CNN Avec peu de données

Chargement du code

In [1]:
```
import numpy as np
from matplotlib import pyplot as plt
import tensorflow as tf

np.random.seed(123)  # for reproducibility
tf.random.set_seed(123)

%load_ext autoreload
%autoreload 2
```

In [2]:
```
from TP3_utils import load_dataset

DATA_PATH = 'Data'
CATEGORIES = ["accordion", "anchor", "barrel", "binocular"]

(X_train, Y_train), (X_test, Y_test), num_classes = load_dataset(DA
data_shape = X_train[0].shape

print("Training data shape:", data_shape)
print("Training labels shape:", Y_train.shape)
print("Test data shape:", X_test.shape)
print("Test labels shape:", Y_test.shape)
print("Number of classes:", num_classes)
```

```
Finished loading 177 images from 4 categories.
Train/Test split: 123 / 54
Training data shape: (224, 224, 3)
Training labels shape: (123, 4)
Test data shape: (54, 224, 224, 3)
Test labels shape: (54, 4)
Number of classes: 4
```


```
Finished loading 177 images from 4 categories.
Train/Test split: 123 / 54
Training data shape: (224, 224, 3)
Training labels shape: (123, 4)
Test data shape: (54, 224, 224, 3)
Test labels shape: (54, 4)
Number of classes: 4
```


Les images d'entrées sont redimensionnées à 224x224, et en couleurs (3 canaux pour RGB).

Il y a 4 classes en sortie, et la répartition est de 70% / 30% soit 123 examples d'entrainement et 54 de test sur les 177 chargés.

In [ ]:
```
import time
import tensorflow as tf
from tensorflow import keras
```

```python
from keras.optimizers import Adam
from TP3_utils import CNN, affiche, eval_classif, data_augmentation
from keras.callbacks import ReduceLROnPlateau, EarlyStopping

lr_scheduler = ReduceLROnPlateau(monitor='val_loss', factor=0.75, p
earlStop = EarlyStopping(monitor='val_loss', min_delta=1e-4, patien

lr=1e-4
batch_size=32
epochs=64
ad= Adam(learning_rate=lr)

model = CNN(data_shape)
model.summary()

plt.figure(figsize=(10, 4))
for i in range(5):
    augmented_img = data_augmentation(tf.expand_dims(X_train[0], 0)
    plt.subplot(1, 5, i+1)
    plt.imshow(augmented_img[0])
    plt.axis("off")
    plt.show()

model.compile(
    loss='categorical_crossentropy',
    optimizer=ad,
    metrics=['accuracy']
)

tps1 = time.time()
history =model.fit(
    X_train,
    Y_train,
    batch_size=batch_size,
    epochs=epochs,
    verbose=1,
    validation_data=(X_test, Y_test),
    callbacks=[earlStop, lr_scheduler]
)
tps2 = time.time()

# Evaluation
loss, accuracy = model.evaluate(X_test, Y_test)
print(f'Test loss: {loss}, Test accuracy: {accuracy}')

affiche(history)
preds = model.predict(X_test)
eval_classif(Y_test, preds)
print("Temps d'entraînement : {:.2f} secondes".format(tps2 - tps1))
```

Model: "functional_1"

Layer (type)	Output Shape	
input_layer (InputLayer)	(None, 224, 224, 3)	
sequential (Sequential)	(None, 224, 224, 3)	
conv2d (Conv2D)	(None, 224, 224, 16)	
max_pooling2d (MaxPooling2D)	(None, 112, 112, 16)	
conv2d_1 (Conv2D)	(None, 112, 112, 32)	
conv2d_2 (Conv2D)	(None, 112, 112, 32)	
max_pooling2d_1 (MaxPooling2D)	(None, 56, 56, 32)	
dropout (Dropout)	(None, 56, 56, 32)	
conv2d_3 (Conv2D)	(None, 56, 56, 64)	
conv2d_4 (Conv2D)	(None, 56, 56, 64)	
dropout_1 (Dropout)	(None, 56, 56, 64)	
flatten (Flatten)	(None, 200704)	
dense (Dense)	(None, 100)	
dropout_2 (Dropout)	(None, 100)	
dense_1 (Dense)	(None, 4)	

Total params: 20,140,664 (76.83 MB)
Trainable params: 20,140,664 (76.83 MB)
Non-trainable params: 0 (0.00 B)

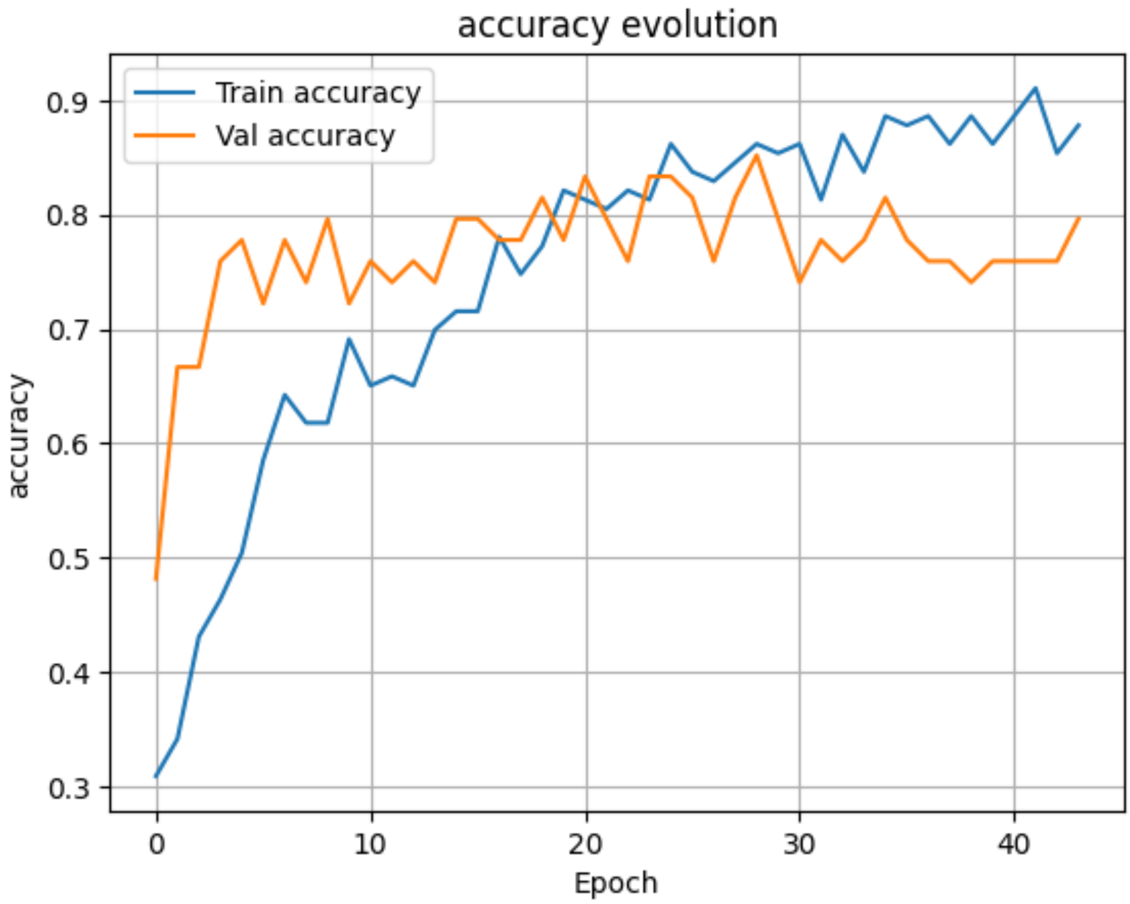
```
Epoch 1/64
4/4 ━━━━━━━━━━━━━━━━━━━━ 3s 497ms/step - accuracy: 0.3089 - loss: 1.
4556 - val_accuracy: 0.4815 - val_loss: 1.3045 - learning_rate: 1.00
00e-04
Epoch 2/64
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 445ms/step - accuracy: 0.3415 - loss: 1.
3691 - val_accuracy: 0.6667 - val_loss: 1.1482 - learning_rate: 1.00
00e-04
Epoch 3/64
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 432ms/step - accuracy: 0.4309 - loss: 1.
2772 - val_accuracy: 0.6667 - val_loss: 1.0795 - learning_rate: 1.00
00e-04
Epoch 4/64
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 427ms/step - accuracy: 0.4634 - loss: 1.
2088 - val_accuracy: 0.7593 - val_loss: 0.9737 - learning_rate: 1.00
00e-04
Epoch 5/64
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 426ms/step - accuracy: 0.5041 - loss: 1.
1355 - val_accuracy: 0.7778 - val_loss: 0.8628 - learning_rate: 1.00
00e-04
Epoch 6/64
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 427ms/step - accuracy: 0.5854 - loss: 1.
0419 - val_accuracy: 0.7222 - val_loss: 0.7416 - learning_rate: 1.00
00e-04
Epoch 7/64
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 428ms/step - accuracy: 0.6423 - loss: 1.
0069 - val_accuracy: 0.7778 - val_loss: 0.6771 - learning_rate: 1.00
00e-04
Epoch 8/64
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 429ms/step - accuracy: 0.6179 - loss: 0.
9297 - val_accuracy: 0.7407 - val_loss: 0.6265 - learning_rate: 1.00
00e-04
Epoch 9/64
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 426ms/step - accuracy: 0.6179 - loss: 0.
9272 - val_accuracy: 0.7963 - val_loss: 0.6031 - learning_rate: 1.00
00e-04
Epoch 10/64
```

