Deep Neural Networks

Apprentissage par réseaux de neurones artificiels

ANNEN Rayane, MARTINS Alexis

HEIG VD

12.04.2023

Contents

Digit recognition from raw data

What is the learning algorithm being used to optimize the weights of the neural networks? What are the parameters (arguments) being used by that algorithm? What cost function is being used ? please, give the equation(s)

The algorithm used to optimize the weight is RMSprop (Root Mean Square Propagation).

The parameters are the following:

```
tf.keras.optimizers.RMSprop(
    learning_rate=0.001,
    rho=0.9,
    momentum=0.0,
    epsilon=1e-07,
    centered=False,
    weight_decay=None,
    clipnorm=None,
    clipvalue=None,
    global_clipnorm=None,
    use_ema=False,
    ema_momentum=0.99,
    ema_overwrite_frequency=100,
    jit_compile=True,
    name="RMSprop",
    **kwargs
)
```

The following equations are used:

$$E[g^2]_t = \beta E[g^2]_{t-1} + (1-\beta)\left(\frac{\partial C}{\partial w}\right)^2$$

$$w_t = w_{t-1} - \frac{\eta}{\sqrt{E[g^2]_t}}\frac{\partial C}{\partial w}$$

where η is the learning rate, w_t the new weight, β is the moving average parameter, $E[g]$ is the moving average of squared gradients and $\frac{\partial C}{\partial w}$ is the derivative of the cost function with respect to the weight.

The cost function is the categorical cross-entropy loss function:

$$\mathsf{CE} = -\frac{1}{N}\sum_{k=0}^{N} \log \vec{p_i}[y_i]$$

where N is the number of samples, $\vec{p_i}$ is the neural network output and y_i is the target class index.

Shallow Neural Network

For this experiment a simple shallow neural network is used. We use raw data to classify the digits.

Hyper-parameters

Changed made to the model from the original: we reduced the number of neurons in the hidden layer from 300 to 100.

- Epochs: 10
- Hidden layers:
  - 100 neurons, reLU
- Output activation function: softmax.
- Batch size: 128
- Weights in the hidden layer: 784 * 100 + 100 = 78400 + 100 = 78500
- Weights in the output layer = 10 * 100 + 10 = 1010
- Total weights: 78500 + 1010 = 79510

Digit recognition from features of the input data

Shallow Neural Network

In this experiment, we use the Histogram of gradients (HOG) features to classify the digits.

Hyper-parameters

HOG: - orientation count: 8 - pixels per cell: 4

- Epochs: 10
- Hidden layers:
  - 100 neurons, reLU
- Output activation function: softmax.
- Batch size: 128

- Weights in the hidden layer: 392 * 200 + 200 = 78400 + 200 = 78600
- Weights in the output layer = 10 * 200 + 10 = 2010
- Total weights : 78600 + 2010 = 80610

Convolutional neural network digit recognition

Deep Convolutional Neural Network

- Epochs: 10
- Batch size: 128
- Hidden layers:
  - Convolutional 2D: 5x5
  - MaxPooling 2D: pool size: 2x2
  - Convolutional 2D: 5x5
  - MaxPooling 2D: pool size: 2x2
  - Convolutional 2D: 3x3
  - MaxPooling 2D: 2x2
  - Flatten layer
  - Dense (25 neurons), activation function: reLU

Output: 10 neurons, activation function: softmax

Model complexity :

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 l0 (InputLayer)             [(None, 28, 28, 1)]       0

 l1 (Conv2D)                 (None, 28, 28, 9)         234

 l1_mp (MaxPooling2D)        (None, 14, 14, 9)         0

 l2 (Conv2D)                 (None, 14, 14, 9)         2034

 l2_mp (MaxPooling2D)        (None, 7, 7, 9)           0

 l3 (Conv2D)                 (None, 7, 7, 16)          1312

 l3_mp (MaxPooling2D)        (None, 3, 3, 16)          0

```

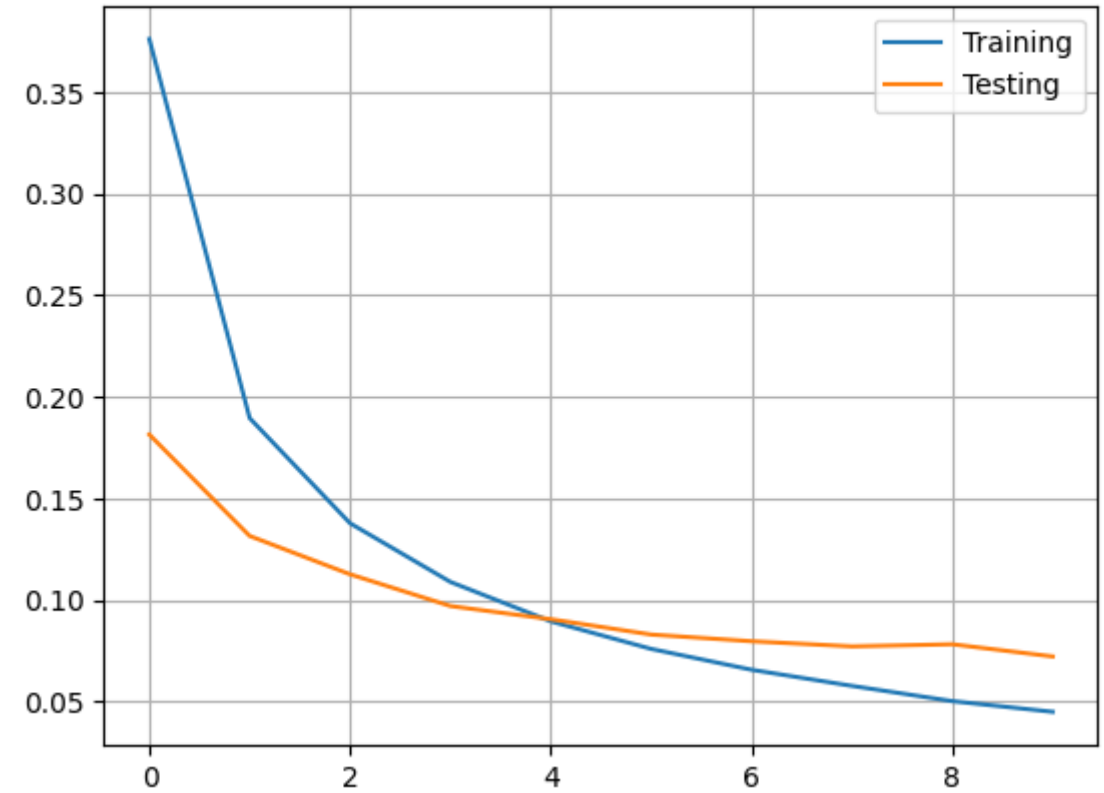
```
 flat (Flatten)              (None, 144)               0

