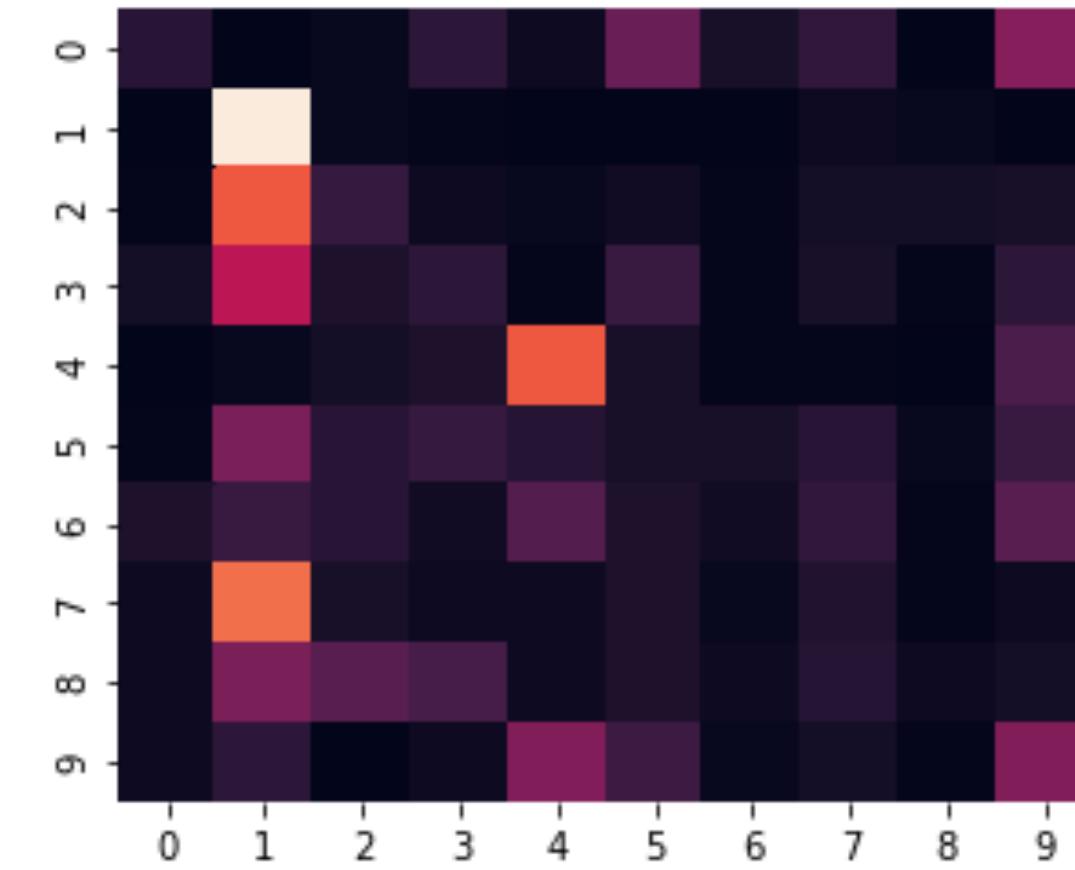
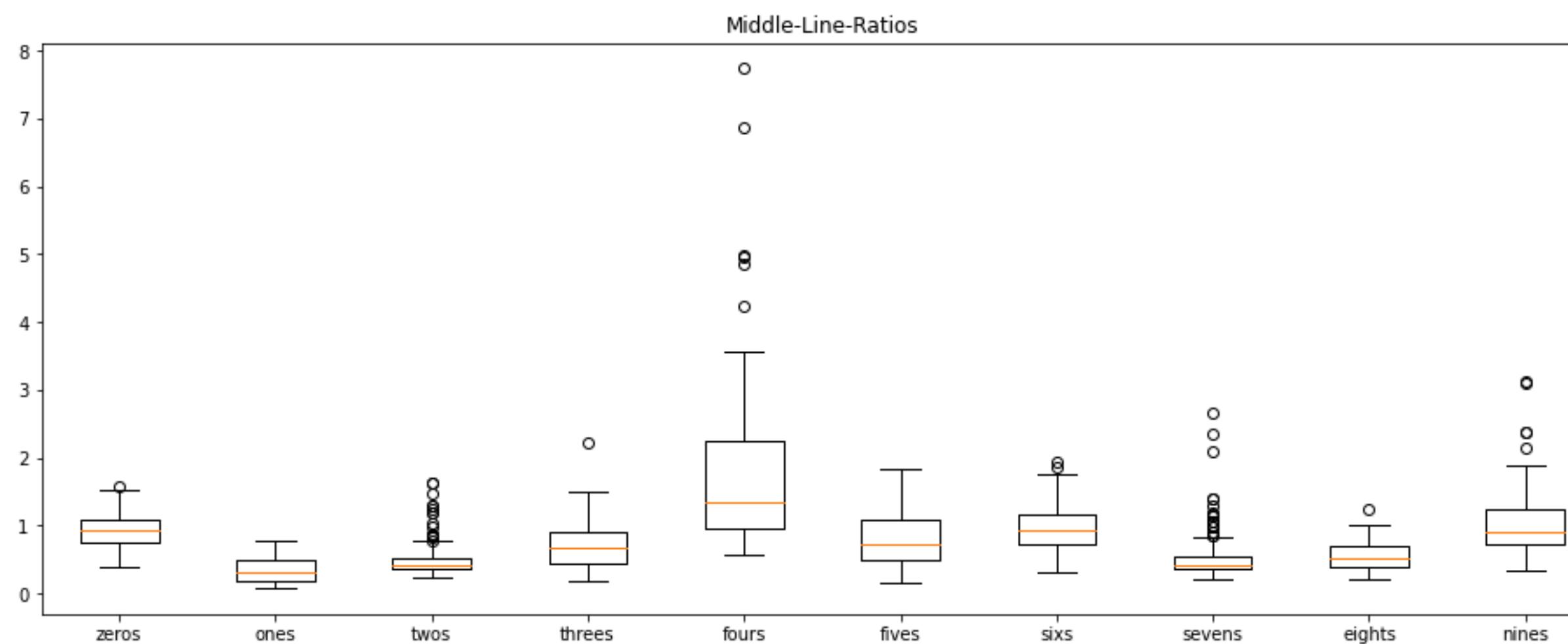
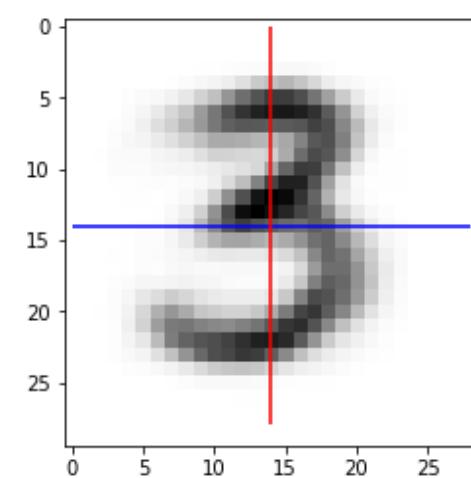
Classification report for classifier:

	precision	recall	f1-score	support
0	0.29	0.28	0.28	100
1	0.28	0.94	0.43	100
2	0.15	0.05	0.07	100
3	0.12	0.09	0.10	100
4	0.12	0.04	0.06	100
5	0.17	0.11	0.13	100
6	0.12	0.02	0.03	100
7	0.11	0.24	0.15	100
8	0.16	0.13	0.14	100
9	0.07	0.03	0.04	100
accuracy			0.19	1000
macro avg	0.16	0.19	0.15	1000
weighted avg	0.16	0.19	0.15	1000

```
: confusion_matrix(true, pred)
: array([[28,  3,  5, 18,  4,  8,  6, 10, 13,  5],
       [ 0, 94,  0,  0,  0,  0,  0,  6,  0,  0],
       [17, 17,  5, 11,  1,  9,  1, 24,  9,  6],
       [10, 26,  2,  9,  5,  2,  0, 30,  9,  7],
       [ 6, 43,  3,  5,  4,  5,  1, 21,  8,  4],
       [10, 27,  3,  6,  3, 11,  3, 31,  4,  2],
       [ 6, 25,  6,  7,  6,  8,  2, 20, 12,  8],
       [ 1, 56,  2,  1,  2,  2,  0, 24, 10,  2],
       [16, 13,  5, 10,  6, 10,  3, 21, 13,  3],
       [ 3, 29,  3,  7,  3, 10,  0, 37,  5,  3]])
```