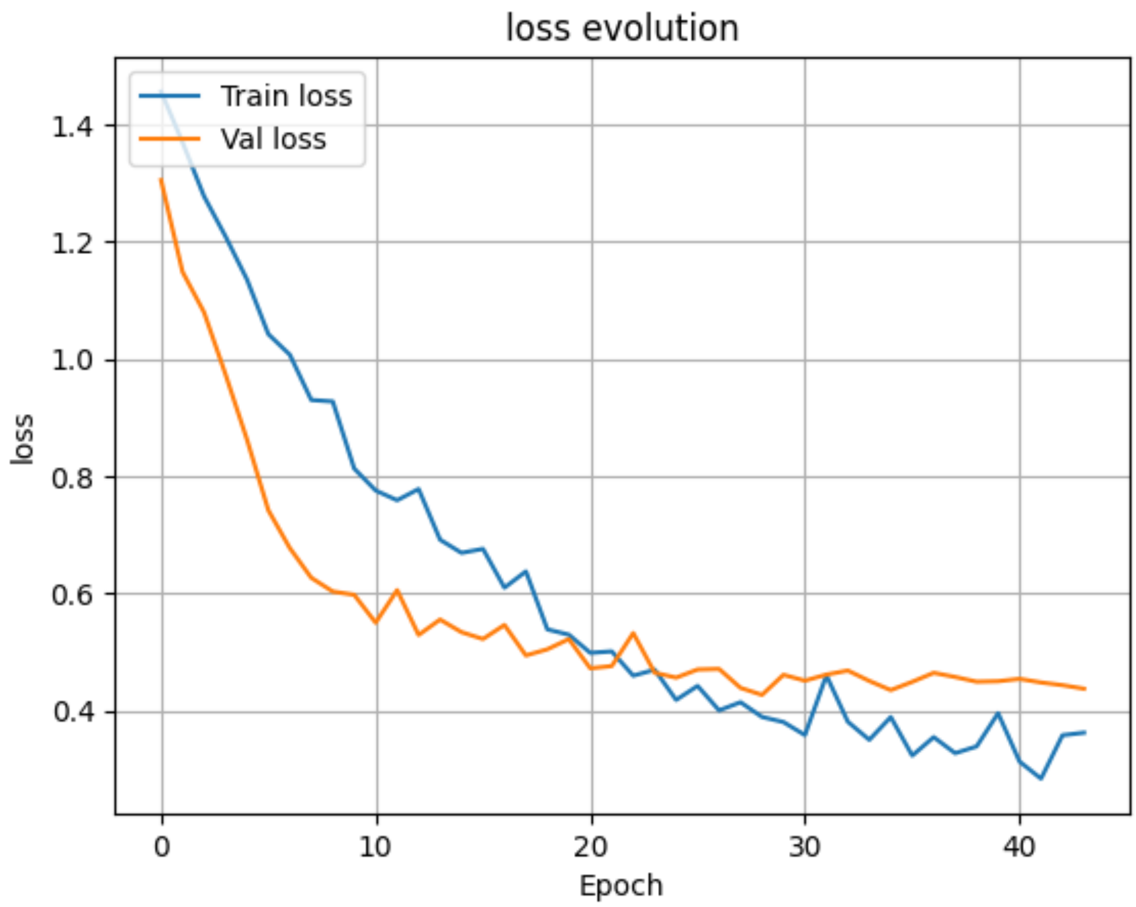
```
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 426ms/step - accuracy: 0.6911 - loss: 0.
8128 - val_accuracy: 0.7222 - val_loss: 0.5975 - learning_rate: 1.00
00e-04
Epoch 11/64
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 426ms/step - accuracy: 0.6504 - loss: 0.
7754 - val_accuracy: 0.7593 - val_loss: 0.5500 - learning_rate: 1.00
00e-04
Epoch 12/64
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 438ms/step - accuracy: 0.6585 - loss: 0.
7587 - val_accuracy: 0.7407 - val_loss: 0.6055 - learning_rate: 1.00
00e-04
Epoch 13/64
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 429ms/step - accuracy: 0.6504 - loss: 0.
7779 - val_accuracy: 0.7593 - val_loss: 0.5290 - learning_rate: 1.00
00e-04
Epoch 14/64
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 438ms/step - accuracy: 0.6992 - loss: 0.
6915 - val_accuracy: 0.7407 - val_loss: 0.5553 - learning_rate: 1.00
00e-04
Epoch 15/64
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 370ms/step - accuracy: 0.6880 - loss: 0.
7282
Epoch 15: ReduceLROnPlateau reducing learning rate to 7.499999810534
064e-05.
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 428ms/step - accuracy: 0.7154 - loss: 0.
6690 - val_accuracy: 0.7963 - val_loss: 0.5338 - learning_rate: 1.00
00e-04
Epoch 16/64
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 428ms/step - accuracy: 0.7154 - loss: 0.
6758 - val_accuracy: 0.7963 - val_loss: 0.5221 - learning_rate: 7.50
00e-05
Epoch 17/64
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 427ms/step - accuracy: 0.7805 - loss: 0.
6097 - val_accuracy: 0.7778 - val_loss: 0.5461 - learning_rate: 7.50
00e-05
Epoch 18/64
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 429ms/step - accuracy: 0.7480 - loss: 0.
6374 - val_accuracy: 0.7778 - val_loss: 0.4941 - learning_rate: 7.50
00e-05
Epoch 19/64
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 426ms/step - accuracy: 0.7724 - loss: 0.
5386 - val_accuracy: 0.8148 - val_loss: 0.5046 - learning_rate: 7.50
00e-05
Epoch 20/64
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 372ms/step - accuracy: 0.7925 - loss: 0.
5780
Epoch 20: ReduceLROnPlateau reducing learning rate to 5.624999721476
3425e-05.
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 430ms/step - accuracy: 0.8211 - loss: 0.
5295 - val_accuracy: 0.7778 - val_loss: 0.5220 - learning_rate: 7.50
00e-05
Epoch 21/64
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 427ms/step - accuracy: 0.8130 - loss: 0.
4985 - val_accuracy: 0.8333 - val_loss: 0.4722 - learning_rate: 5.62
50e-05
Epoch 22/64
```

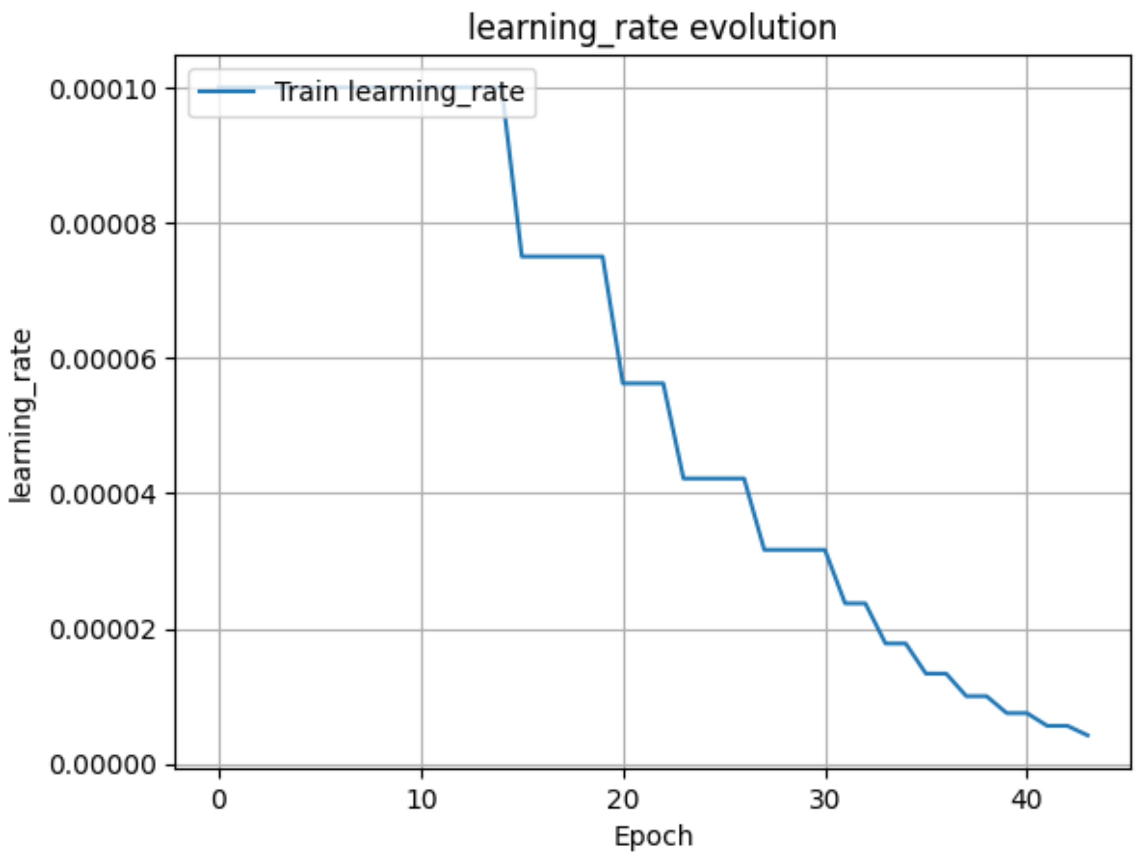
```
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 427ms/step - accuracy: 0.8049 - loss: 0.
5010 - val_accuracy: 0.7963 - val_loss: 0.4760 - learning_rate: 5.62
50e-05
Epoch 23/64
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 379ms/step - accuracy: 0.8212 - loss: 0.
4520
Epoch 23: ReduceLROnPlateau reducing learning rate to 4.218749927531
462e-05.
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 437ms/step - accuracy: 0.8211 - loss: 0.
4596 - val_accuracy: 0.7593 - val_loss: 0.5322 - learning_rate: 5.62
50e-05
Epoch 24/64
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 431ms/step - accuracy: 0.8130 - loss: 0.
4697 - val_accuracy: 0.8333 - val_loss: 0.4646 - learning_rate: 4.21
88e-05
Epoch 25/64
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 426ms/step - accuracy: 0.8618 - loss: 0.
4183 - val_accuracy: 0.8333 - val_loss: 0.4563 - learning_rate: 4.21
88e-05
Epoch 26/64
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 428ms/step - accuracy: 0.8374 - loss: 0.
4422 - val_accuracy: 0.8148 - val_loss: 0.4702 - learning_rate: 4.21
88e-05
Epoch 27/64
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 370ms/step - accuracy: 0.8219 - loss: 0.
3739
Epoch 27: ReduceLROnPlateau reducing learning rate to 3.164062582072
802e-05.
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 428ms/step - accuracy: 0.8293 - loss: 0.
4005 - val_accuracy: 0.7593 - val_loss: 0.4711 - learning_rate: 4.21
88e-05
Epoch 28/64
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 428ms/step - accuracy: 0.8455 - loss: 0.
4143 - val_accuracy: 0.8148 - val_loss: 0.4388 - learning_rate: 3.16
41e-05
Epoch 29/64
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 431ms/step - accuracy: 0.8618 - loss: 0.
3893 - val_accuracy: 0.8519 - val_loss: 0.4269 - learning_rate: 3.16
41e-05
Epoch 30/64
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 426ms/step - accuracy: 0.8537 - loss: 0.
3806 - val_accuracy: 0.7963 - val_loss: 0.4611 - learning_rate: 3.16
41e-05
Epoch 31/64
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 369ms/step - accuracy: 0.8274 - loss: 0.
3692
Epoch 31: ReduceLROnPlateau reducing learning rate to 2.373046936554
6014e-05.
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 427ms/step - accuracy: 0.8618 - loss: 0.
3584 - val_accuracy: 0.7407 - val_loss: 0.4511 - learning_rate: 3.16
41e-05
Epoch 32/64
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 431ms/step - accuracy: 0.8130 - loss: 0.
4602 - val_accuracy: 0.7778 - val_loss: 0.4614 - learning_rate: 2.37
30e-05
Epoch 33/64
```