 l4 (Dense)                  (None, 25)                3625

 l5 (Dense)                  (None, 10)                260

=================================================================
Total params: 7,465
Trainable params: 7,465
Non-trainable params: 0
_________________________________________________________________
```

Experiments

Raw data

100 neurones

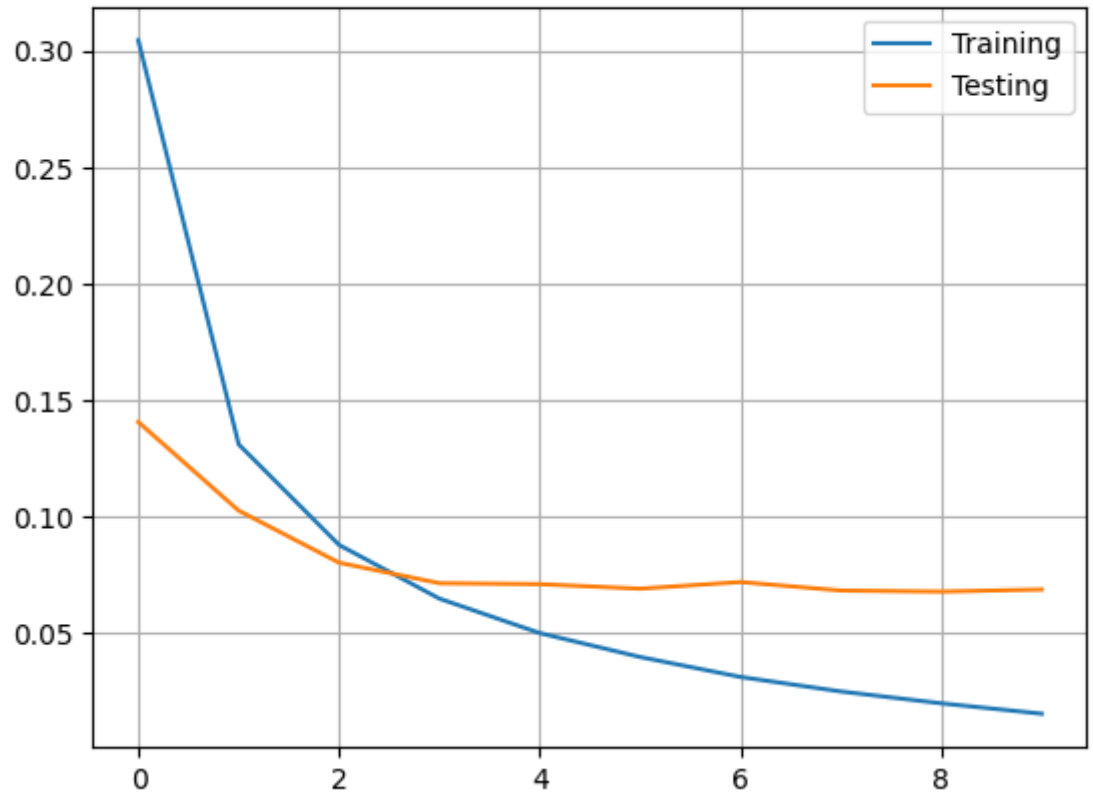
```
Test score: 0.07702907919883728
Test accuracy: 0.9757000207901001
```



