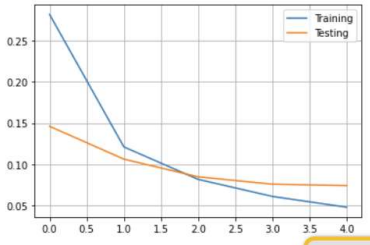
ML-PW-10

by Aurélien Héritier and Jean Nanchen

Exercice 1

ReLU
- Computes f(x) = max(0,x)
- Does not saturate
- Very computationally efficient
- Converges much faster than sigmoid/tanh in practice! (e.g. 6x)

```
Test score: 0.07383717596530914
Test accuracy: 0.9769999980926514
```



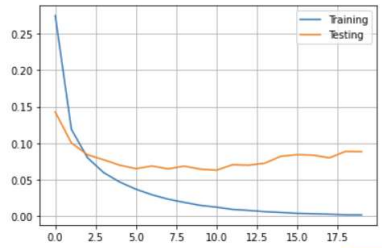
softmax,
epoch=5,
batch_size=128

Confusion matrix

```
[31] pred = model.predict(X_test)
     pred = np.argmax(pred, axis=-1)
     me.confusion_matrix(y_test, pred)

     array([[ 973,    0,    1,    1,    0,    0,    1,    1,    3,    0],
            [   0, 1114,    5,    1,    0,    1,    3,    1,   10,    0],
            [   5,    0, 1012,    3,    1,    0,    2,    5,    4,    0],
            [   0,    0,    6,  987,    0,    2,    0,    5,    2,    8],
            [   3,    0,    2,    1,  957,    0,    2,    2,    0,   15],
            [   2,    0,    0,    6,    1,  868,    7,    2,    4,    2],
            [   7,    2,    0,    1,    7,    7,  933,    0,    1,    0],
            [   1,    1,   11,    1,    2,    0,    0, 1000,    3,    9],
            [   6,    0,    6,    4,    4,    4,    4,    4,  938,    4],
            [   4,    2,    0,    3,    3,    1,    0,    5,    3,  988]])
```

```
Test score: 0.0883360430598259
Test accuracy: 0.9812999963760376
```



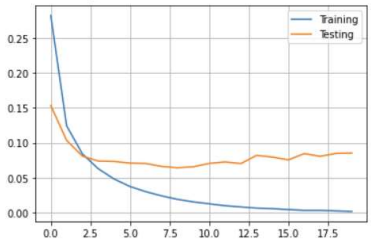
softmax, epoch=20, batch_size=128

Confusion matrix

```
pred = model.predict(X_test)
pred = np.argmax(pred, axis=-1)
me.confusion_matrix(y_test, pred)
```

```
array([[ 973,    0,    0,    1,    0,    1,    3,    1,    1,    0],
       [   0, 1126,    3,    0,    0,    1,    2,    1,    2,    0],
       [   3,    2, 1001,    7,    1,    0,    3,    8,    6,    1],
       [   0,    0,    1,  994,    0,    4,    0,    1,    3,    7],
       [   1,    0,    1,    0,  966,    0,    5,    2,    0,    7],
       [   2,    0,    0,    4,    1,  876,    4,    0,    4,    1],
       [   3,    2,    1,    1,    3,    4,  943,    1,    0,    0],
       [   1,    2,    6,    3,    1,    0,    0, 1005,    3,    7],
       [   3,    0,    4,    3,    5,    3,    2,    2,  946,    6],
       [   1,    2,    0,    3,   11,    4,    0,    4,    1,  983]])
```

```
Test score: 0.08526221662759781
Test accuracy: 0.9817000031471252
```