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 371ms/step - accuracy: 0.8998 - loss: 0.
3796
Epoch 33: ReduceLROnPlateau reducing learning rate to 1.779785270628
0537e-05.
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 438ms/step - accuracy: 0.8699 - loss: 0.
3810 - val_accuracy: 0.7593 - val_loss: 0.4684 - learning_rate: 2.37
30e-05
Epoch 34/64
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 428ms/step - accuracy: 0.8374 - loss: 0.
3503 - val_accuracy: 0.7778 - val_loss: 0.4507 - learning_rate: 1.77
98e-05
Epoch 35/64
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 373ms/step - accuracy: 0.8986 - loss: 0.
4195
Epoch 35: ReduceLROnPlateau reducing learning rate to 1.334838952971
0403e-05.
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 431ms/step - accuracy: 0.8862 - loss: 0.
3891 - val_accuracy: 0.8148 - val_loss: 0.4350 - learning_rate: 1.77
98e-05
Epoch 36/64
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 428ms/step - accuracy: 0.8780 - loss: 0.
3234 - val_accuracy: 0.7778 - val_loss: 0.4491 - learning_rate: 1.33
48e-05
Epoch 37/64
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 371ms/step - accuracy: 0.8921 - loss: 0.
3341
Epoch 37: ReduceLROnPlateau reducing learning rate to 1.001129248834
3315e-05.
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 429ms/step - accuracy: 0.8862 - loss: 0.
3549 - val_accuracy: 0.7593 - val_loss: 0.4645 - learning_rate: 1.33
48e-05
Epoch 38/64
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 430ms/step - accuracy: 0.8618 - loss: 0.
3275 - val_accuracy: 0.7593 - val_loss: 0.4574 - learning_rate: 1.00
11e-05
Epoch 39/64
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 370ms/step - accuracy: 0.8986 - loss: 0.
3300
Epoch 39: ReduceLROnPlateau reducing learning rate to 7.508469025196
973e-06.
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 429ms/step - accuracy: 0.8862 - loss: 0.
3390 - val_accuracy: 0.7407 - val_loss: 0.4494 - learning_rate: 1.00
11e-05
Epoch 40/64
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 429ms/step - accuracy: 0.8618 - loss: 0.
3956 - val_accuracy: 0.7593 - val_loss: 0.4502 - learning_rate: 7.50
85e-06
Epoch 41/64
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 370ms/step - accuracy: 0.8713 - loss: 0.
3323
Epoch 41: ReduceLROnPlateau reducing learning rate to 5.631351768897
73e-06.
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 428ms/step - accuracy: 0.8862 - loss: 0.
3132 - val_accuracy: 0.7593 - val_loss: 0.4542 - learning_rate: 7.50
85e-06
Epoch 42/64
```

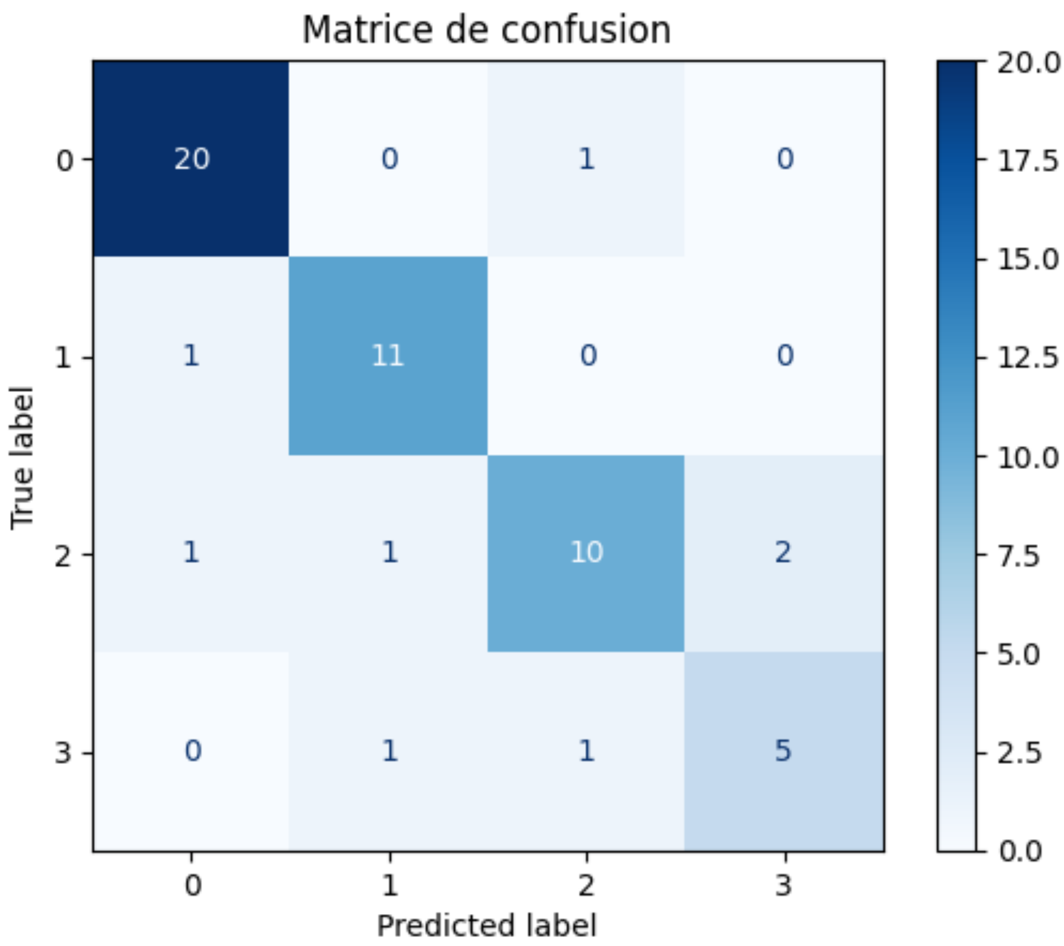
```
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 428ms/step - accuracy: 0.9106 - loss: 0.
2840 - val_accuracy: 0.7593 - val_loss: 0.4480 - learning_rate: 5.63
14e-06
Epoch 43/64
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 370ms/step - accuracy: 0.8111 - loss: 0.
4033
Epoch 43: ReduceLROnPlateau reducing learning rate to 4.223513997203
554e-06.
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 438ms/step - accuracy: 0.8537 - loss: 0.
3581 - val_accuracy: 0.7593 - val_loss: 0.4434 - learning_rate: 5.63
14e-06
Epoch 44/64
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 430ms/step - accuracy: 0.8780 - loss: 0.
3624 - val_accuracy: 0.7963 - val_loss: 0.4374 - learning_rate: 4.22
35e-06
Epoch 44: early stopping
Restoring model weights from the end of the best epoch: 29.
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 73ms/step - accuracy: 0.8519 - loss: 0.4
269
Test loss: 0.42687469720840454, Test accuracy: 0.8518518805503845
```








```
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 101ms/step
```




```
Classification report:
              precision    recall  f1-score   support

           0       0.91      0.95      0.93        21
           1       0.85      0.92      0.88        12
           2       0.83      0.71      0.77        14
           3       0.71      0.71      0.71         7

    accuracy                           0.85        54
   macro avg       0.83      0.82      0.82        54
weighted avg       0.85      0.85      0.85        54

Temps d'entraînement : 75.47 secondes
```

Le modèle initial à **20,090,296** paramètres entrainable. Après un entrainement de **19.32s**, le modèle atteint **loss=0.8; accuracy=66.66%**

Pour définir la structure du modèle, je fixe:

- lr=1e-4
- batch_size=16
- epochs=16
- optimizer=Adam(lr)
- MLP finale: Dense(100)

Puis, je procède de la façon suivante:

itération	nb étages	taille des filtres	nb params	tps d'entrainement	loss	accuracy
1	2	16-32	40,146,392	23.37s	0.7255	72.22%
2	3	16-32-64	20,094,488	19.03s	0.5276	77.77%
3	4	16-32-64-128	10,133,144	18.83s	0.8526	72.22%
4	5	16-32-64-128-256	5,410,712	20.09s	0.9057	70.37%
5	3	32-64-128	40,234,552	47.86s	0.8762	72.22%
6	4	32-64-128-256	20,459,320	44.29s	0.9546	75.92%
7	5	32-64-128-256-512	11,604,280	51.48s	0.7749	74.07%
8	2	8-16	20,072,296	13.89s	0.7267	66.66%
9	3	8-16-32	10,041,736	12.12s	0.7994	68.51%

Pour les N premiers, overfitting très rapide; aucun réel apprentissage après quelques epochs seulements. La majorité des paramètres sont à la couche dense après le flatten, donc augmenter le nombre de couche permet de faire intervenir plus de max pooling, donc réduire le temps d'apprentissage tout en ayant une meilleure répartition des paramètres le long du réseau. Cependant, un réseau trop long réduit trop le nombre de paramètres, donc la capacité d'expression du réseau? Ou bien est-ce un souci d'evanescence du gradient? On peut alors tenter d'augmenter le nombre de filtres, mais on remarque que cela n'améliore pas la précision et ralentit largment l'apprentissage. On constate nettement des signe d'overfitting.
Au contraire, en tentant de réduire le nombre de filtre, on réduit le temps d'apprentissage mais là encore, la précision diminue.

Le meilleur compromis semble alors être d'utiliser 3 étages, avec respectivement 16, 32 et 64 filtres. Pour enrichir le réseau, on peut maintenant essayer d'augmenter le nombre de couches par étages.

itération	nb couches/ étage	nb params	tps d'entrainement (total)	loss	best val_accuracy
0	1-1-1	20,094,488	19.03s	0.5276	77.77%
1	2-1-1	20,096,808	28.84s	0.8320	75.92%
2	1-2-1	20,103,736	25.42s	0.9896	72.22%
3	1-1-2	20,131,416	23.72s	0.9335	64.81%
4	2-2-1	20,106,056	34.36s	0.5813	74.07%
5	2-1-2	20,133,736	34.46s	0.6585	72.22%
6	1-2-2	20,140,664	31.39s	0.7302	79.62%
7	2-2-2	20,142,984	40.51s	0.7089	75.92%

Bien que cette fois, l'impact est moins clair, la configuration qui semble la meilleure est 1-2-2 (bien que 79% semble avoir été un outlier, cette configuration donne de façon plutôt consistante >73% ce qui n'est pas le cas des autres). Ce que j'utiliserai par la suite. Cependant on remarque un fort overfitting. Pour palier cela, nous ajouterons du dropout pour les différentes layers, puis de la data augmentation.

La transformation qui me semble la plus efficace ici est une rotation légère. Aussi, avec data augmentation et dropout, l'entrainement était très instable, alors j'ai augmenté le batch_size. Malgrès tout, le modèle était très sensible et je n'ai pas pu mettre beaucoup de dropout, surtout sur les couches hautes. En m'aidant d'early stopping et d'un lr scheduler, j'ai atteint au mieux ces résultats:

```
Classification report:
              precision    recall  f1-score   support

           0       0.91      0.95      0.93        21
           1       0.85      0.92      0.88        12
           2       0.83      0.71      0.77        14
           3       0.71      0.71      0.71         7

    accuracy                           0.85        54
   macro avg       0.83      0.82      0.82        54
weighted avg       0.85      0.85      0.85        54

Temps d'entraînement : 75.47 secondes
```

In [9]:
```
from tensorflow.keras.layers import Input, Flatten
from tensorflow.keras.models import Model
from TP3_utils import MLP_transfer

lr=1e-4
batch_size=16
```

```python
epochs=16
ad=Adam(learning_rate=lr)

# Transfer learning
input_tensor = Input(shape=(224,224,3))
VGG = tf.keras.applications.VGG16(
    weights='imagenet',
    include_top=False,
    input_tensor=input_tensor
)
for layer in VGG.layers:
    layer.trainable = False

x = VGG.output
x = Flatten()(x)
x = MLP_transfer(x, num_classes)
model_transfer = Model(inputs=VGG.input, outputs=x)

model_transfer.summary()

model_transfer.compile(
    loss='categorical_crossentropy',
    optimizer=Adam(1e-5),
    metrics=['accuracy']
)

tps1 = time.time()
history =model_transfer.fit(
    X_train,
    Y_train,
    batch_size=batch_size,
    epochs=epochs,
    verbose=1,
    validation_data=(X_test, Y_test)
#     callbacks=[earlStopBis, lr_schedulerBis],
)
tps2 = time.time()

loss, accuracy = model_transfer.evaluate(X_test, Y_test)
print(f'Test loss: {loss}, Test accuracy: {accuracy}')