```
313/313 [==============================] - 1s 2ms/step
array([[ 972,    0,    1,    1,    0,    0,    2,    1,    3,    0],
       [   0, 1126,    2,    0,    0,    2,    2,    1,    2,    0],
       [   2,    1, 1016,    1,    3,    0,    2,    3,    4,    0],
       [   1,    0,    7,  994,    0,    2,    0,    1,    3,    2],
       [   2,    0,    2,    1,  971,    0,    2,    1,    0,    3],
       [   4,    0,    0,   11,    1,  863,    7,    0,    5,    1],
       [   3,    3,    1,    1,    3,    3,  943,    0,    1,    0],
       [   0,    4,   10,    4,    3,    0,    0,  998,    3,    6],
       [   3,    0,    2,    5,    2,    2,    1,    3,  954,    2],
       [   2,    2,    0,    7,   13,    4,    1,    3,    2,  975]])
```

300 neurones

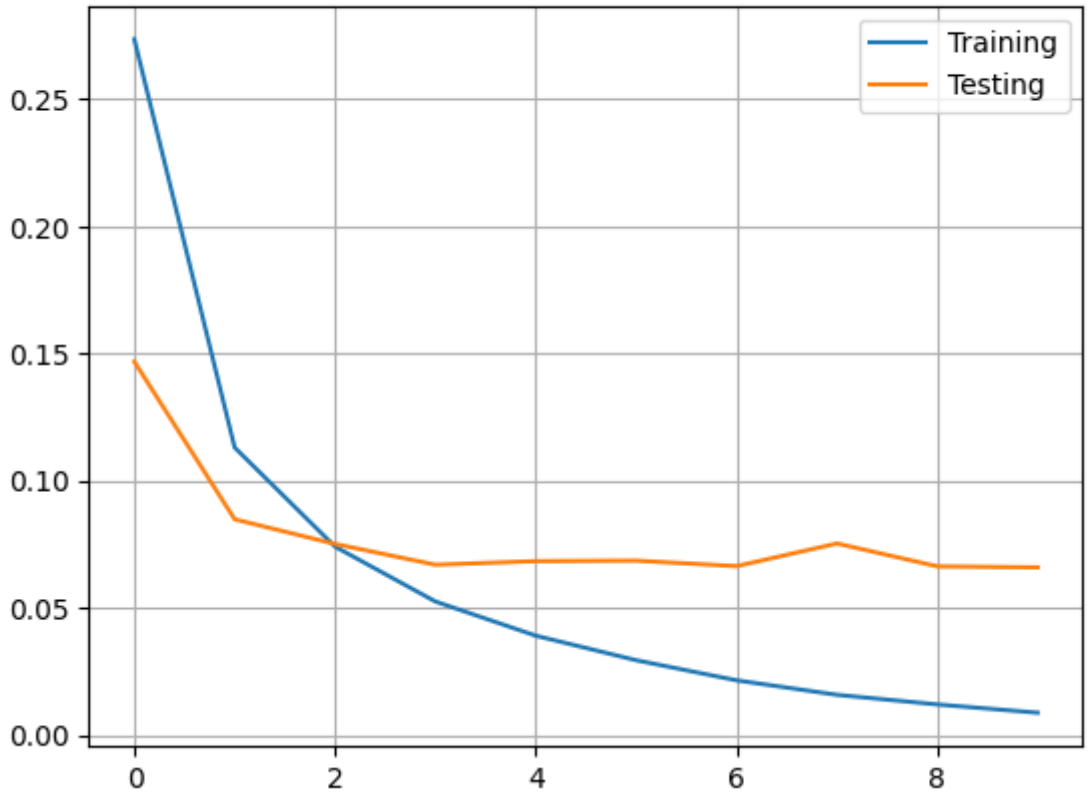
```
Test score: 0.06406917423009872
Test accuracy: 0.9811999797821045
```



```
313/313 [==============================] - 1s 2ms/step
array([[ 972,    0,    1,    1,    0,    0,    2,    1,    3,    0],
       [   0, 1126,    2,    0,    0,    2,    2,    1,    2,    0],
       [   2,    1, 1016,    1,    3,    0,    2,    3,    4,    0],
       [   1,    0,    7,  994,    0,    2,    0,    1,    3,    2],
       [   2,    0,    2,    1,  971,    0,    2,    1,    0,    3],
       [   4,    0,    0,   11,    1,  863,    7,    0,    5,    1],
       [   3,    3,    1,    1,    3,    3,  943,    0,    1,    0],
       [   0,    4,   10,    4,    3,    0,    0,  998,    3,    6],
       [   3,    0,    2,    5,    2,    2,    1,    3,  954,    2],
       [   2,    2,    0,    7,   13,    4,    1,    3,    2,  975]])
```

600 neurones

```
Test score: 0.060618676245212555
Test accuracy: 0.9815000295639038
```

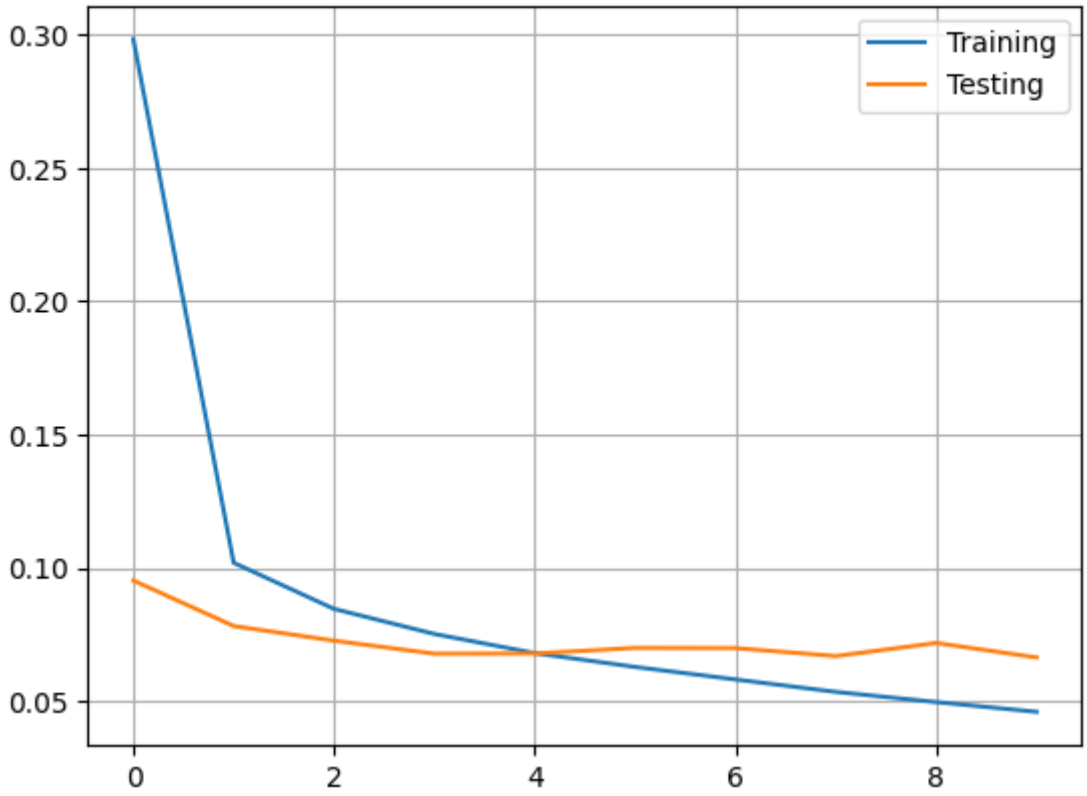


