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 :

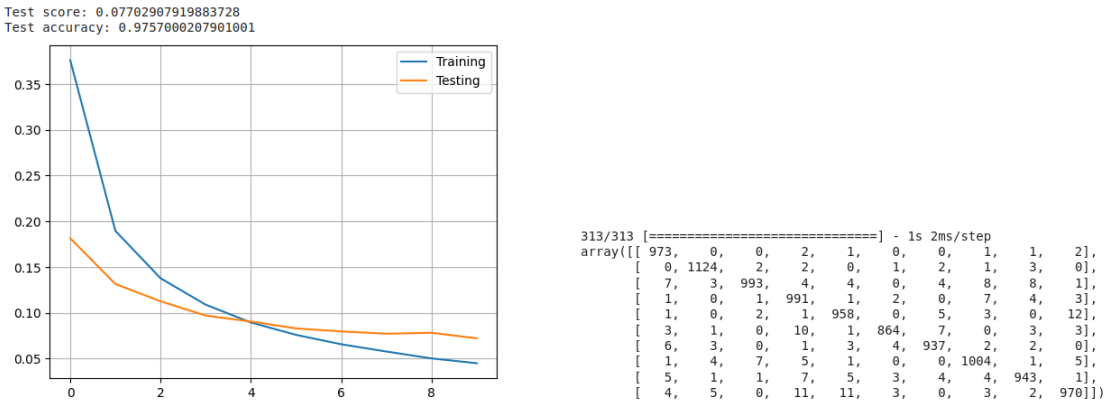
```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 l0 (InputLayer)             [(None, 28, 28, 1)]       0

 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

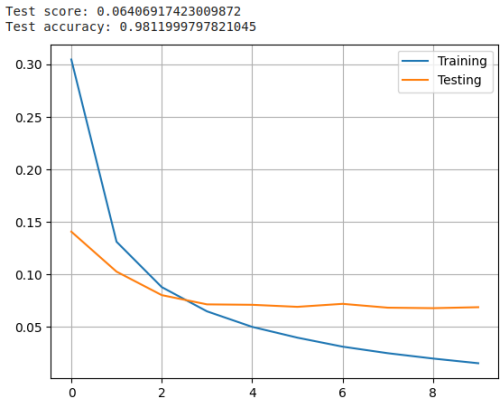
100 neurons



(a) Error graph **(b)** Confusion matrix

Figure 1: Model with raw data and 100 neurons in the dense layer

300 neurons



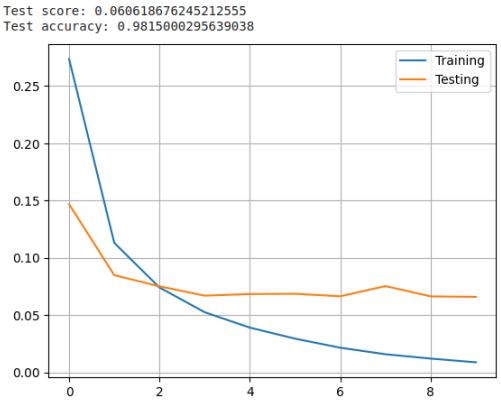
(a) Error graph

```
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]])
```

(b) Confusion matrix

Figure 2: Model with raw data and 300 neurons in the dense layer

600 neurones



(a) Error graph