affiche(history)
preds = model_transfer.predict(X_test)
eval_classif(Y_test, preds)
print("Temps d'entraînement : {:.2f} secondes".format(tps2 - tps1))
```

Model: "functional_8"

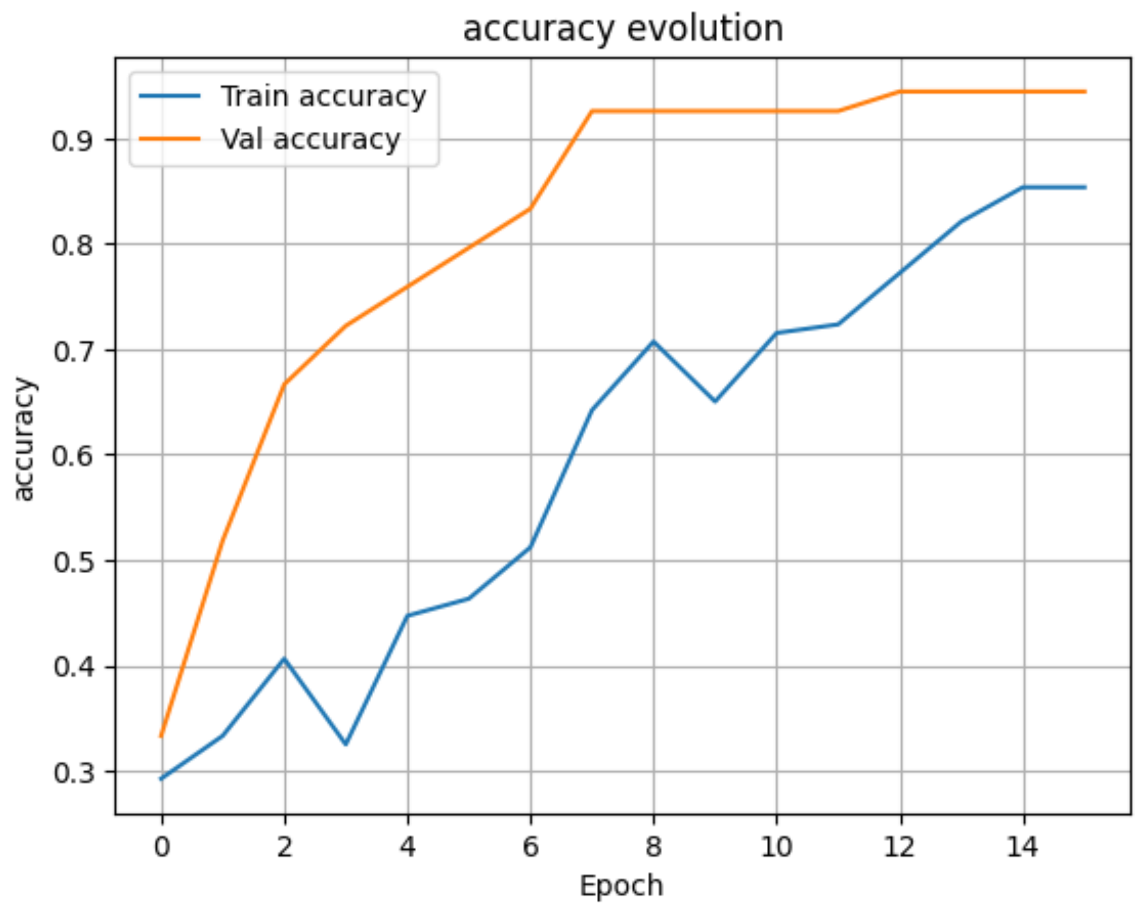
Layer (type)	Output Shape	
input_layer_13 (InputLayer)	(None, 224, 224, 3)	
block1_conv1 (Conv2D)	(None, 224, 224, 64)	
block1_conv2 (Conv2D)	(None, 224, 224, 64)	
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	
block2_conv1 (Conv2D)	(None, 112, 112, 128)	
block2_conv2 (Conv2D)	(None, 112, 112, 128)	
block2_pool (MaxPooling2D)	(None, 56, 56, 128)	
block3_conv1 (Conv2D)	(None, 56, 56, 256)	
block3_conv2 (Conv2D)	(None, 56, 56, 256)	
block3_conv3 (Conv2D)	(None, 56, 56, 256)	
block3_pool (MaxPooling2D)	(None, 28, 28, 256)	
block4_conv1 (Conv2D)	(None, 28, 28, 512)	
block4_conv2 (Conv2D)	(None, 28, 28, 512)	
block4_conv3 (Conv2D)	(None, 28, 28, 512)	
block4_pool (MaxPooling2D)	(None, 14, 14, 512)	
block5_conv1 (Conv2D)	(None, 14, 14, 512)	
block5_conv2 (Conv2D)	(None, 14, 14, 512)	
block5_conv3 (Conv2D)	(None, 14, 14, 512)	
block5_pool (MaxPooling2D)	(None, 7, 7, 512)	
flatten_8 (Flatten)	(None, 25088)	
dense_11 (Dense)	(None, 256)	
dropout_3 (Dropout)	(None, 256)	
dense_12 (Dense)	(None, 128)	
dropout_4 (Dropout)	(None, 128)	
dense_13 (Dense)	(None, 4)	

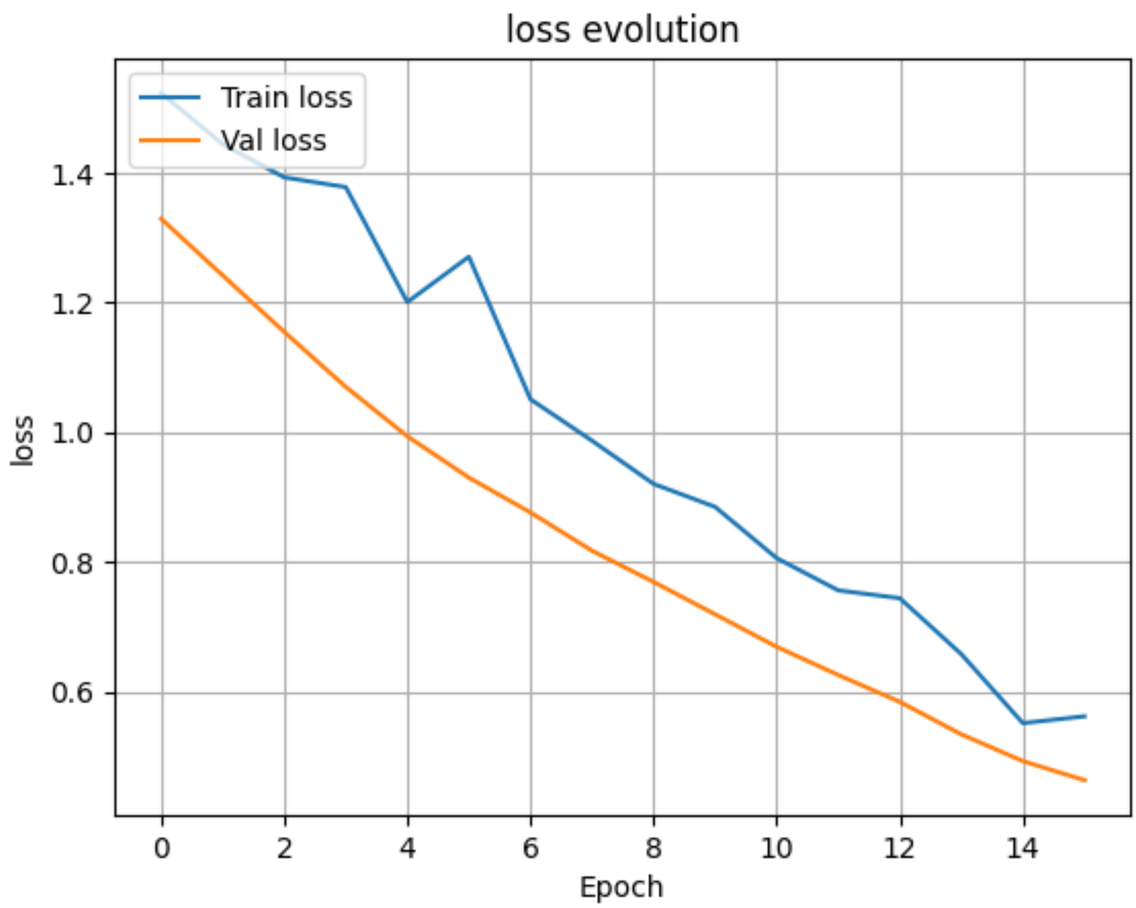
Total params: 21,170,884 (80.76 MB)

Trainable params: 6,456,196 (24.63 MB)

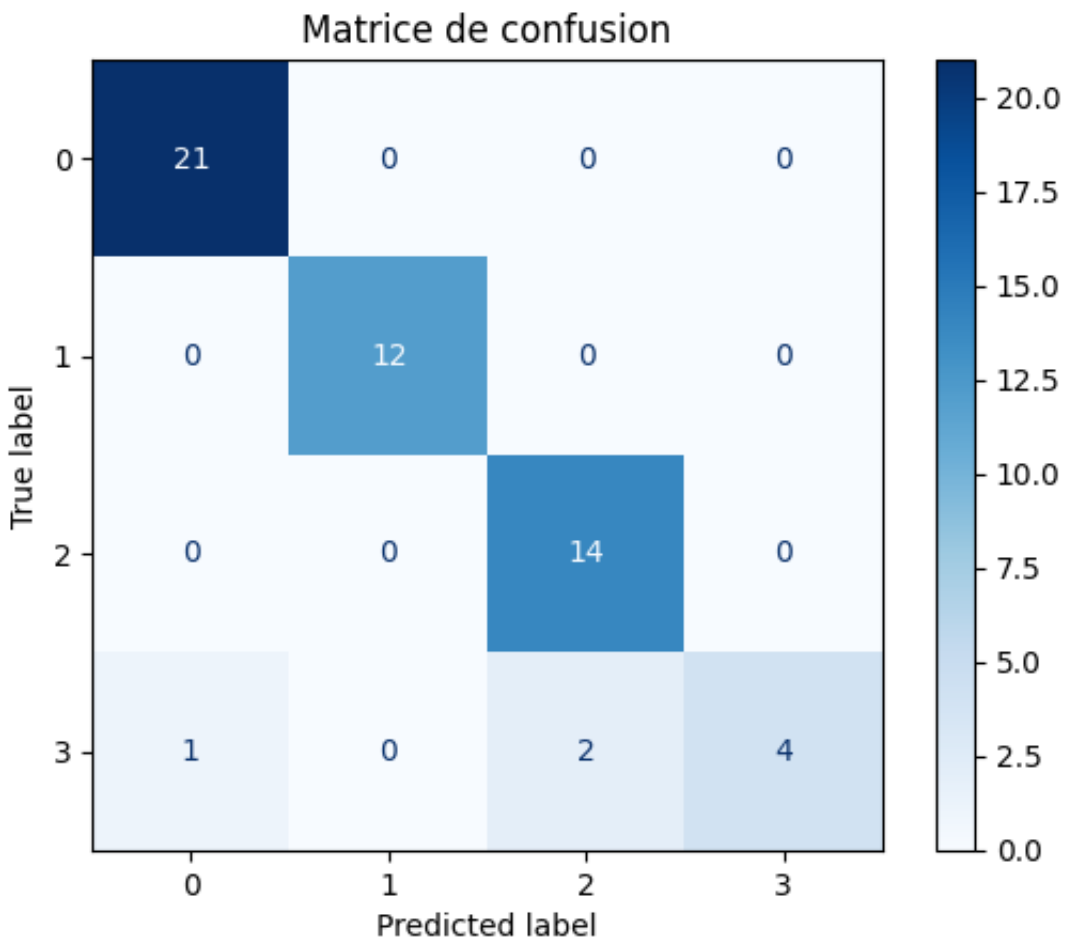
Non-trainable params: 14,714,688 (56.13 MB)