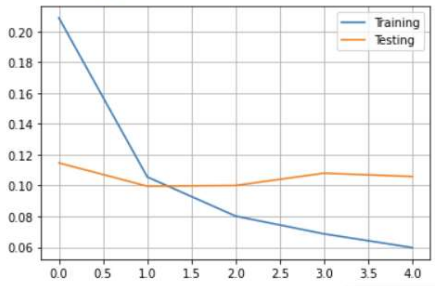
Confusion matrix

sigmoid, epoch=20, batch_size=128

```
pred = model.predict(X_test)
pred = np.argmax(pred, axis=-1)
me.confusion_matrix(y_test, pred)
```

```
array([[ 972,    0,    0,    1,    1,    0,    2,    1,    3,    0],
       [   0, 1125,    2,    1,    0,    1,    2,    1,    3,    0],
       [   3,    2, 1010,    1,    1,    0,    2,    5,    7,    1],
       [   0,    0,    5,  990,    0,    4,    0,    3,    3,    5],
       [   1,    1,    2,    1,  952,    0,    3,    3,    1,   18],
       [   2,    0,    0,    3,    0,  877,    3,    0,    5,    2],
       [   4,    2,    0,    1,    2,    5,  942,    0,    2,    0],
       [   1,    5,    7,    1,    0,    0,    0, 1002,    4,    8],
       [   0,    0,    2,    4,    3,    2,    1,    2,  957,    3],
       [   1,    3,    0,    2,    5,    2,    0,    2,    4,  990]])
```

```
Test score: 0.1058349460363388
Test accuracy: 0.9787999987602234
```



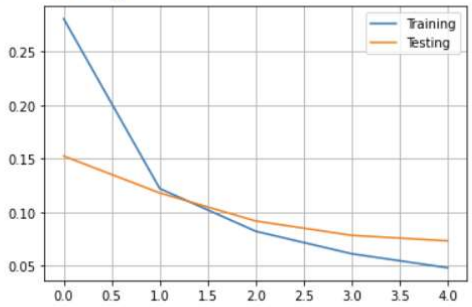
Confusion matrix

sigmoid, epoch=5, batch_size=16

```
pred = model.predict(X_test)
pred = np.argmax(pred, axis=-1)
me.confusion_matrix(y_test, pred)
```

```
array([[ 971,    0,    0,    1,    0,    0,    4,    1,    3,    0],
       [   0, 1126,    3,    1,    0,    0,    2,    0,    3,    0],
       [  20,    3,  991,    4,    1,    0,    3,    2,    8,    0],
       [   6,    0,    3,  991,    0,    2,    0,    3,    4,    1],
       [   3,    0,    3,    1,  955,    1,    8,    2,    1,    8],
       [  21,    0,    0,    7,    1,  850,    6,    1,    4,    2],
       [   7,    3,    0,    1,    2,    2,  942,    0,    1,    0],
       [   3,    4,   13,    4,    0,    1,    0,  992,    4,    7],
       [   8,    0,    3,    3,    3,    5,    3,    2,  943,    4],
       [   3,    2,    0,    7,    5,    4,    1,    3,    5,  979]])
```

```
Test score: 0.07305025309324265
Test accuracy: 0.9769999980926514
```



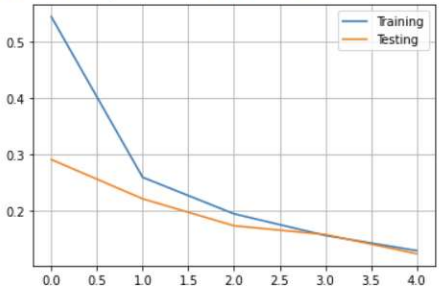
Confusion matrix

sigmoid, epoch=5, batch_size=128

```
pred = model.predict(X_test)
pred = np.argmax(pred, axis=-1
me.confusion_matrix(y_test, pr
```

```
array([[ 966,    0,    0,    1,    1,    3,    3,    2,    1,    3],
       [   0, 1122,    4,    2,    0,    1,    4,    0,    2,    0],
       [   5,    0, 1005,    3,    3,    0,    3,    8,    5,    0],
       [   0,    0,    4,  988,    0,    8,    0,    5,    2,    3],
       [   0,    0,    1,    0,  960,    0,    7,    2,    0,   12],
       [   2,    0,    0,    4,    2,  878,    5,    0,    0,    1],
       [   4,    2,    1,    1,    4,    6,  940,    0,    0,    0],
       [   1,    5,    7,    5,    1,    0,    0, 1002,    1,    6],
       [   8,    1,    5,   10,    4,   15,    4,    3,  918,    6],
       [   0,    3,    0,    5,    3,    3,    2,    2,    0,  991]])
```

```
Test score: 0.12375474721193314
Test accuracy: 0.9631999731063843
```