```
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]])
```

(b) Confusion matrix

Figure 3: Model with raw data and 600 neurons in the dense layer

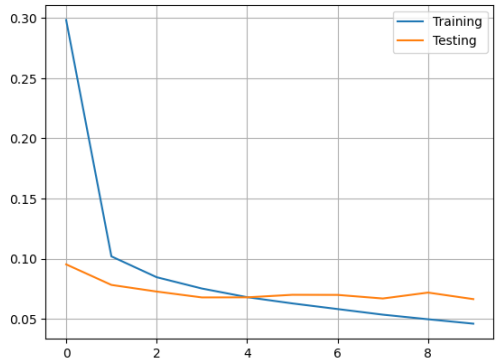
We can notice that the first model is indeed the most suitable for this situation. The more we increase the number of neurons, the more it overfits and the results aren't better.

An overall observation of the models is they often confuse the 9 with the 4. For the last two, they also have a tendency to confuse the 5 with the 3.

Features-based (HOG)

PIX_P_CELL 4, orientation 8, 100 neurons

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



(a) Error graph

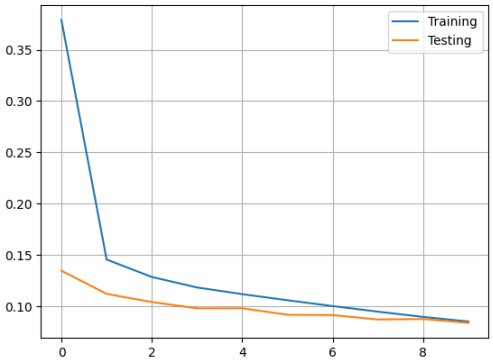
```
313/313 [==============================] - 1s 1ms/step
array([[ 971,    0,    1,    0,    0,    2,    3,    2,    0,    1],
       [   4, 1117,    1,    3,    1,    1,    4,    2,    2,    0],
       [   2,    1, 1015,    2,    2,    0,    2,    5,    3,    0],
       [   0,    1,    4,  986,    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]])
```

(b) Confusion matrix

Figure 4: Model using HOG features with 4 pixels per cell, 8 orientations and 100 neurons

PIX_P_CELL 4, orientation 8, 300 neurons

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



(a) Error graph

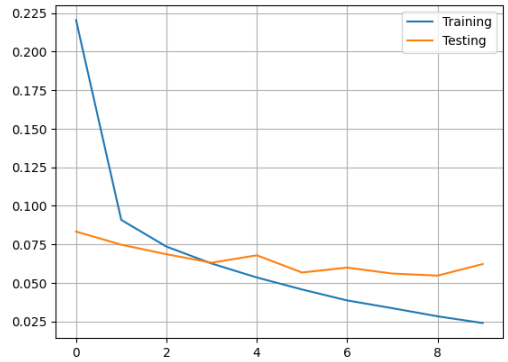
```
313/313 [==============================] - 0s 1ms/step
array([[ 966,    1,    2,    0,    0,    3,    4,    1,    3,    0],
       [   0, 1123,    3,    1,    0,    0,    4,    0,    4,    0],
       [   4,    7,  998,    4,    1,    0,    2,    8,    7,    1],
       [   0,    1,    2,  976,    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]])
```

(b) Confusion matrix

Figure 5: Model using HOG features with 4 pixels per cell, 4 orientations and 100 neurons

PIX_P_CELL 4, orientation 4, 100 neurons

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



(a) Error graph

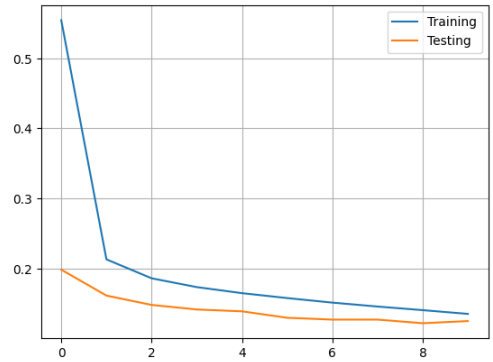
```
313/313 [==============================] - 0s 1ms/step
array([[ 972,    0,    1,    0,    0,    1,    3,    1,    1,    1],
       [   4, 1122,    1,    2,    0,    1,    2,    0,    3,    0],
       [   2,    4, 1017,    1,    1,    0,    2,    2,    3,    0],
       [   0,    1,    2,  987,    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]])
```

(b) Confusion matrix

Figure 6: Model using HOG features with 4 pixels per cell, 8 orientations and 300 neurons

PIX_P_CELL 7, orientation 8, 100 neurons

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



(a) Error graph