```
Epoch 1/16
8/8 ━━━━━━━━━━━━━━━━━━━━ 16s 2s/step - accuracy: 0.2927 - loss: 1.52
28 - val_accuracy: 0.3333 - val_loss: 1.3289
Epoch 2/16
8/8 ━━━━━━━━━━━━━━━━━━━━ 15s 2s/step - accuracy: 0.3333 - loss: 1.44
40 - val_accuracy: 0.5185 - val_loss: 1.2417
Epoch 3/16
8/8 ━━━━━━━━━━━━━━━━━━━━ 14s 2s/step - accuracy: 0.4065 - loss: 1.39
28 - val_accuracy: 0.6667 - val_loss: 1.1546
Epoch 4/16
8/8 ━━━━━━━━━━━━━━━━━━━━ 15s 2s/step - accuracy: 0.3252 - loss: 1.37
78 - val_accuracy: 0.7222 - val_loss: 1.0697
Epoch 5/16
8/8 ━━━━━━━━━━━━━━━━━━━━ 16s 2s/step - accuracy: 0.4472 - loss: 1.20
05 - val_accuracy: 0.7593 - val_loss: 0.9936
Epoch 6/16
8/8 ━━━━━━━━━━━━━━━━━━━━ 15s 2s/step - accuracy: 0.4634 - loss: 1.27
04 - val_accuracy: 0.7963 - val_loss: 0.9298
Epoch 7/16
8/8 ━━━━━━━━━━━━━━━━━━━━ 16s 2s/step - accuracy: 0.5122 - loss: 1.05
07 - val_accuracy: 0.8333 - val_loss: 0.8760
Epoch 8/16
8/8 ━━━━━━━━━━━━━━━━━━━━ 16s 2s/step - accuracy: 0.6423 - loss: 0.98
67 - val_accuracy: 0.9259 - val_loss: 0.8171
Epoch 9/16
8/8 ━━━━━━━━━━━━━━━━━━━━ 16s 2s/step - accuracy: 0.7073 - loss: 0.92
01 - val_accuracy: 0.9259 - val_loss: 0.7690
Epoch 10/16
8/8 ━━━━━━━━━━━━━━━━━━━━ 16s 2s/step - accuracy: 0.6504 - loss: 0.88
47 - val_accuracy: 0.9259 - val_loss: 0.7190
Epoch 11/16
8/8 ━━━━━━━━━━━━━━━━━━━━ 15s 2s/step - accuracy: 0.7154 - loss: 0.80
57 - val_accuracy: 0.9259 - val_loss: 0.6689
Epoch 12/16
8/8 ━━━━━━━━━━━━━━━━━━━━ 15s 2s/step - accuracy: 0.7236 - loss: 0.75
59 - val_accuracy: 0.9259 - val_loss: 0.6255
Epoch 13/16
8/8 ━━━━━━━━━━━━━━━━━━━━ 15s 2s/step - accuracy: 0.7724 - loss: 0.74
39 - val_accuracy: 0.9444 - val_loss: 0.5841
Epoch 14/16
8/8 ━━━━━━━━━━━━━━━━━━━━ 15s 2s/step - accuracy: 0.8211 - loss: 0.65
76 - val_accuracy: 0.9444 - val_loss: 0.5339
Epoch 15/16
8/8 ━━━━━━━━━━━━━━━━━━━━ 15s 2s/step - accuracy: 0.8537 - loss: 0.55
12 - val_accuracy: 0.9444 - val_loss: 0.4927
Epoch 16/16
8/8 ━━━━━━━━━━━━━━━━━━━━ 15s 2s/step - accuracy: 0.8537 - loss: 0.56
17 - val_accuracy: 0.9444 - val_loss: 0.4633
2/2 ━━━━━━━━━━━━━━━━━━━━ 5s 2s/step - accuracy: 0.9444 - loss: 0.463
3
Test loss: 0.46325498819351196, Test accuracy: 0.9444444179534912
```


accuracy evolution
Train accuracy
Val accuracy
accuracy
0.3
0.4
0.5
0.6
0.7
0.8
0.9
0
2
4
6
8
10
12
14
Epoch


loss evolution
Train loss
Val loss
loss
0.6
0.8
1.0
1.2
1.4
0
2
4
6
8
10
12
14
Epoch

2/2 ━━━━━━━━━━━━━━━━━━━━ **6s** 2s/step



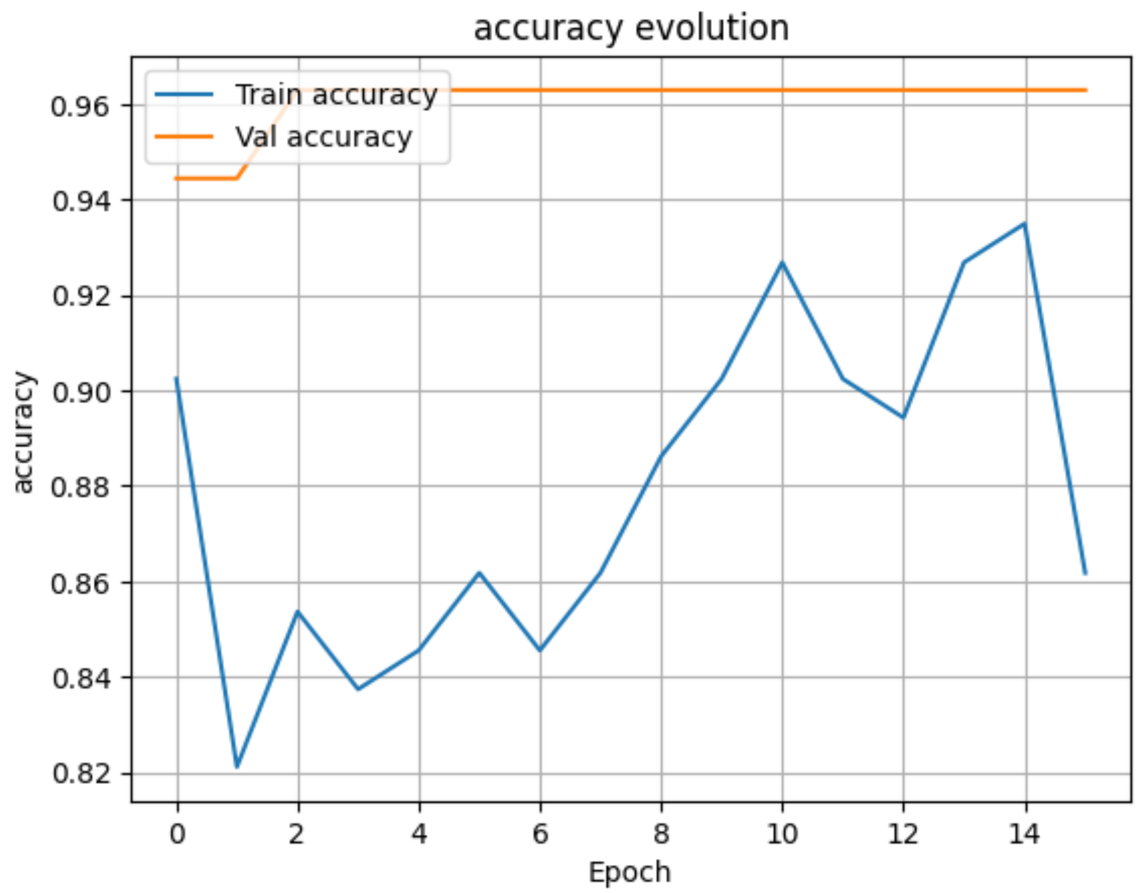

```
Classification report:
              precision    recall  f1-score   support

           0       0.95      1.00      0.98        21
           1       1.00      1.00      1.00        12
           2       0.88      1.00      0.93        14
           3       1.00      0.57      0.73         7

    accuracy                           0.94        54
   macro avg       0.96      0.89      0.91        54
weighted avg       0.95      0.94      0.94        54

Temps d'entraînement : 243.76 secondes
```

On obtiens très facilement des performances nettement meilleures qu'avant:

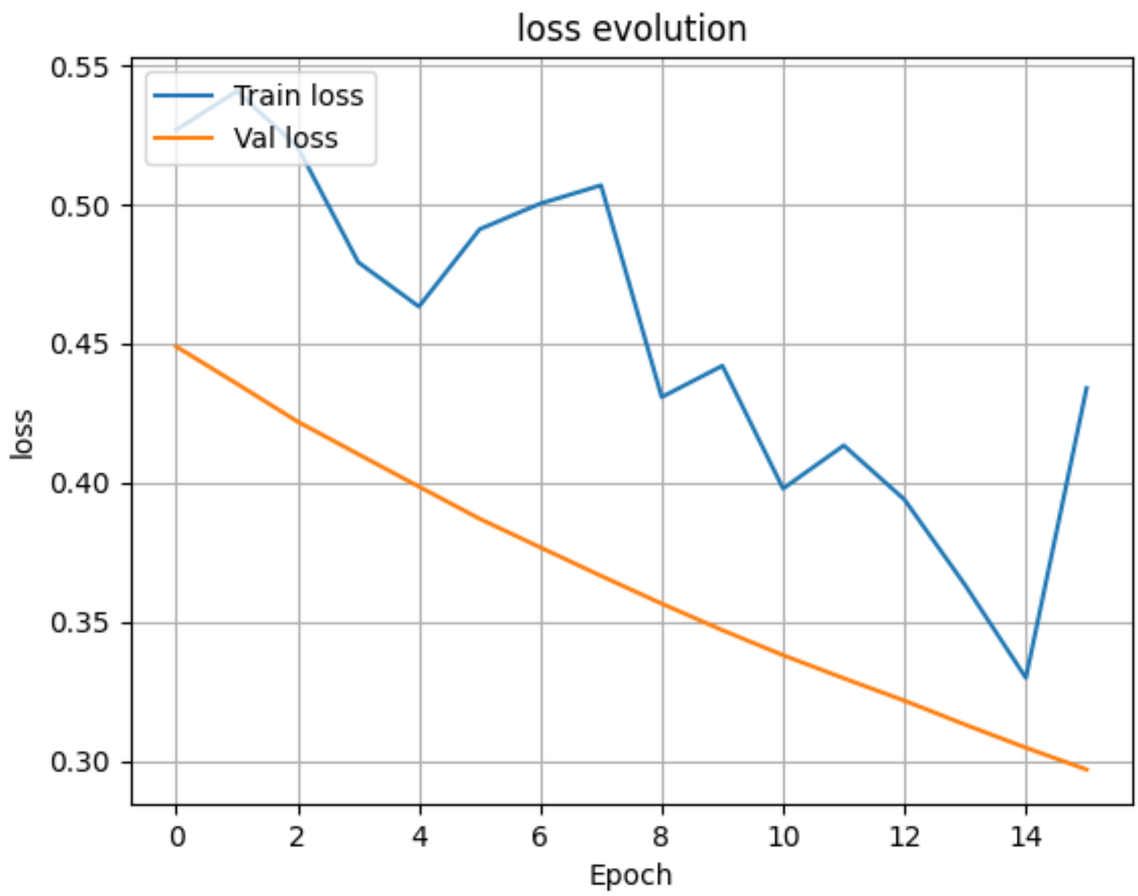

```
Classification report:
              precision    recall  f1-score   support

           0       0.95      1.00      0.98        21
           1       1.00      1.00      1.00        12
           2       0.88      1.00      0.93        14
           3       1.00      0.57      0.73         7

    accuracy                           0.94        54
   macro avg       0.96      0.89      0.91        54
weighted avg       0.95      0.94      0.94        54

Temps d'entraînement : 243.76 secondes
```

In [10]:
```
################## Fine-tuning #################

for layer in VGG.layers[-4:]:
    layer.trainable = True

model_transfer.compile(
    loss='categorical_crossentropy',
    optimizer=Adam(1e-6),
    metrics=['accuracy']
)

tps1 = time.time()
history =model_transfer.fit(
    X_train,
    Y_train,
    batch_size=batch_size,
    epochs=epochs,
    verbose=1,
    validation_data=(X_test, Y_test)
#     callbacks=[earlStopBis, lr_schedulerBis],
)
tps2 = time.time()

loss, accuracy = model_transfer.evaluate(X_test, Y_test)
print(f'Test loss: {loss}, Test accuracy: {accuracy}')

affiche(history)
preds = model_transfer.predict(X_test)
eval_classif(Y_test, preds)
print("Temps d'entraînement : {:.2f} secondes".format(tps2 - tps1))
```