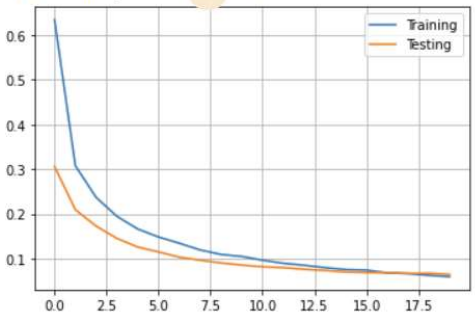
Confusion matrix

sigmoid, epoch=5, batch_size=1024

```
pred = model.predict(X_test)
pred = np.argmax(pred, axis=-1)
me.confusion_matrix(y_test, pred)
```

```
array([[ 970,    0,    0,    1,    0,    4,    2,    1,    1,    1],
       [   0, 1124,    4,    0,    0,    1,    3,    0,    3,    0],
       [   7,    3,  990,    4,    3,    1,    3,    6,   13,    2],
       [   0,    1,    6,  951,    0,   14,    0,   10,   19,    9],
       [   1,    1,    5,    0,  933,    0,    4,    3,    4,   31],
       [   2,    1,    0,    5,    2,  864,    7,    0,    6,    5],
       [   8,    3,    2,    0,    6,   12,  923,    0,    4,    0],
       [   0,   11,   15,    3,    0,    1,    0,  980,    1,   17],
       [   3,    2,    3,    7,    3,   10,    4,    5,  930,    7],
       [   4,    7,    1,    8,    8,    3,    1,    5,    5,  967]])
```

```
Test score: 0.06439674645662308
Test accuracy: 0.97[illegible]9791145325
```



Confusion matrix

sigmoid, epoch=5, batch_size=1024, add dropout

```
pred = model.predict(X_test)
pred = np.argmax(pred, axis=-1)
me.confusion_matrix(y_test, pred)
```

```
array([[ 969,    0,    2,    1,    1,    0,    2,    1,    3,    1],
       [   0, 1125,    3,    1,    0,    0,    3,    0,    3,    0],
       [   3,    2, 1008,    1,    3,    0,    3,    7,    5,    0],
       [   0,    0,    4,  993,    0,    2,    0,    6,    3,    2],
       [   0,    0,    3,    0,  965,    0,    5,    0,    2,    7],
       [   2,    0,    0,   10,    1,  865,    5,    2,    6,    1],
       [   5,    2,    1,    1,    4,    3,  938,    0,    4,    0],
       [   1,    6,   11,    1,    0,    0,    0, 1001,    1,    7],
       [   3,    0,    1,    4,    4,    1,    1,    4,  953,    3],
       [   3,    4,    0,    4,   11,    1,    0,    7,    2,  977]])
```

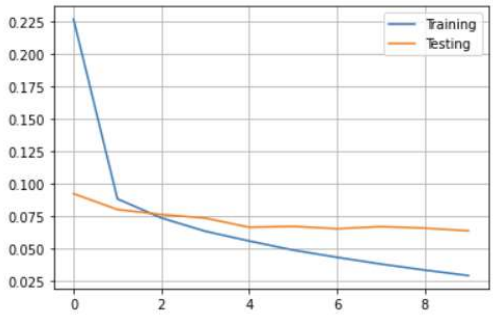
Best case is with the dropout, because it prevent the overfitting, epoch = 20, batch_size=1024

Exercice 2

It’s way better

With pix_p_cell = 4 :

```
Test score: 0.06342324614524841
Test accuracy: 0.9789999723434448
```