智商低下的表情包 The Average-Ink Algorithms results are disappointing: the average accuracy is only 19% and probably because of its very simple formula so low.

# "Middle Line Ratio" Algo



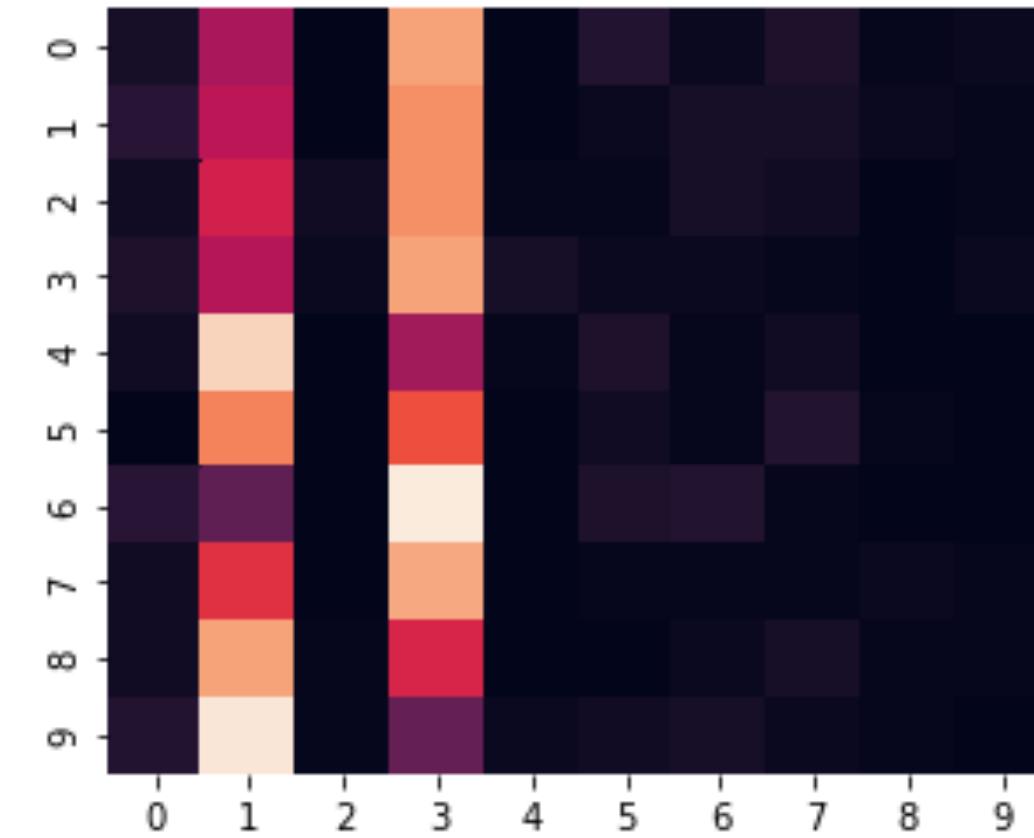
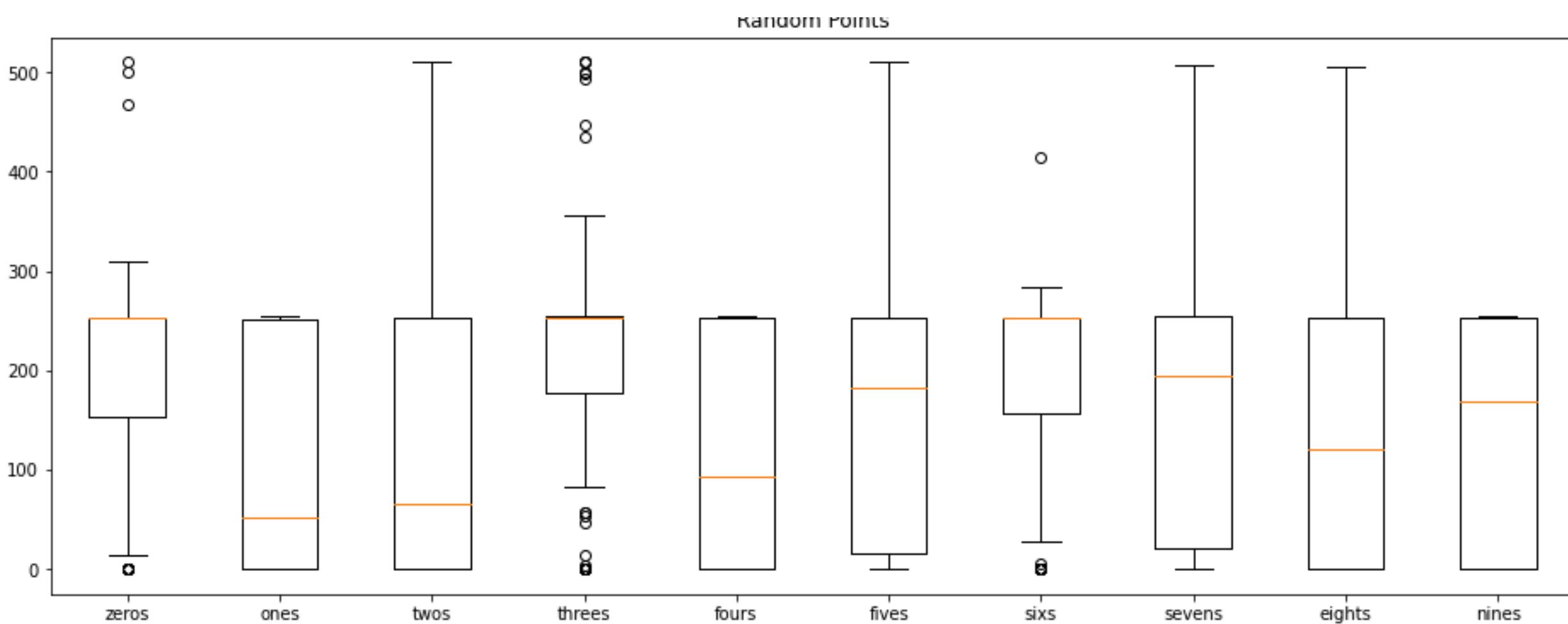
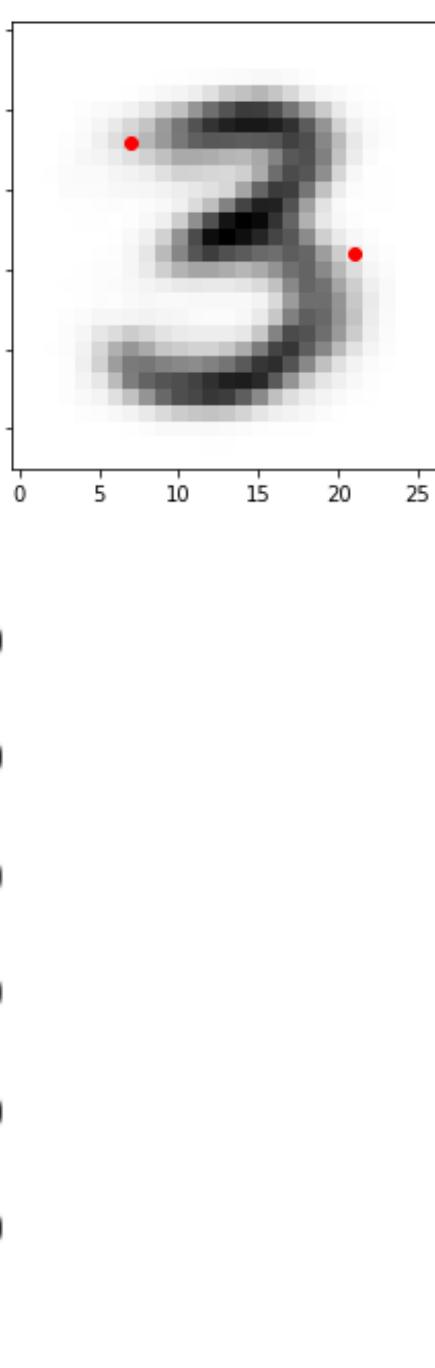
Classification report for classifier:				
	precision	recall	f1-score	support
0	0.30	0.10	0.15	100
1	0.27	0.92	0.41	100
2	0.17	0.13	0.15	100
3	0.15	0.11	0.13	100
4	0.46	0.60	0.52	100
5	0.07	0.06	0.06	100
6	0.15	0.04	0.06	100
7	0.11	0.08	0.09	100
8	0.19	0.03	0.05	100
9	0.22	0.30	0.25	100
accuracy			0.24	1000
macro avg	0.21	0.24	0.19	1000
weighted avg	0.21	0.24	0.19	1000

```
confusion_matrix(true, pred)
```

```
array([[10,  0,  2, 11,  3, 25,  6, 12,  0, 31],
       [ 0, 92,  2,  1,  0,  0,  0,  3,  2,  0],
       [ 1, 60, 13,  3,  2,  4,  1,  5,  5,  6],
       [ 5, 43,  7, 11,  1, 14,  1,  6,  1, 11],
       [ 0,  2,  5,  7, 60,  6,  1,  1,  0, 18],
       [ 1, 29, 10, 13,  9,  6,  6, 10,  2, 14],
       [ 7, 14, 10,  4, 20,  7,  4, 12,  1, 21],
       [ 3, 64,  6,  3,  3,  7,  2,  8,  1,  3],
       [ 3, 29, 21, 17,  3,  7,  3,  9,  3,  5],
       [ 3, 11,  0,  3, 30, 15,  2,  5,  1, 30]])
```