```
Epoch 1/16
8/8 ━━━━━━━━━━━━━━━━━━━━ 18s 2s/step - accuracy: 0.9024 - loss: 0.52
67 - val_accuracy: 0.9444 - val_loss: 0.4488
Epoch 2/16
8/8 ━━━━━━━━━━━━━━━━━━━━ 17s 2s/step - accuracy: 0.8211 - loss: 0.54
06 - val_accuracy: 0.9444 - val_loss: 0.4355
Epoch 3/16
8/8 ━━━━━━━━━━━━━━━━━━━━ 17s 2s/step - accuracy: 0.8537 - loss: 0.52
06 - val_accuracy: 0.9630 - val_loss: 0.4219
Epoch 4/16
8/8 ━━━━━━━━━━━━━━━━━━━━ 17s 2s/step - accuracy: 0.8374 - loss: 0.47
91 - val_accuracy: 0.9630 - val_loss: 0.4101
Epoch 5/16
8/8 ━━━━━━━━━━━━━━━━━━━━ 16s 2s/step - accuracy: 0.8455 - loss: 0.46
32 - val_accuracy: 0.9630 - val_loss: 0.3985
Epoch 6/16
8/8 ━━━━━━━━━━━━━━━━━━━━ 17s 2s/step - accuracy: 0.8618 - loss: 0.49
10 - val_accuracy: 0.9630 - val_loss: 0.3869
Epoch 7/16
8/8 ━━━━━━━━━━━━━━━━━━━━ 17s 2s/step - accuracy: 0.8455 - loss: 0.50
02 - val_accuracy: 0.9630 - val_loss: 0.3766
Epoch 8/16
8/8 ━━━━━━━━━━━━━━━━━━━━ 16s 2s/step - accuracy: 0.8618 - loss: 0.50
68 - val_accuracy: 0.9630 - val_loss: 0.3664
Epoch 9/16
8/8 ━━━━━━━━━━━━━━━━━━━━ 16s 2s/step - accuracy: 0.8862 - loss: 0.43
06 - val_accuracy: 0.9630 - val_loss: 0.3564
Epoch 10/16
8/8 ━━━━━━━━━━━━━━━━━━━━ 16s 2s/step - accuracy: 0.9024 - loss: 0.44
19 - val_accuracy: 0.9630 - val_loss: 0.3469
Epoch 11/16
8/8 ━━━━━━━━━━━━━━━━━━━━ 16s 2s/step - accuracy: 0.9268 - loss: 0.39
78 - val_accuracy: 0.9630 - val_loss: 0.3379
Epoch 12/16
8/8 ━━━━━━━━━━━━━━━━━━━━ 16s 2s/step - accuracy: 0.9024 - loss: 0.41
33 - val_accuracy: 0.9630 - val_loss: 0.3296
Epoch 13/16
8/8 ━━━━━━━━━━━━━━━━━━━━ 16s 2s/step - accuracy: 0.8943 - loss: 0.39
38 - val_accuracy: 0.9630 - val_loss: 0.3215
Epoch 14/16
8/8 ━━━━━━━━━━━━━━━━━━━━ 16s 2s/step - accuracy: 0.9268 - loss: 0.36
30 - val_accuracy: 0.9630 - val_loss: 0.3129
Epoch 15/16
8/8 ━━━━━━━━━━━━━━━━━━━━ 16s 2s/step - accuracy: 0.9350 - loss: 0.32
97 - val_accuracy: 0.9630 - val_loss: 0.3046
Epoch 16/16
8/8 ━━━━━━━━━━━━━━━━━━━━ 16s 2s/step - accuracy: 0.8618 - loss: 0.43
39 - val_accuracy: 0.9630 - val_loss: 0.2968
2/2 ━━━━━━━━━━━━━━━━━━━━ 5s 2s/step - accuracy: 0.9630 - loss: 0.296
8
Test loss: 0.29680973291397095, Test accuracy: 0.9629629850387573
```

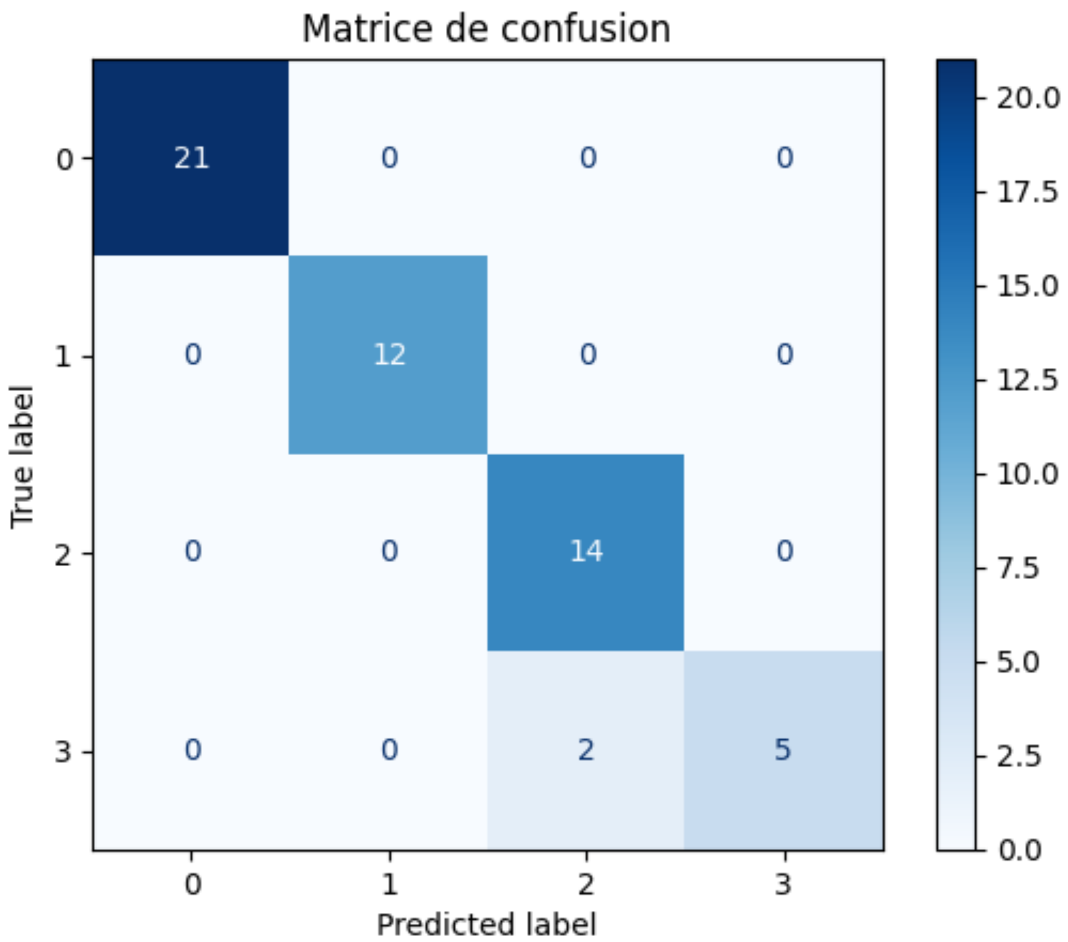


accuracy evolution
Train accuracy
Val accuracy
accuracy
0.96
0.94
0.92
0.90
0.88
0.86
0.84
0.82
0
2
4
6
8
10
12
14
Epoch



loss evolution
Train loss
Val loss
loss
0.55
0.50
0.45
0.40
0.35
0.30
0
2
4
6
8
10
12
14
Epoch

```
WARNING:tensorflow:5 out of the last 5 calls to <function TensorFlow
Trainer.make_predict_function.<locals>.one_step_on_data_distributed
at 0x1208b6d40> triggered tf.function retracing. Tracing is expensiv
e and the excessive number of tracings could be due to (1) creating
@tf.function repeatedly in a loop, (2) passing tensors with differen
t shapes, (3) passing Python objects instead of tensors. For (1), pl
ease define your @tf.function outside of the loop. For (2), @tf.func
tion has reduce_retracing=True option that can avoid unnecessary ret
racing. For (3), please refer to https://www.tensorflow.org/guide/fu
nction#controlling_retracing and https://www.tensorflow.org/api_doc
s/python/tf/function for  more details.
1/2 ━━━━━━━━━━━━━━━━━━━━ 2s 3s/stepWARNING:tensorflow:6 out of the l
ast 6 calls to <function TensorFlowTrainer.make_predict_function.<lo
cals>.one_step_on_data_distributed at 0x1208b6d40> triggered tf.func
tion retracing. Tracing is expensive and the excessive number of tra
cings could be due to (1) creating @tf.function repeatedly in a loo
p, (2) passing tensors with different shapes, (3) passing Python obj
ects instead of tensors. For (1), please define your @tf.function ou
tside of the loop. For (2), @tf.function has reduce_retracing=True o
ption that can avoid unnecessary retracing. For (3), please refer to
https://www.tensorflow.org/guide/function#controlling_retracing and
https://www.tensorflow.org/api_docs/python/tf/function for  more det
ails.
2/2 ━━━━━━━━━━━━━━━━━━━━ 5s 2s/step
```



```
Classification report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        21
           1       1.00      1.00      1.00        12
           2       0.88      1.00      0.93        14
           3       1.00      0.71      0.83         7

    accuracy                           0.96        54
   macro avg       0.97      0.93      0.94        54
weighted avg       0.97      0.96      0.96        54

Temps d'entraînement : 264.70 secondes
```

Avec du fine-tuning les performances sont encore améliorées:

```
Classification report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        21
           1       1.00      1.00      1.00        12
           2       0.88      1.00      0.93        14
           3       1.00      0.71      0.83         7

    accuracy                           0.96        54
   macro avg       0.97      0.93      0.94        54
weighted avg       0.97      0.96      0.96        54

Temps d'entraînement : 264.70 secondes
```

Pour éviter de ruiner les poids appris par VGG, j'ai changé la façon dont je permettais aux couches d'être fine-tuned pour ne laisser que les 4 dernieres couches (celles-ci étant les plus spécialisées).
On note aussi que la précision elle même a eu l'air de stagner, car ce fine-tuning n'a corrigé qu'une des 3 erreurs que faisais le modèle auparavant. Cependant, la loss à bel et bien progressé signifiant sûrement une hausse de la certitude.

In [32]:
```
from TP3_utils import load_mnist_with_noise
(X_train, Y_train, X_train_noise), (X_test, Y_test, X_test_noise) =

print(X_train_noise.shape, X_train.shape)
print(X_train_noise.min(), X_train_noise.max())
print(X_train.min(), X_train.max())

# Display the train data and a version of it with added noise
for i in range(5):
    plt.subplot(2,5,i+1)
    plt.imshow(X_train[i,:].reshape([28,28]), cmap='gray')
    plt.axis('off')
    plt.subplot(2,5,i+6)
    plt.imshow(X_train_noise[i,:].reshape([28,28]), cmap='gray')
    plt.axis('off')
    plt.show()
```

```
(60000, 28, 28, 1) (60000, 28, 28, 1)
0.0 1.0
0.0 1.0
```

In [ ]:

```
from TP3_utils import build_autoencoder


lr_scheduler = ReduceLROnPlateau(monitor='val_loss', factor=0.75, p
earlStop = EarlyStopping(monitor='val_loss', min_delta=1e-4, patien


autoencoder = build_autoencoder(X_train[0].shape)
autoencoder.summary()


autoencoder.compile(optimizer=Adam(1e-3), loss='mse')

history_ae = autoencoder.fit(
    X_train_noise,
    X_train,
    validation_data=(X_test_noise, X_test),
    epochs=8,
    batch_size=256,
    callbacks=[earlStop, lr_scheduler]
)


# Visualisation reconstruction
decoded_imgs = autoencoder.predict(X_test_noise)
for i in range(5):
    plt.subplot(2,5,i+1)
    plt.imshow(X_test[i].reshape(28,28), cmap='gray')
    plt.axis('off')
    plt.subplot(2,5,i+6)
    plt.imshow(decoded_imgs[i].reshape(28,28), cmap='gray')
    plt.axis('off')
plt.show()


affiche(history)
test_loss = autoencoder.evaluate(X_test_noise, X_test)
print(f'Test loss: {test_loss:.4f}')


decoded_imgs = autoencoder.predict(X_test_noise)

for i in range(5):
    plt.subplot(3,5,i+1)
    plt.imshow(X_test_noise[i].reshape(28,28), cmap='gray')
    plt.axis('off')
    if i == 0: plt.ylabel('Bruitée')

    plt.subplot(3,5,i+6)
    plt.imshow(decoded_imgs[i].reshape(28,28), cmap='gray')
    plt.axis('off')
    if i == 0: plt.ylabel('Reconstruite')

    plt.subplot(3,5,i+11)
    plt.imshow(X_test[i].reshape(28,28), cmap='gray')
    plt.axis('off')
```