```
313/313 [==============================] - 1s 2ms/step
array([[ 970,    1,    2,    0,    1,    0,    3,    1,    2,    0],
       [   0, 1128,    2,    1,    0,    1,    2,    1,    0,    0],
       [   4,    2, 1005,    3,    3,    0,    2,    5,    8,    0],
       [   1,    0,    4,  994,    0,    5,    0,    1,    3,    2],
       [   0,    0,    1,    1,  967,    0,    6,    1,    0,    6],
       [   2,    0,    0,    5,    1,  871,    6,    0,    5,    2],
       [   3,    2,    1,    1,    3,    2,  945,    0,    1,    0],
       [   2,    5,    7,    4,    0,    0,    0, 1002,    4,    4],
       [   1,    2,    3,    4,    3,    2,    1,    2,  955,    1],
       [   3,    3,    0,    5,   10,    2,    1,    4,    3,  978]])
```

Features-based (HOG)

PIX_P_CELL 4, orientation 8, 100 neurones

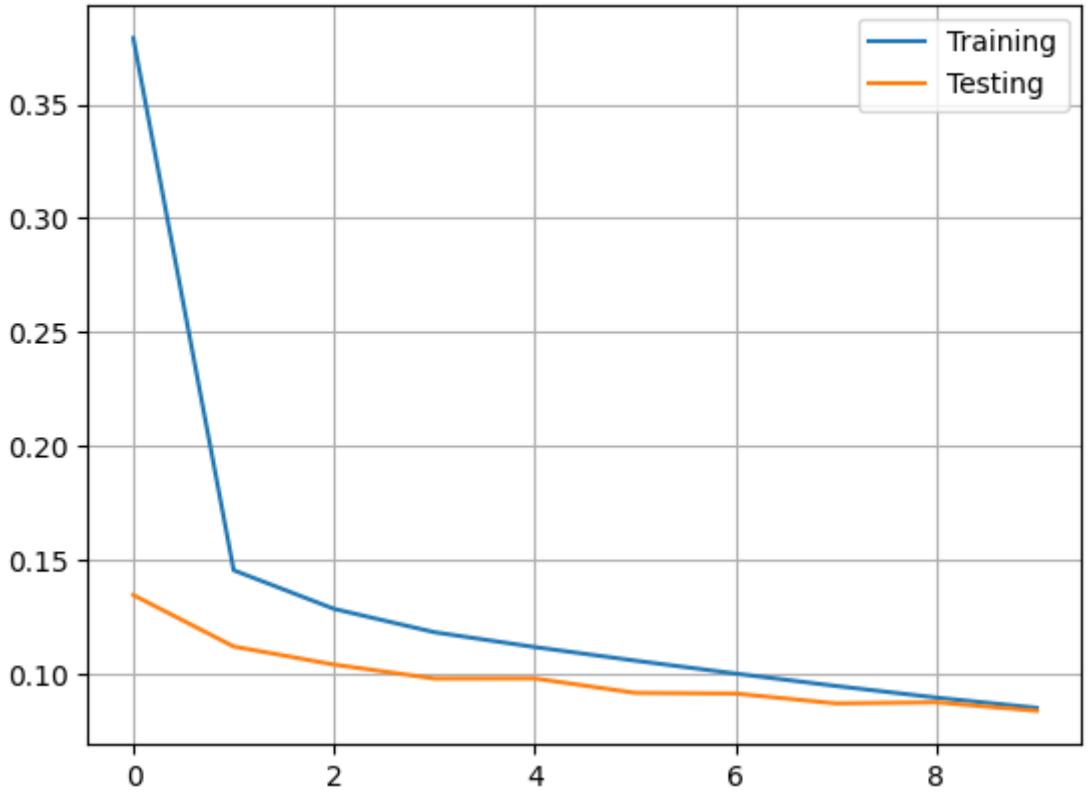
```
Test score: 0.06830427795648575
Test accuracy: 0.9771999716758728
```