智商低下的表情包 The Middle-Line-Ratio Algorithm is almost as disappointing as the Average-Ink Algorithm. The average accuracy is only 24%.

# "Random Points" Algo



Classification report for classifier:

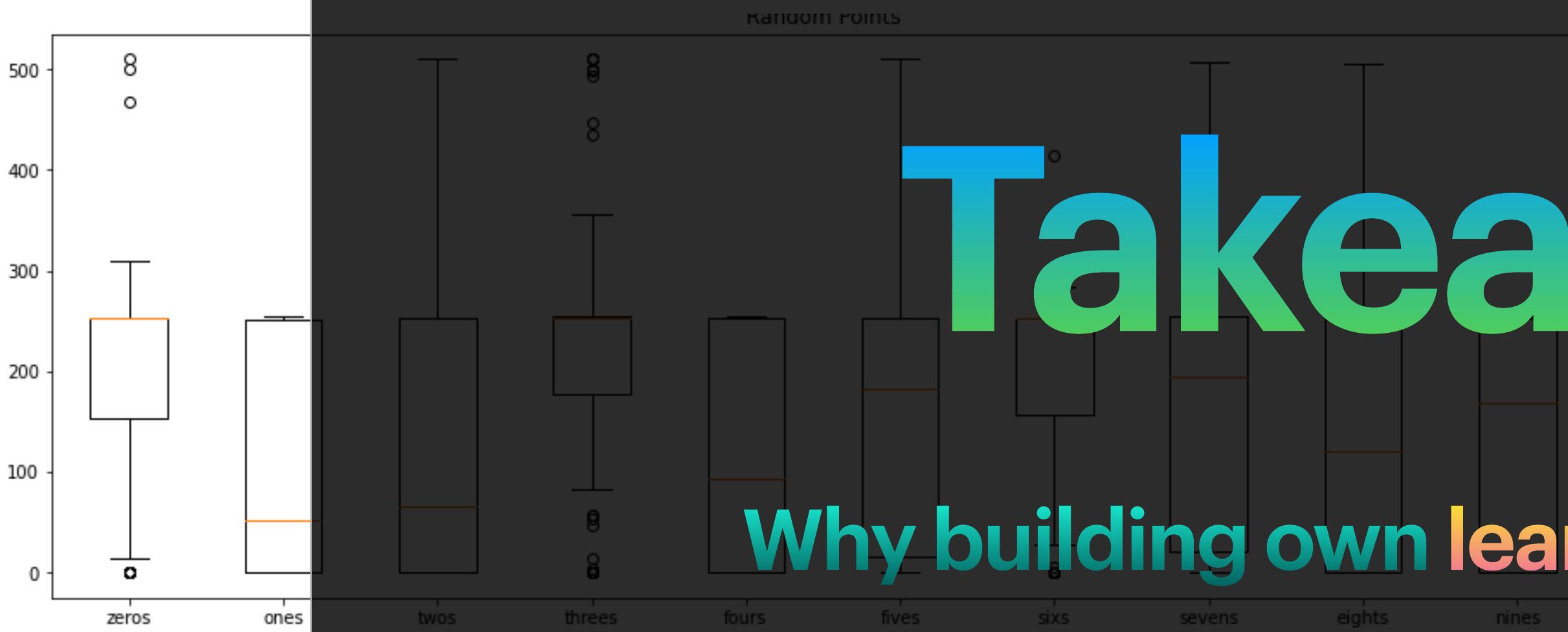
	precision	recall	f1-score	support
0	0.10	0.04	0.06	100
1	0.08	0.30	0.12	100
2	0.43	0.03	0.06	100
3	0.12	0.53	0.19	100
4	0.12	0.01	0.02	100
5	0.11	0.03	0.05	100
6	0.22	0.06	0.09	100
7	0.03	0.01	0.02	100
8	0.12	0.01	0.02	100
9	0.00	0.00	0.00	100
accuracy			0.10	1000
macro avg	0.13	0.10	0.06	1000
weighted avg	0.13	0.10	0.06	1000

```
confusion_matrix(true, pred)
```

```
array([[ 4, 27,  0, 53,  0,  6,  2,  5,  1,  2],
       [ 7, 30,  0, 50,  0,  2,  4,  4,  2,  1],
       [ 3, 34,  3, 50,  1,  1,  4,  3,  0,  1],
       [ 5, 29,  2, 53,  4,  2,  2,  1,  0,  2],
       [ 3, 61,  0, 26,  1,  5,  1,  3,  0,  0],
       [ 0, 48,  0, 41,  0,  3,  1,  6,  1,  0],
       [ 7, 16,  0, 65,  0,  5,  6,  1,  0,  0],
       [ 3, 37,  0, 54,  0,  1,  1,  1,  2,  1],
       [ 3, 53,  1, 35,  0,  0,  2,  4,  1,  1],
       [ 6, 64,  1, 17,  2,  3,  4,  2,  1,  0]])
```

智商低下的随机点算法仅能实现10%的准确率。但它的表现严重依赖于随机点的位置和数量，因此是不可靠的 ...

# "Random Points" Algo



# Takeaway

Why building own learning metrics?