```python
    if i == 0: plt.ylabel('Originale')
plt.show()

print("Temps d'entraînement : {:.2f} secondes".format(tps2 - tps1))
```

Model: "functional_32"

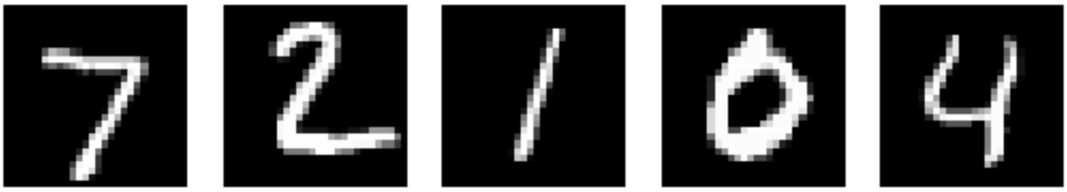
Layer (type)	Output Shape	
input_layer_44 (InputLayer)	(None, 28, 28, 1)	
conv2d_98 (Conv2D)	(None, 28, 28, 16)	
max_pooling2d_48 (MaxPooling2D)	(None, 14, 14, 16)	
conv2d_99 (Conv2D)	(None, 14, 14, 32)	
max_pooling2d_49 (MaxPooling2D)	(None, 7, 7, 32)	
conv2d_100 (Conv2D)	(None, 7, 7, 64)	
conv2d_transpose_46 (Conv2DTranspose)	(None, 14, 14, 32)	
conv2d_transpose_47 (Conv2DTranspose)	(None, 28, 28, 16)	
conv2d_101 (Conv2D)	(None, 28, 28, 1)	

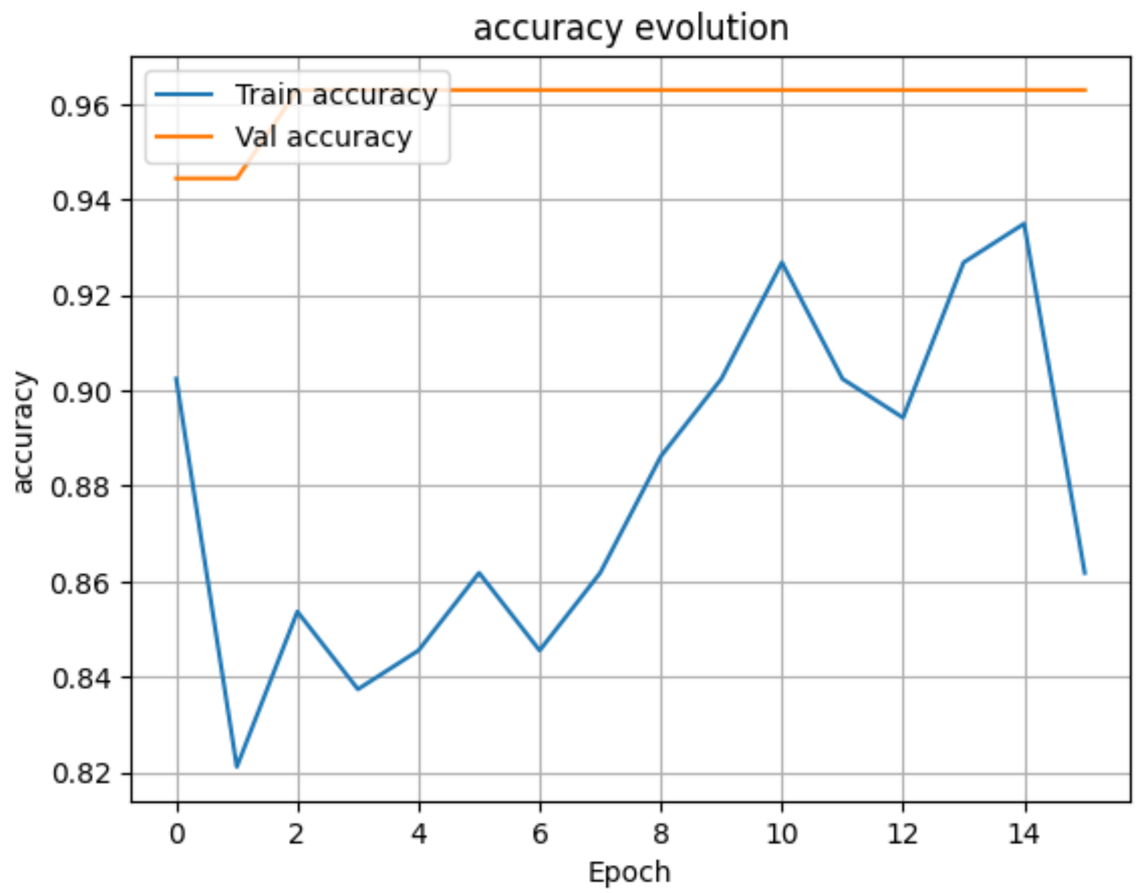
Total params: 46,529 (181.75 KB)

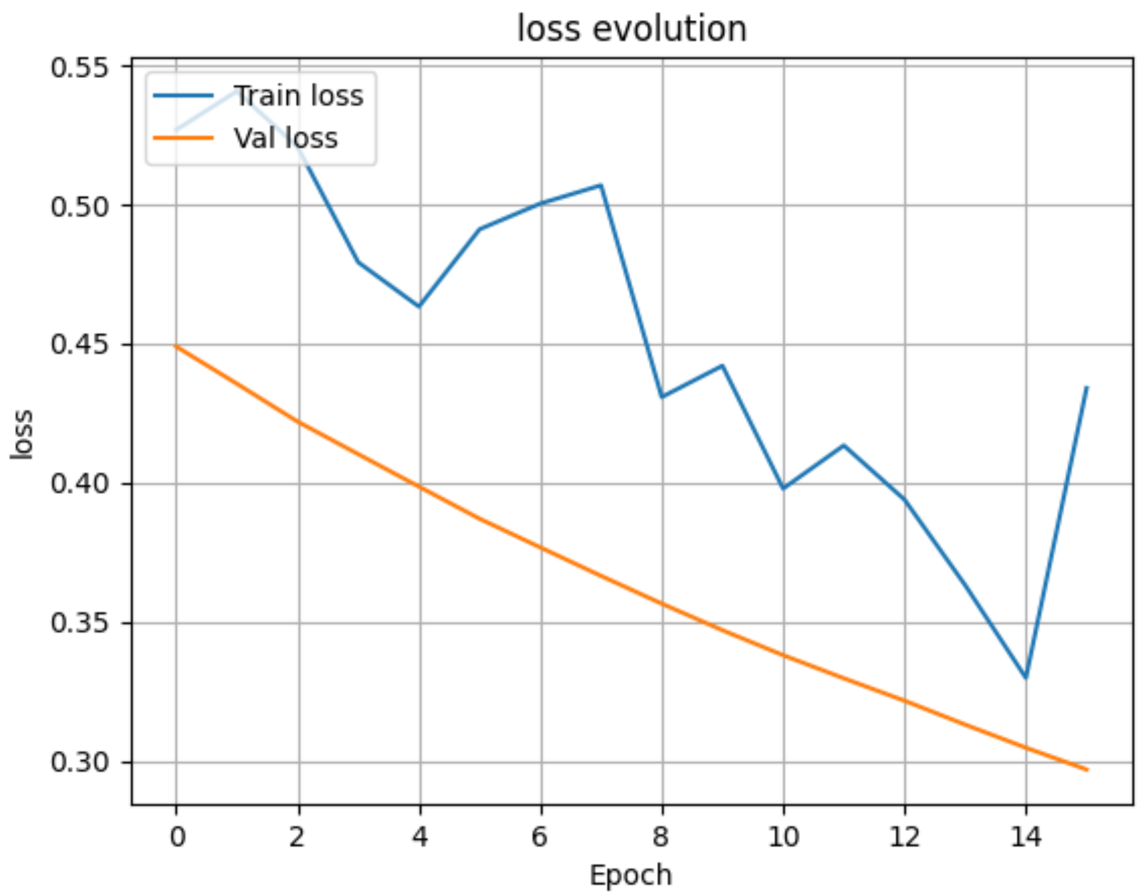
Trainable params: 46,529 (181.75 KB)

Non-trainable params: 0 (0.00 B)

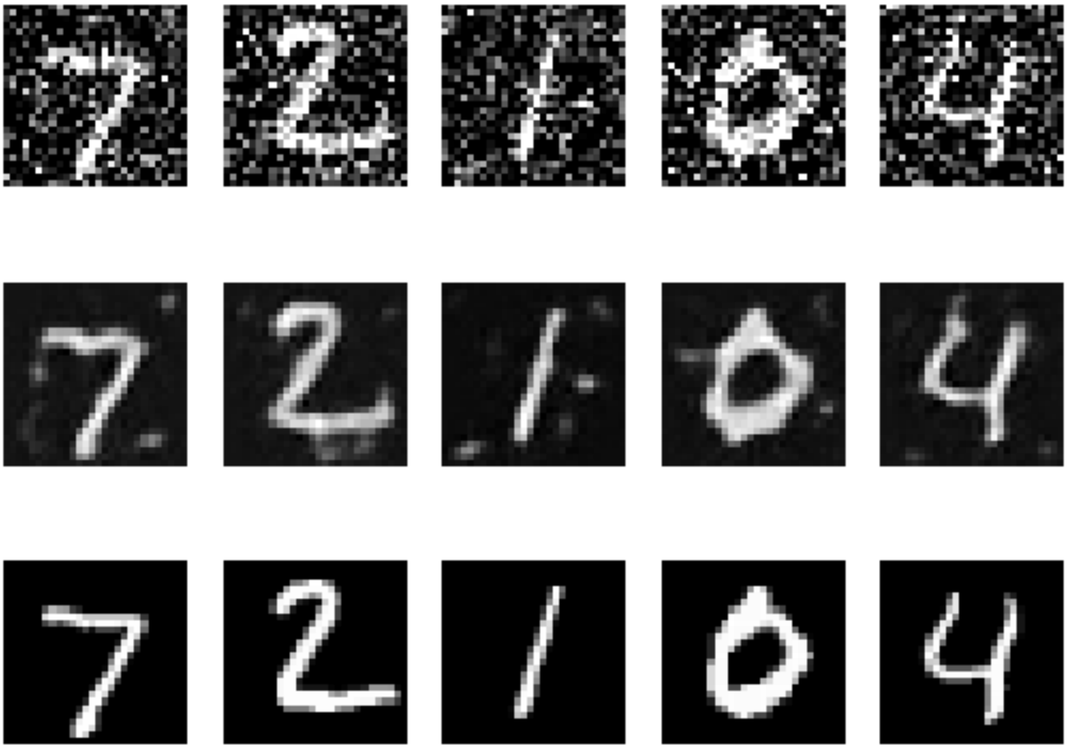
```
Epoch 1/8
235/235 ━━━━━━━━━━━━━━━━━━━━ 12s 46ms/step - loss: 0.0241 - val_los
s: 0.0157 - learning_rate: 0.0010
Epoch 2/8
235/235 ━━━━━━━━━━━━━━━━━━━━ 10s 44ms/step - loss: 0.0069 - val_los
s: 0.0143 - learning_rate: 0.0010
Epoch 3/8
235/235 ━━━━━━━━━━━━━━━━━━━━ 10s 44ms/step - loss: 0.0058 - val_los
s: 0.0136 - learning_rate: 0.0010
Epoch 4/8
235/235 ━━━━━━━━━━━━━━━━━━━━ 11s 45ms/step - loss: 0.0053 - val_los
s: 0.0132 - learning_rate: 0.0010
Epoch 5/8
235/235 ━━━━━━━━━━━━━━━━━━━━ 10s 44ms/step - loss: 0.0049 - val_los
s: 0.0130 - learning_rate: 0.0010
Epoch 6/8
235/235 ━━━━━━━━━━━━━━━━━━━━ 10s 44ms/step - loss: 0.0047 - val_los
s: 0.0130 - learning_rate: 0.0010
Epoch 7/8
235/235 ━━━━━━━━━━━━━━━━━━━━ 0s 42ms/step - loss: 0.0045
Epoch 7: ReduceLROnPlateau reducing learning rate to 0.0007500000356
230885.
235/235 ━━━━━━━━━━━━━━━━━━━━ 10s 44ms/step - loss: 0.0045 - val_los
s: 0.0129 - learning_rate: 0.0010
Epoch 8/8
235/235 ━━━━━━━━━━━━━━━━━━━━ 10s 44ms/step - loss: 0.0043 - val_los
s: 0.0129 - learning_rate: 7.5000e-04
Restoring model weights from the end of the best epoch: 5.
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step
```