```
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]])
```

(b) Confusion matrix

Figure 7: Model using HOG features with 7 pixels per cell, 8 orientations and 100 neurons

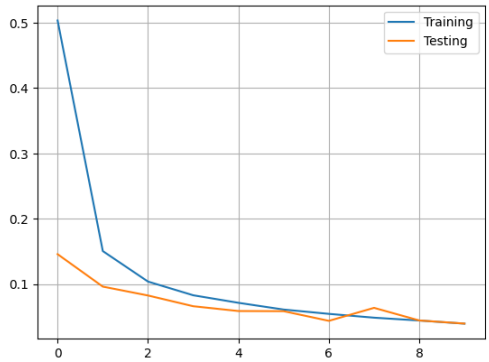
For us the best model is the seconde one. Compared to the others, it's one of the best in terms of performances and it doesn't seem to overfit. Models (1) and (3) are totally overfitting and the performances are not a lot better than (2). The last model is also very good, but the loss is higher than the second.

For this set of models, the number 9 is clearly the worse. It often is confused as other numbers (like 3 and 4) or other numbers are confused as a 9 (like 4, 7 and 8).

CNN

25 neurons

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



(a) Error graph

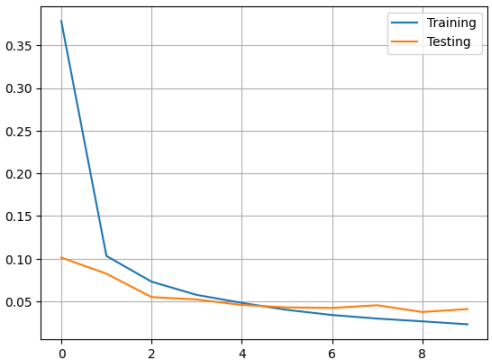
```
313/313 [==============================] - 1s 2ms/step
pred.shape =  (10000, 10)
array([[ 975,    0,    0,    0,    0,    0,    1,    1,    3,    0],
       [   1, 1127,    2,    0,    0,    2,    0,    1,    2,    0],
       [   3,    2, 1017,    2,    0,    0,    0,    7,    1,    0],
       [   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]])
```

(b) Confusion matrix

Figure 8: Model using CNN 25 neurons in the feed forward part

250 neurons

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



(a) Error graph

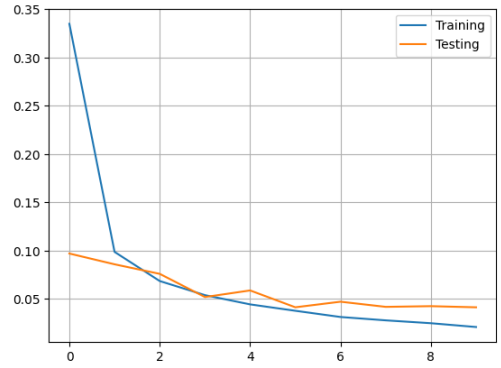
```
313/313 [==============================] - 1s 2ms/step
pred.shape =  (10000, 10)
array([[ 975,    0,    1,    0,    0,    0,    1,    0,    2,    1],
       [   1, 1131,    2,    0,    0,    0,    1,    0,    0,    0],
       [   2,    0, 1028,    0,    0,    0,    0,    1,    1,    0],
       [   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]])
```

(b) Confusion matrix

Figure 9: Model using CNN 250 neurons in the feed forward part

500 neurons

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



(a) Error graph

```
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]])
```

(b) Confusion matrix

Figure 10: Model using CNN 500 neurons in the feed forward part

Once again the model with less neurons is the best one. Not necessarily in the performances displayed by the loss function, it's slightly higher than the rest. But at least it doesn't overfit like the two models remaining.

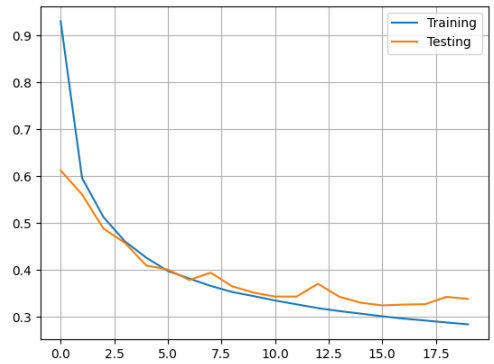
In this set, the confusion is more spread. We can't see a particular number with a lot of failures in all the models.

Custom model with convolutional deep neural networks

Train a CNN to solve the MNIST Fashion problem, present your evolution of the errors during training and perform a test. Present a confusion matrix, accuracy, F-score and discuss your results. Are there particular fashion categories that are frequently confused?

Experiments