```
313/313 [==============================] - 1s 1ms/step
array([[ 971,    0,    1,    0,    0,    2,    3,    2,    0,    1],
       [   4, 1117,    1,    3,    1,    1,    4,    2,    2,    0],
       [   2,    1, 1015,    2,    2,    0,    2,    5,    3,    0],
       [   0,    1,    4,  986,    0,   10,    0,    4,    5,    0],
       [   3,    1,    3,    0,  955,    0,    1,    1,    3,   15],
       [   2,    1,    0,   10,    0,  875,    4,    0,    0,    0],
       [   4,    2,    1,    0,    3,    3,  945,    0,    0,    0],
       [   0,    2,    7,    3,    5,    0,    0,  995,    3,   13],
       [   7,    0,    6,    8,    2,    2,    4,    4,  927,   14],
       [   1,    2,    1,    5,    7,    1,    0,    5,    1,  986]])
```

PIX_P_CELL 4, orientation 8, 300 neurones

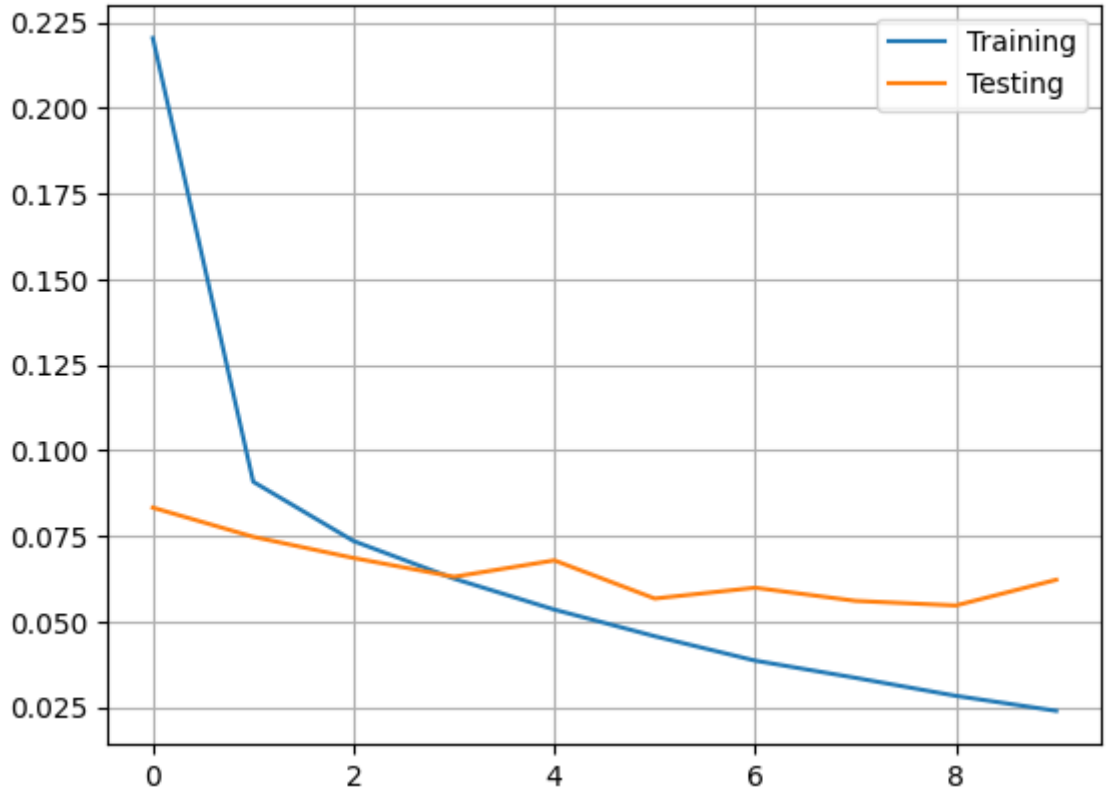
```
Test score: 0.09305289387702942
Test accuracy: 0.9696000218391418
```



