AI.Lab4.autoencoders

December 22, 2021

1 Lab 3 - Création de simples auto-encodeurs avec Keras

2 3.2 Encodeur automatique le plus simple possible

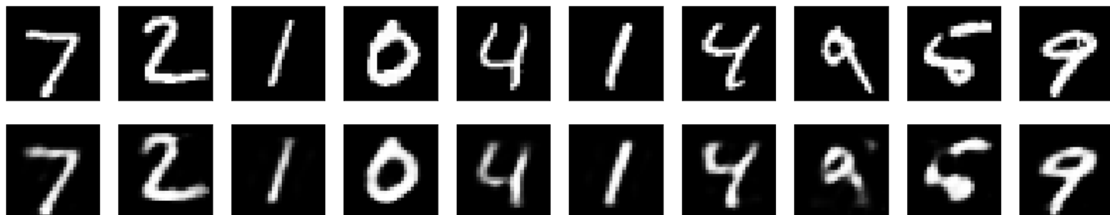
[1]:
```
import tensorflow.keras
from tensorflow.keras.layers import Input, Dense
from tensorflow.keras.models import Model
# this is the size of our encoded representations
encoding_dim = 32
# 32 floats -> compression of factor 24.5, assuming the input is 784 floats
# this is our input placeholder
input_img = Input(shape=(784,))
# "encoded" is the encoded representation of the input
encoded = Dense(encoding_dim, activation='relu')(input_img)
# "decoded" is the lossy reconstruction of the input
decoded = Dense(784, activation='sigmoid')(encoded)
# this model maps an input to its reconstruction
autoencoder = Model(input_img, decoded)
# this model maps an input to its encoded representation
encoder = Model(input_img, encoded)
# create a placeholder for an encoded (32-dimensional) input
encoded_input = Input(shape=(encoding_dim,))
# retrieve the last layer of the autoencoder model
decoder_layer = autoencoder.layers[-1]
# create the decoder model
decoder = Model(encoded_input, decoder_layer(encoded_input))
autoencoder.compile(optimizer='adam', loss='binary_crossentropy')
autoencoder.summary()

from tensorflow.keras.datasets import mnist
import numpy as np

(x_train, _), (x_test, _) = mnist.load_data()
x_train = x_train.astype('float32') / 255.
x_test = x_test.astype('float32') / 255.
x_train = x_train.reshape((len(x_train), np.prod(x_train.shape[1:])))
x_test = x_test.reshape((len(x_test), np.prod(x_test.shape[1:])))
```

```python
autoencoder.fit(x_train, x_train,
                epochs=20,
                batch_size=64,
                shuffle=True,
                validation_data=(x_test, x_test))
# encode and decode some digits
# note that we take them from the *test* set
encoded_imgs = encoder.predict(x_test)
decoded_imgs = decoder.predict(encoded_imgs)

# use Matplotlib
import matplotlib.pyplot as plt
n = 10 # how many digits we will display
plt.figure(figsize=(20, 4))

for i in range(n):
# display original
  ax = plt.subplot(2, n, i + 1)
  plt.imshow(x_test[i].reshape(28, 28))
  plt.gray()
  ax.get_xaxis().set_visible(False)
  ax.get_yaxis().set_visible(False)
  # display reconstruction
  ax = plt.subplot(2, n, i + 1 + n)
  plt.imshow(decoded_imgs[i].reshape(28, 28))
  plt.gray()
  ax.get_xaxis().set_visible(False)
  ax.get_yaxis().set_visible(False)

plt.show()
```

```
Model: "model"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 784)]             0

 dense (Dense)               (None, 32)                25120

 dense_1 (Dense)             (None, 784)               25872

=================================================================
Total params: 50,992
Trainable params: 50,992
Non-trainable params: 0
_________________________________________________________________
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-
datasets/mnist.npz
11493376/11490434 [==============================] - 0s 0us/step
11501568/11490434 [==============================] - 0s 0us/step
Epoch 1/20
938/938 [==============================] - 9s 7ms/step - loss: 0.1884 -
val_loss: 0.1319
Epoch 2/20
938/938 [==============================] - 6s 6ms/step - loss: 0.1192 -
val_loss: 0.1081
Epoch 3/20
938/938 [==============================] - 6s 6ms/step - loss: 0.1044 -
val_loss: 0.0987
Epoch 4/20
938/938 [==============================] - 6s 6ms/step - loss: 0.0979 -
val_loss: 0.0949
Epoch 5/20
938/938 [==============================] - 7s 7ms/step - loss: 0.0957 -
val_loss: 0.0939
Epoch 6/20
938/938 [==============================] - 5s 6ms/step - loss: 0.0949 -
val_loss: 0.0931
Epoch 7/20
938/938 [==============================] - 5s 5ms/step - loss: 0.0944 -
val_loss: 0.0929
Epoch 8/20
938/938 [==============================] - 5s 5ms/step - loss: 0.0942 -
val_loss: 0.0927
Epoch 9/20
938/938 [==============================] - 5s 5ms/step - loss: 0.0940 -
val_loss: 0.0927
Epoch 10/20
938/938 [==============================] - 5s 5ms/step - loss: 0.0938 -
val_loss: 0.0924
Epoch 11/20
938/938 [==============================] - 5s 5ms/step - loss: 0.0937 -
val_loss: 0.0924
Epoch 12/20
938/938 [==============================] - 5s 5ms/step - loss: 0.0936 -
val_loss: 0.0923
Epoch 13/20
938/938 [==============================] - 5s 5ms/step - loss: 0.0935 -
val_loss: 0.0922
Epoch 14/20
938/938 [==============================] - 5s 5ms/step - loss: 0.0935 -
val_loss: 0.0922
Epoch 15/20
938/938 [==============================] - 3s 4ms/step - loss: 0.0934 -
```

```
val_loss: 0.0924
Epoch 16/20
938/938 [==============================] - 3s 4ms/step - loss: 0.0933 -
val_loss: 0.0920
Epoch 17/20
938/938 [==============================] - 3s 4ms/step - loss: 0.0933 -
val_loss: 0.0920
Epoch 18/20
938/938 [==============================] - 3s 4ms/step - loss: 0.0932 -
val_loss: 0.0921
Epoch 19/20
938/938 [==============================] - 3s 4ms/step - loss: 0.0932 -
val_loss: 0.0919
Epoch 20/20
938/938 [==============================] - 3s 4ms/step - loss: 0.0932 -
val_loss: 0.0919
```