```
Test score: 0.347442090511322
Test accuracy: 0.8781999945640564
```



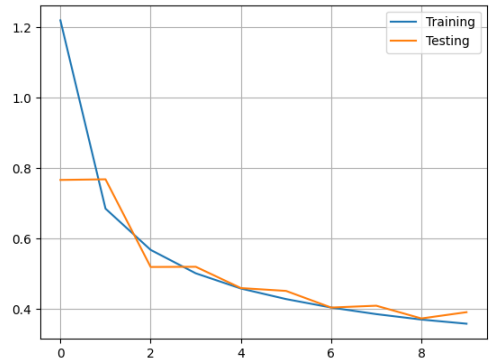
(a) Error graph

```
313/313 [==============================] - 1s 3ms/step
pred.shape =  (10000, 10)
F1-score : 0.8767111886301882
array([[857,   2,  19,  24,   6,   3,  68,   0,  21,   0],
       [  3, 959,   3,  27,   3,   0,   2,   0,   3,   0],
       [ 17,   0, 851,  12,  70,   0,  46,   0,   4,   0],
       [ 24,   6,  11, 897,  29,   0,  25,   0,   8,   0],
       [  1,   1,  88,  39, 817,   0,  50,   0,   4,   0],
       [  1,   0,   0,   1,   0, 925,   0,  65,   1,   7],
       [163,   1,  91,  31,  93,   0, 604,   0,  17,   0],
       [  0,   0,   0,   0,   0,   3,   0, 989,   0,   8],
       [  4,   1,   6,   3,   4,   4,   1,   8, 969,   0],
       [  0,   0,   1,   0,   0,   9,   0,  76,   0, 914]])
```

(b) Confusion matrix

Figure 11: Model using CNN 15 neurons in the feed forward part and 20 epochs

```
Test score: 0.40733426809310913
Test accuracy: 0.8501999974250793
```

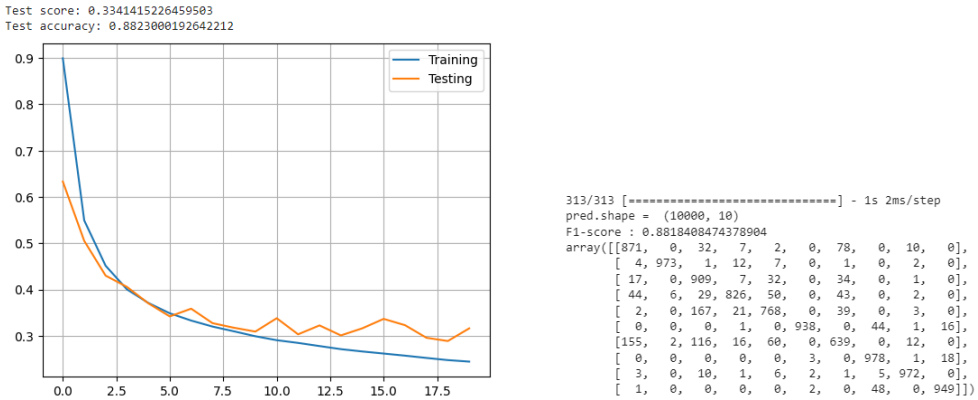


(a) Error graph

```
313/313 [==============================] - 1s 2ms/step
pred.shape =  (10000, 10)
F1-score : 0.8492636535421069
array([[674,   0,  40,  69,   5,   1, 202,   0,   9,   0],
       [  1, 952,   0,  33,   9,   0,   2,   0,   3,   0],
       [  4,   0, 870,  12,  63,   0,  45,   1,   5,   0],
       [  9,   3,  31, 899,  31,   0,  27,   0,   0,   0],
       [  0,   0, 148,  39, 773,   0,  32,   0,   8,   0],
       [  0,   0,   0,   3,   0, 917,   0,  61,   1,  18],
       [ 90,   0, 152,  46, 124,   0, 572,   1,  15,   0],
       [  0,   0,   0,   0,   0,  12,   0, 967,   0,  21],
       [  0,   1,  10,   6,  10,   1,  11,   7, 954,   0],
       [  0,   0,   1,   0,   0,   8,   0,  67,   0, 924]])
```

(b) Confusion matrix

Figure 12: Model using CNN 10 neurons in the feed forward part and 10 epochs



(a) Error graph **(b)** Confusion matrix

Figure 13: Model using CNN 30 neurons in the feed forward part and 20 epochs

We notice that all the models are not very good. The best one is probably the last because it didn't have time to start overfitting like the others. The performances is clearly not the best.

All the models tends to confuse tops (t-shirts, shirts, etc.).

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.