```
313/313 [==============================] - 0s 1ms/step
array([[ 966,    1,    2,    0,    0,    3,    4,    1,    3,    0],
       [   0, 1123,    3,    1,    0,    0,    4,    0,    4,    0],
       [   4,    7,  998,    4,    1,    0,    2,    8,    7,    1],
       [   0,    1,    2,  976,    0,   12,    0,    2,   16,    1],
       [   3,    2,    3,    0,  956,    0,    4,    2,    4,    8],
       [   4,    1,    0,   13,    0,  863,    5,    0,    5,    1],
       [   5,    1,    1,    0,    5,    4,  939,    0,    3,    0],
       [   2,    5,    6,    3,    7,    0,    0,  993,    3,    9],
       [   7,    2,    2,   11,    3,    3,    3,    3,  936,    4],
       [   5,    6,    0,   10,   18,    6,    1,   11,    6,  946]])
```

PIX_P_CELL 4, orientation 4, 100 neurones

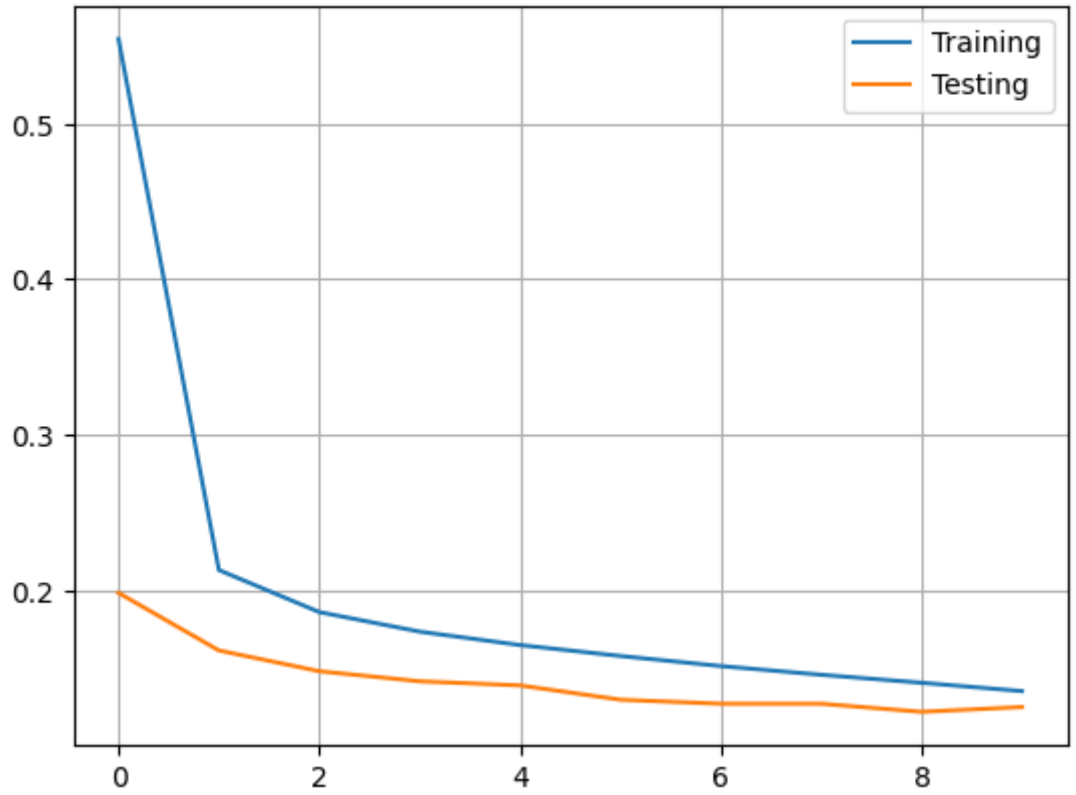
```
Test score: 0.06113172695040703
Test accuracy: 0.9801999926567078
```



```
313/313 [==============================] - 0s 1ms/step
array([[ 972,    0,    1,    0,    0,    1,    3,    1,    1,    1],
       [   4, 1122,    1,    2,    0,    1,    2,    0,    3,    0],
       [   2,    4, 1017,    1,    1,    0,    2,    2,    3,    0],
       [   0,    1,    2,  987,    0,    6,    1,    3,   10,    0],
       [   1,    1,    1,    0,  962,    0,    0,    1,    3,   13],
       [   2,    1,    0,   12,    0,  869,    4,    0,    1,    3],
       [   4,    2,    1,    0,    3,    2,  945,    0,    1,    0],
       [   0,    4,    8,    4,    4,    0,    0,  988,    3,   17],
       [   6,    0,    4,    2,    1,    1,    1,    1,  949,    9],
       [   0,    3,    1,    3,    5,    1,    0,    4,    1,  991]])
```

PIX_P_CELL 7, orientation 8, 100 neurones

```
Test score: 0.13366223871707916
Test accuracy: 0.9563999772071838
```

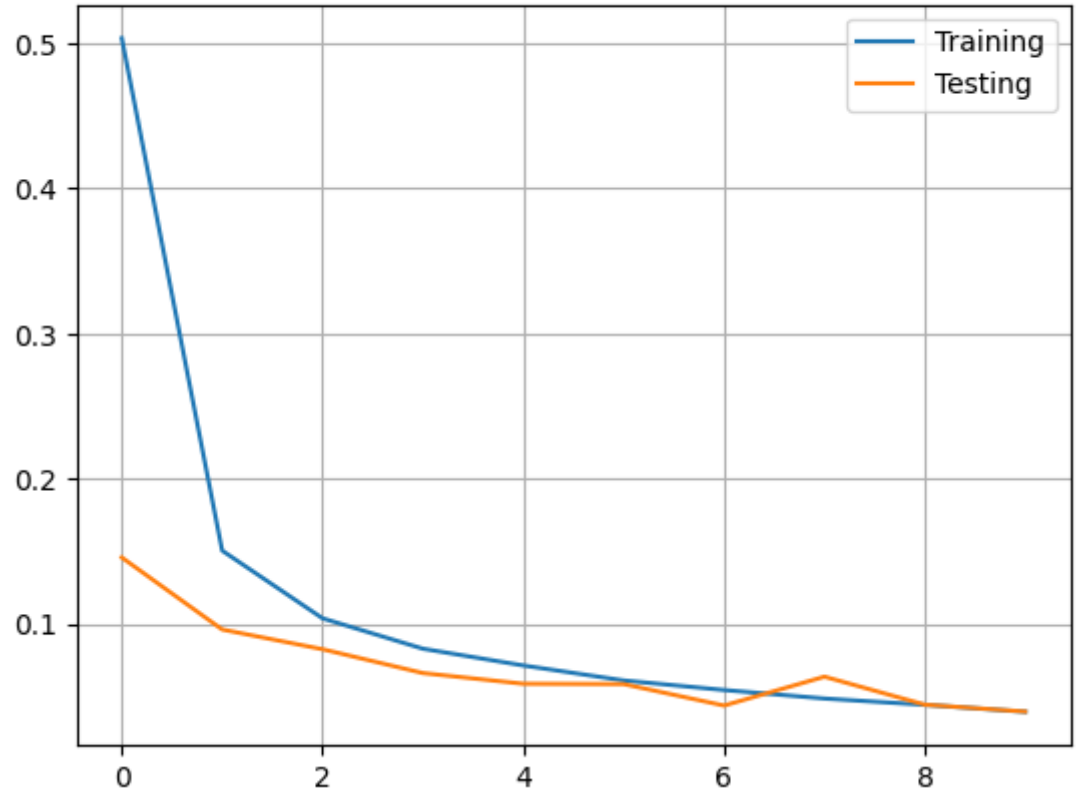