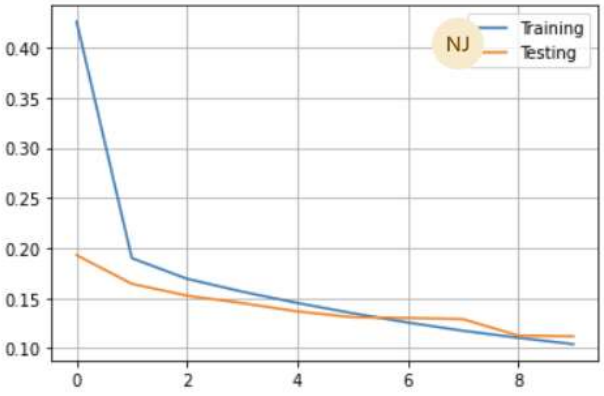
Confusion matrix

```
pred = model.predict(X_test_hog)
pred = np.argmax(pred, axis=-1)
me.confusion_matrix(y_test, pred)
```

```
array([[ 969,    0,    2,    0,    0,    2,    4,    1,    1,    1],
       [   3, 1121,    1,    3,    0,    2,    2,    2,    1,    0],
       [   2,    2, 1013,    1,    1,    0,    2,    6,    5,    0],
       [   0,    0,    2,  994,    0,    8,    1,    2,    3,    0],
       [   2,    2,    1,    0,  958,    0,    1,    0,    5,   13],
       [   2,    1,    0,   12,    0,  869,    4,    0,    1,    3],
       [   4,    2,    1,    0,    3,    2,  945,    0,    1,    0],
       [   0,    3,    8,    3,    3,    0,    0,  988,    5,   18],
       [   5,    0,    2,    6,    1,    1,    1,    4,  947,    7],
       [   0,    2,    1,    5,    6,    2,    0,    4,    3,  986]])
```

With pix_p_cell = 7 :
It's worse with pix_p_cell = 7, because the characteristics extracted are too complicated (not low level)

```
Test score: 0.11181405931711197
Test accuracy: 0.9632999897003174
```



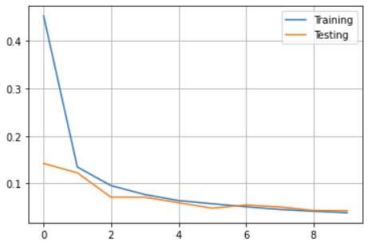
Confusion matrix

```
[ ]  pred = model.predict(X_test_hog)
     pred = np.argmax(pred, axis=-1)
     me.confusion_matrix(y_test, pred)
```

```
array([[ 964,    1,    2,    2,    1,    2,    4,    0,    4
       [   0, 1114,    3,    3,    4,    0,    5,    1,    5
       [   2,    2, 1001,   13,    0,    0,    2,    5,    7
       [   1,    0,    8,  957,    0,   23,    0,    5,   13
       [   0,    1,    3,    1,  947,    2,    2,    7,    2
       [   1,    0,    0,    8,    0,  869,    2,    0,   12
       [   8,    1,    0,    1,    4,   15,  920,    0,    8
       [   0,    4,   11,    4,    9,    0,    0,  982,    3
       [   2,    4,    7,   13,    4,   16,    5,    4,  913
       [   1,    3,    0,    5,    6,    6,    0,   14,    8
```

Exercice 3

```
[ ]  Test score: 0.04222225770354271
     Test accuracy: 0.9865000247955322
```



Confusion matrix

```
[ ]  pred = model.predict_on_batch(X_test)
     pred = np.argmax(pred, axis=-1)
     me.confusion_matrix(y_test, pred)
```

```
array([[ 966,    0,    5,    0,    0,    0,    7,    1,    1,    0],
       [   0, 1129,    1,    0,    0,    0,    1,    3,    1,    0],
       [   1,    0, 1027,    0,    0,    0,    1,    3,    0,    0],
       [   0,    0,    5,  997,    0,    2,    0,    5,    1,    0],
       [   0,    0,    2,    0,  973,    0,    4,    1,    0,    2],
       [   2,    1,    1,    5,    0,  877,    3,    2,    0,    1],
       [   2,    2,    0,    1,    1,    1,  949,    0,    2,    0],
       [   0,    1,   10,    2,    0,    1,    0, 1012,    1,    1],
       [   1,    2,    9,    1,    1,    1,    1,    3,  951,    4],
       [   3,    2,    1,    0,    7,    4,    0,    8,    0,  984]])
```

It use convolution layer and down-sampling layer to select the best feature and send it to the next layer. With this iterations, it can achieve the best test score.