```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.0130
Test loss: 0.0130
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step
```



```
Temps d'entraînement : 264.70 secondes
```

In [ ]:
```python
from TP3_utils import build_vae
lr_scheduler = ReduceLROnPlateau(monitor='val_loss', factor=0.8, pa
earlStop = EarlyStopping(monitor='val_loss', min_delta=1e-4, patien

latent_dim = 2
coeff = 0.01
vae, encoder, decoder = build_vae(latent_dim=latent_dim, coeff=coef
vae.compile(optimizer=Adam(1e-3), loss='mse')

history = vae.fit(
    X_train_noise, X_train,
    validation_data=(X_test_noise, X_test),
    epochs=20,
    batch_size=128,
    callbacks=[earlStop, lr_scheduler]
)
```

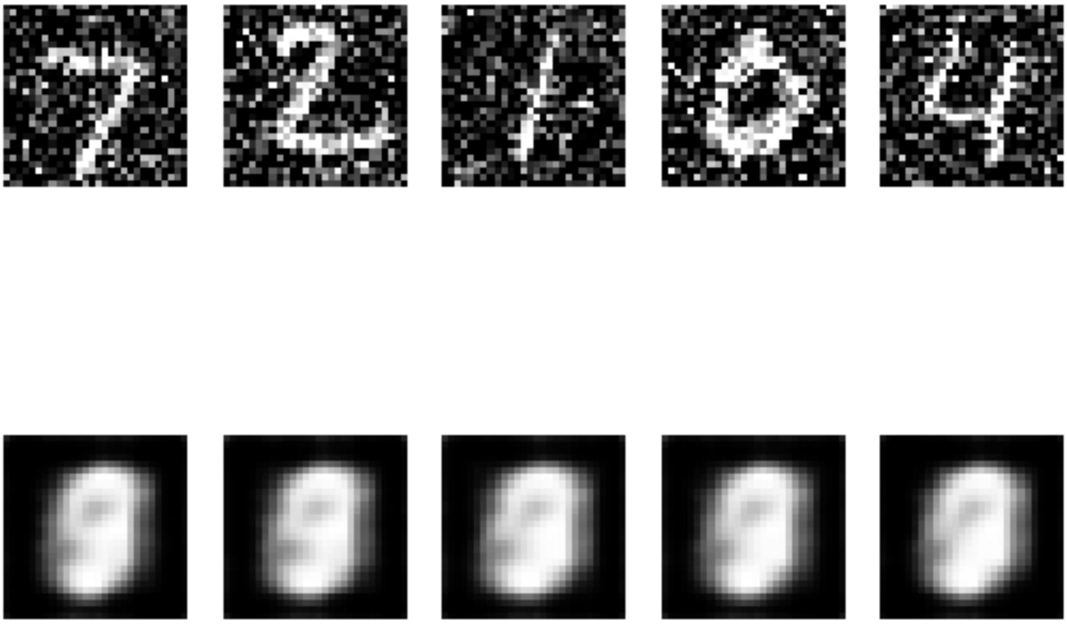
```
Epoch 1/20
469/469 ━━━━━━━━━━━━━━━━━━━━ 14s 28ms/step - loss: 0.0781 - val_los
s: 0.0680 - learning_rate: 0.0010
Epoch 2/20
467/469 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step - loss: 0.0675
Epoch 2: ReduceLROnPlateau reducing learning rate to 0.0007500000356
230885.
469/469 ━━━━━━━━━━━━━━━━━━━━ 12s 26ms/step - loss: 0.0675 - val_los
s: 0.0676 - learning_rate: 0.0010
Epoch 3/20
469/469 ━━━━━━━━━━━━━━━━━━━━ 13s 27ms/step - loss: 0.0674 - val_los
s: 0.0675 - learning_rate: 7.5000e-04
Epoch 4/20
468/469 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step - loss: 0.0672
Epoch 4: ReduceLROnPlateau reducing learning rate to 0.0005625000048
894435.
```

```
469/469 ━━━━━━━━━━━━━━━━━━━━ 13s 27ms/step - loss: 0.0673 - val_loss: 0.0675 - learning_rate: 7.5000e-04
Epoch 5/20
469/469 ━━━━━━━━━━━━━━━━━━━━ 13s 28ms/step - loss: 0.0673 - val_loss: 0.0675 - learning_rate: 5.6250e-04
Epoch 6/20
467/469 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step - loss: 0.0672
Epoch 6: ReduceLROnPlateau reducing learning rate to 0.0004218749818392098.
469/469 ━━━━━━━━━━━━━━━━━━━━ 13s 28ms/step - loss: 0.0673 - val_loss: 0.0675 - learning_rate: 5.6250e-04
Epoch 7/20
469/469 ━━━━━━━━━━━━━━━━━━━━ 13s 28ms/step - loss: 0.0673 - val_loss: 0.0675 - learning_rate: 4.2187e-04
Epoch 8/20
468/469 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step - loss: 0.0672
Epoch 8: ReduceLROnPlateau reducing learning rate to 0.00031640623637940735.
469/469 ━━━━━━━━━━━━━━━━━━━━ 13s 27ms/step - loss: 0.0673 - val_loss: 0.0675 - learning_rate: 4.2187e-04
Epoch 9/20
469/469 ━━━━━━━━━━━━━━━━━━━━ 13s 28ms/step - loss: 0.0673 - val_loss: 0.0675 - learning_rate: 3.1641e-04
Epoch 10/20
467/469 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step - loss: 0.0672
Epoch 10: ReduceLROnPlateau reducing learning rate to 0.00023730468819849193.
469/469 ━━━━━━━━━━━━━━━━━━━━ 13s 27ms/step - loss: 0.0673 - val_loss: 0.0675 - learning_rate: 3.1641e-04
Epoch 11/20
469/469 ━━━━━━━━━━━━━━━━━━━━ 13s 27ms/step - loss: 0.0673 - val_loss: 0.0675 - learning_rate: 2.3730e-04
Epoch 12/20
467/469 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step - loss: 0.0672
Epoch 12: ReduceLROnPlateau reducing learning rate to 0.00017797851614886895.
469/469 ━━━━━━━━━━━━━━━━━━━━ 13s 27ms/step - loss: 0.0673 - val_loss: 0.0675 - learning_rate: 2.3730e-04
Epoch 13/20
469/469 ━━━━━━━━━━━━━━━━━━━━ 13s 27ms/step - loss: 0.0673 - val_loss: 0.0675 - learning_rate: 1.7798e-04
Epoch 14/20
468/469 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step - loss: 0.0672
Epoch 14: ReduceLROnPlateau reducing learning rate to 0.0001334838816546835.
469/469 ━━━━━━━━━━━━━━━━━━━━ 13s 27ms/step - loss: 0.0673 - val_loss: 0.0675 - learning_rate: 1.7798e-04
Epoch 15/20
469/469 ━━━━━━━━━━━━━━━━━━━━ 13s 28ms/step - loss: 0.0673 - val_loss: 0.0675 - learning_rate: 1.3348e-04
Epoch 15: early stopping
Restoring model weights from the end of the best epoch: 1.
```

In [53]:
```
decoded_imgs = vae.predict(X_test_noise)
for i in range(5):
    plt.subplot(2,5,i+1)
```

```
    plt.imshow(X_test_noise[i].reshape(28,28), cmap='gray')
    plt.axis('off')
    if i==0: plt.ylabel('Bruitée')

    plt.subplot(2,5,i+6)
    plt.imshow(decoded_imgs[i].reshape(28,28), cmap='gray')
    plt.axis('off')
    if i==0: plt.ylabel('Reconstruite')
plt.show()
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step
```



In [54]:

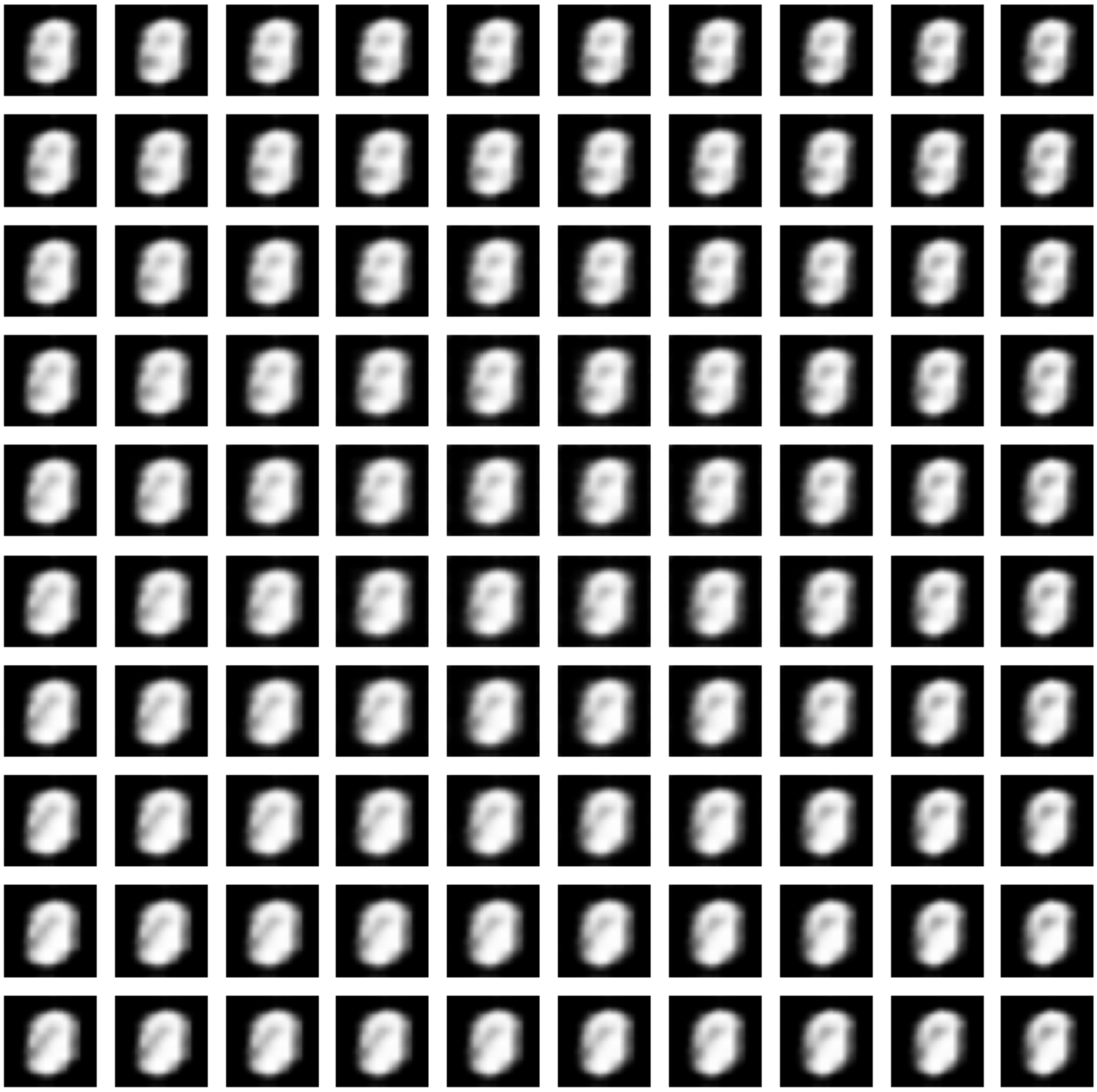
```
n = 10
grid_x = np.linspace(-3, 3, n)
grid_y = np.linspace(-3, 3, n)

plt.figure(figsize=(10,10))
for i, xi in enumerate(grid_x):
    for j, yi in enumerate(grid_y):
        z_sample = np.array([[xi, yi]])
        x_decoded = decoder.predict(z_sample)
        plt.subplot(n, n, i*n+j+1)
        plt.imshow(x_decoded[0].reshape(28,28), cmap='gray')
        plt.axis('off')
plt.suptitle("Espace latent du VAE")
plt.show()
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 29ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step
```

Espace latent du VAE



```
In [ ]:
```