Avec le dataset **fasion_MINST** la perte est plus élévée, due à un nombre de détails et d'informations dans les images plus élévée que dans le MINST chiffre.

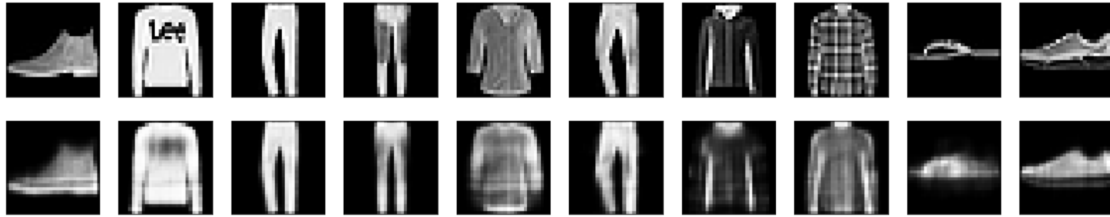
Après 20 epochs : loss: 0.2821

[ ]:
```
from tensorflow.keras.layers import Input, Dense
from tensorflow.keras.models import Model
# this is the size of our encoded representations
encoding_dim = 32
# 32 floats -> compression of factor 24.5, assuming the input is 784 floats
# this is our input placeholder
input_img = Input(shape=(784,))
# "encoded" is the encoded representation of the input
encoded = Dense(encoding_dim, activation='relu')(input_img)
# "decoded" is the lossy reconstruction of the input
decoded = Dense(784, activation='sigmoid')(encoded)
# this model maps an input to its reconstruction
autoencoder = Model(input_img, decoded)
# this model maps an input to its encoded representation
encoder = Model(input_img, encoded)
# create a placeholder for an encoded (32-dimensional) input
encoded_input = Input(shape=(encoding_dim,))
```

```
# retrieve the last layer of the autoencoder model
decoder_layer = autoencoder.layers[-1]
# create the decoder model
decoder = Model(encoded_input, decoder_layer(encoded_input))
autoencoder.compile(optimizer='adam', loss='binary_crossentropy')
autoencoder.summary()

from tensorflow.keras.datasets import fashion_mnist
import numpy as np

(x_train, _), (x_test, _) = fashion_mnist.load_data()
x_train = x_train.astype('float32') / 255.
x_test = x_test.astype('float32') / 255.
x_train = x_train.reshape((len(x_train), np.prod(x_train.shape[1:])))
x_test = x_test.reshape((len(x_test), np.prod(x_test.shape[1:])))

autoencoder.fit(x_train, x_train,
                epochs=20,
                batch_size=64,
                shuffle=True,
                validation_data=(x_test, x_test))
# encode and decode some digits
# note that we take them from the *test* set
encoded_imgs = encoder.predict(x_test)
decoded_imgs = decoder.predict(encoded_imgs)

# use Matplotlib
import matplotlib.pyplot as plt
n = 10 # how many digits we will display
plt.figure(figsize=(20, 4))

for i in range(n):
# display original
  ax = plt.subplot(2, n, i + 1)
  plt.imshow(x_test[i].reshape(28, 28))
  plt.gray()
  ax.get_xaxis().set_visible(False)
  ax.get_yaxis().set_visible(False)
  # display reconstruction
  ax = plt.subplot(2, n, i + 1 + n)
  plt.imshow(decoded_imgs[i].reshape(28, 28))
  plt.gray()
  ax.get_xaxis().set_visible(False)
  ax.get_yaxis().set_visible(False)

plt.show()
```

```
Model: "model_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_3 (InputLayer)        [(None, 784)]             0

 dense_2 (Dense)             (None, 32)                25120

 dense_3 (Dense)             (None, 784)               25872

=================================================================
Total params: 50,992
Trainable params: 50,992
Non-trainable params: 0
_________________________________________________________________
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-
datasets/train-labels-idx1-ubyte.gz
32768/29515 [=================================] - 0s 0us/step
40960/29515 [=========================================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-
datasets/train-images-idx3-ubyte.gz
26427392/26421880 [==============================] - 0s 0us/step
26435584/26421880 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-
datasets/t10k-labels-idx1-ubyte.gz
16384/5148 [==============================================================
==========================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-
datasets/t10k-images-idx3-ubyte.gz