```
313/313 [==============================] - 1s 2ms/step
array([[ 950,    1,    4,    1,    0,    1,   10,    1,    9,    3],
       [   0, 1109,    3,    2,    3,    1,    7,    1,    9,    0],
       [   3,    2,  989,   10,    6,    1,    2,    7,   11,    1],
       [   1,    1,    7,  940,    2,   25,    1,    7,   24,    2],
       [   0,    2,    2,    0,  953,    1,    1,    4,    5,   14],
       [   1,    0,    1,    8,    0,  865,    1,    0,   15,    1],
       [   4,    2,    0,    1,    8,   10,  927,    0,    4,    2],
       [   0,    5,   16,    5,    8,    0,    0,  957,   10,   27],
       [   2,    3,    2,    8,    4,   15,    3,    5,  924,    8],
       [   0,    3,    2,   11,   15,    5,    0,   12,   11,  950]])
```

CNN

25 neurones

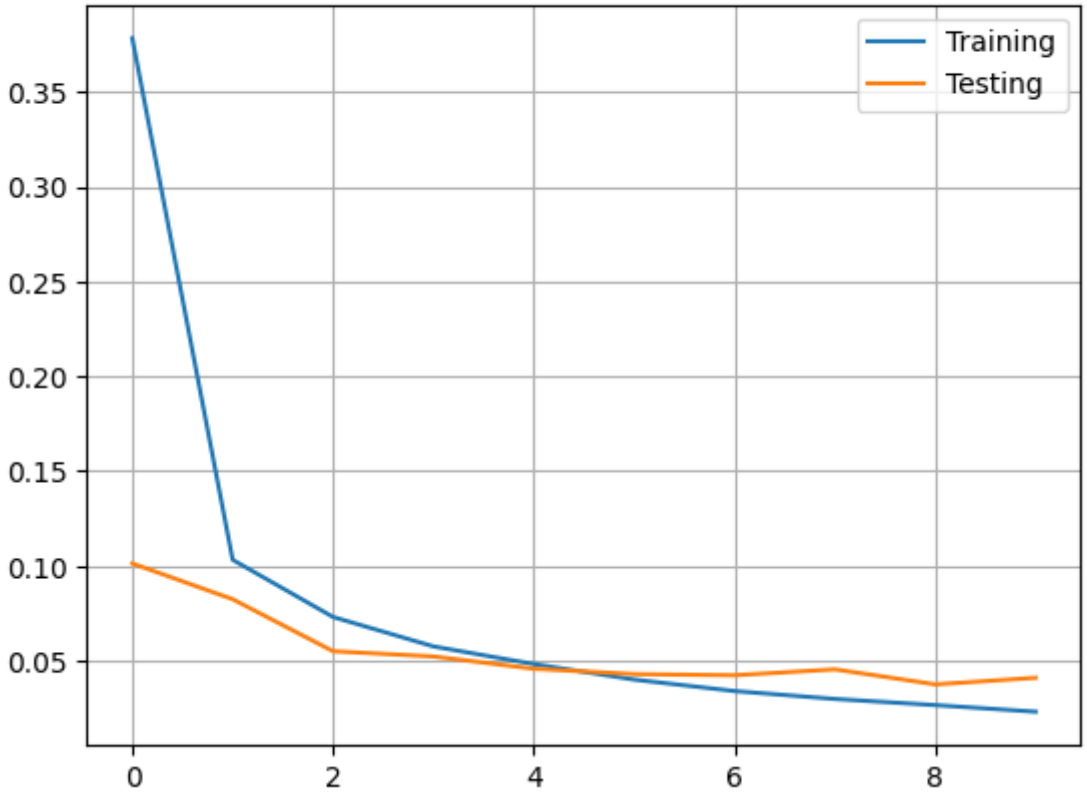
```
Test score: 0.03834224492311478
Test accuracy: 0.9879999756813049
```