Not a substitute for NN, but might be an enhancement:

Classification report for classifier:				
	precision	recall	f1-score	support
0	0.10	0.00	0.00	1000
1	0.08	0.30	0.12	100
2	0.43	0.03	0.06	100
3	0.12	0.01	0.02	100
4	0.12	0.01	0.02	100
5	0.11	0.03	0.05	100
6	0.22	0.01	0.02	100
7	0.03	0.01	0.02	100
8	0.12	0.01	0.02	100
9	0.00	0.00	0.00	1000
accuracy			0.10	1000
macro avg	0.13	0.10	0.06	1000
weighted avg	0.13	0.10	0.06	1000

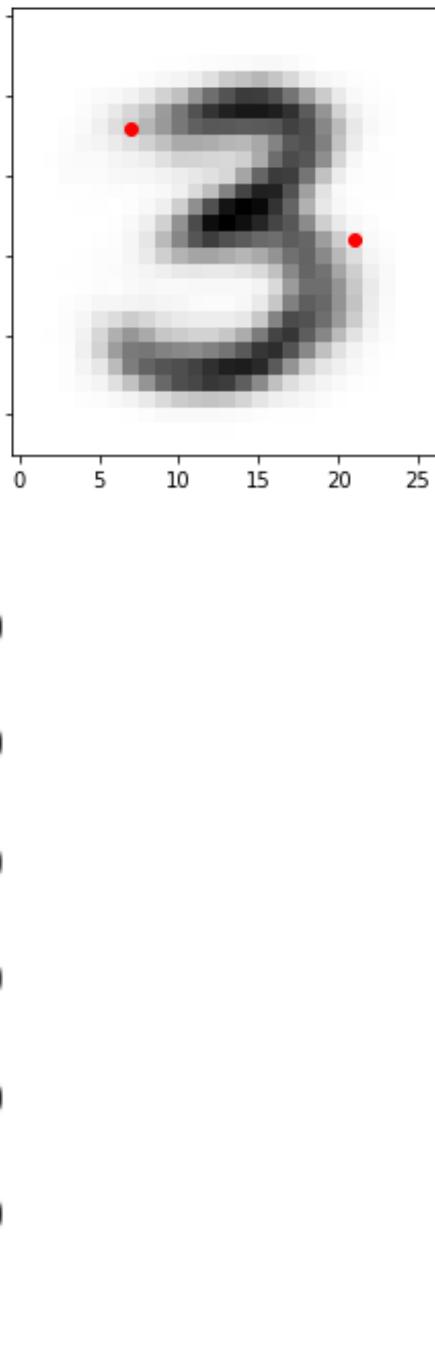
- can increase performance
- not enough classified sample images
- "extra boost" to stand out with your NN (e.g. Kaggle)
- Example:

confusion\_matrix(true, pred)

```
array([[ 4, 27,  0, 53,  0,  6,  2,  5,  1,  2],  
       [ 7, 30,  0, 50,  0,  2,  4,  4,  2,  1],  
       [ 3, 34,  3, 50,  1,  1,  4,  3,  0,  1],  
       [ 5, 29,  2, 53,  4,  2,  2,  1,  0,  2],  
       [ 6, 48,  0, 41,  6,  3,  1,  6,  1,  0],  
       [ 0, 48,  0, 41,  6,  3,  1,  6,  1,  0],  
       [ 7, 16,  0, 65,  0,  5,  6,  1,  0,  0],  
       [ 3, 37,  0, 54,  0,  1,  1,  1,  2,  1],  
       [ 3, 53,  1, 35,  0,  0,  2,  4,  1,  1],  
       [ 6, 64,  1, 17,  2,  3,  4,  2,  1,  0]])
```



The Random-Points Algorithm gives only 10% accuracy. But it heavily depends on where to place the points and how many. So it is improvable ...



# References

1. Howards, J & Gugger, S. (2020): Deep Learning for Coders with fastai & PTorch. O'Reilly or *corresponding Jupyter Notebook on Github*
2. Code on [my Github Repo](#)