```
313/313 [==============================] - 1s 2ms/step
pred.shape =  (10000, 10)
array([[ 975,    0,    0,    0,    0,    0,    1,    1,    3,    0],
       [   1, 1127,    2,    0,    0,    2,    0,    1,    2,    0],
       [   3,    2, 1017,    2,    0,    0,    0,    7,    1,    0],
       [   0,    0,    1, 1003,    0,    3,    0,    2,    1,    0],
       [   0,    0,    2,    0,  971,    0,    4,    2,    0,    3],
       [   0,    1,    0,    6,    0,  882,    1,    0,    1,    1],
       [   5,    2,    0,    0,    2,    6,  939,    0,    4,    0],
       [   0,    2,    2,    2,    0,    1,    0, 1020,    1,    0],
       [   4,    0,    2,    0,    0,    1,    0,    2,  964,    1],
       [   2,    3,    0,    3,    6,    3,    0,    6,    4,  982]])
```

250 neurones

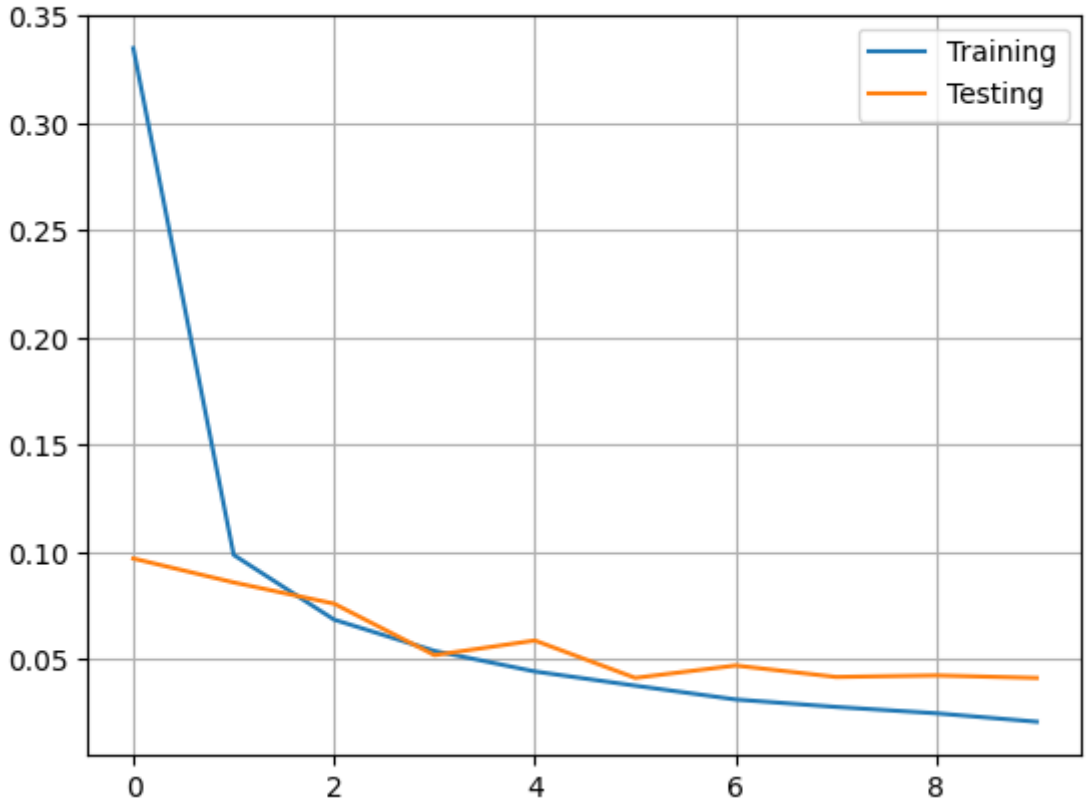
```
Test score: 0.037620659917593
Test accuracy: 0.9884999990463257
```



```
313/313 [==============================] - 1s 2ms/step
pred.shape =  (10000, 10)
array([[ 975,    0,    1,    0,    0,    0,    1,    0,    2,    1],
       [   1, 1131,    2,    0,    0,    0,    1,    0,    0,    0],
       [   2,    0, 1028,    0,    0,    0,    0,    1,    1,    0],
       [   0,    0,    4,  999,    0,    3,    0,    2,    1,    1],
       [   0,    1,    3,    0,  964,    0,    4,    0,    3,    7],
       [   2,    0,    0,    5,    0,  882,    1,    0,    0,    2],
       [   7,    1,    1,    0,    1,    3,  945,    0,    0,    0],
       [   0,    1,   12,    0,    0,    0,    0, 1012,    1,    2],
       [   4,    0,    4,    2,    0,    2,    1,    2,  955,    4],
       [   0,    1,    0,    0,    5,    4,    0,    1,    4,  994]])
```

500 neurones

```
Test score: 0.04010535404086113
Test accuracy: 0.9871000051498413
```



```
313/313 [==============================] - 1s 2ms/step
pred.shape =  (10000, 10)
array([[ 973,    0,    0,    0,    0,    2,    2,    0,    2,    1],
       [   0, 1131,    1,    1,    0,    0,    1,    1,    0,    0],
       [   1,    3, 1008,   12,    1,    0,    0,    5,    1,    1],
       [   0,    0,    0, 1006,    0,    3,    0,    0,    1,    0],
       [   0,    1,    0,    0,  975,    0,    0,    0,    1,    5],
       [   1,    0,    0,   10,    0,  877,    1,    0,    0,    3],
       [   1,    2,    0,    1,    2,   13,  938,    0,    1,    0],
       [   0,    1,    5,    3,    1,    0,    0, 1010,    1,    7],
       [   0,    0,    0,   15,    1,    1,    0,    1,  953,    3],
       [   0,    0,    0,    2,    3,    2,    0,    1,    1, 1000]])
```

General questions

Do the deep neural networks have much more “capacity” (i.e., do they have more weights?) than the shallow ones? explain with one example

Yes, deep neural networks generally have much more capacity than shallow ones, as they have more layers and consequently more weights.

Counterintuitively, it is observed that the shallow model has many more parameters than the deep model. This is because a shallow model is heavily interconnected, which increases the number of parameters.

We can use the example of the laboratory where the shallow model has 10 times more parameters than the deep one.