4423680/4422102 [==============================] - 0s 0us/step
4431872/4422102 [==============================] - 0s 0us/step
Epoch 1/20
938/938 [==============================] - 5s 5ms/step - loss: 0.3484 -
val_loss: 0.3097
Epoch 2/20
938/938 [==============================] - 4s 4ms/step - loss: 0.2986 -
val_loss: 0.2941
Epoch 3/20
938/938 [==============================] - 4s 4ms/step - loss: 0.2892 -
val_loss: 0.2893
Epoch 4/20
938/938 [==============================] - 4s 4ms/step - loss: 0.2862 -
val_loss: 0.2875
Epoch 5/20
938/938 [==============================] - 4s 4ms/step - loss: 0.2846 -
val_loss: 0.2862
Epoch 6/20
938/938 [==============================] - 4s 4ms/step - loss: 0.2837 -
```

```
val_loss: 0.2856
Epoch 7/20
938/938 [==============================] - 4s 4ms/step - loss: 0.2832 -
val_loss: 0.2853
Epoch 8/20
938/938 [==============================] - 4s 4ms/step - loss: 0.2828 -
val_loss: 0.2849
Epoch 9/20
938/938 [==============================] - 4s 4ms/step - loss: 0.2826 -
val_loss: 0.2847
Epoch 10/20
938/938 [==============================] - 4s 4ms/step - loss: 0.2824 -
val_loss: 0.2845
Epoch 11/20
938/938 [==============================] - 4s 4ms/step - loss: 0.2822 -
val_loss: 0.2844
Epoch 12/20
938/938 [==============================] - 4s 4ms/step - loss: 0.2821 -
val_loss: 0.2843
Epoch 13/20
938/938 [==============================] - 4s 4ms/step - loss: 0.2820 -
val_loss: 0.2844
Epoch 14/20
938/938 [==============================] - 4s 4ms/step - loss: 0.2819 -
val_loss: 0.2841
Epoch 15/20
938/938 [==============================] - 4s 4ms/step - loss: 0.2818 -
val_loss: 0.2840
Epoch 16/20
938/938 [==============================] - 4s 4ms/step - loss: 0.2818 -
val_loss: 0.2842
Epoch 17/20
938/938 [==============================] - 5s 5ms/step - loss: 0.2817 -
val_loss: 0.2840
Epoch 18/20
938/938 [==============================] - 4s 4ms/step - loss: 0.2817 -
val_loss: 0.2839
Epoch 19/20
938/938 [==============================] - 4s 4ms/step - loss: 0.2816 -
val_loss: 0.2840
Epoch 20/20
938/938 [==============================] - 4s 4ms/step - loss: 0.2816 -
val_loss: 0.2845
```



3 3.2.1 Exercice

Modifiez ces paramètres avec epochs=50, et batch_size=256, et comparez les résultats d'affichage.

Malgré l'augmentation du nombre d'epoch et de bach size, la perte n'est pas plus faible.

[ ]:
```
from tensorflow.keras.layers import Input, Dense
from tensorflow.keras.models import Model
# this is the size of our encoded representations
encoding_dim = 32
# 32 floats -> compression of factor 24.5, assuming the input is 784 floats
# this is our input placeholder
input_img = Input(shape=(784,))
# "encoded" is the encoded representation of the input
encoded = Dense(encoding_dim, activation='relu')(input_img)
# "decoded" is the lossy reconstruction of the input
decoded = Dense(784, activation='sigmoid')(encoded)
# this model maps an input to its reconstruction
autoencoder = Model(input_img, decoded)
# this model maps an input to its encoded representation
encoder = Model(input_img, encoded)
# create a placeholder for an encoded (32-dimensional) input
encoded_input = Input(shape=(encoding_dim,))
# retrieve the last layer of the autoencoder model
decoder_layer = autoencoder.layers[-1]
# create the decoder model
decoder = Model(encoded_input, decoder_layer(encoded_input))
autoencoder.compile(optimizer='adam', loss='binary_crossentropy')
autoencoder.summary()

from tensorflow.keras.datasets import mnist
import numpy as np

(x_train, _), (x_test, _) = mnist.load_data()
x_train = x_train.astype('float32') / 255.
x_test = x_test.astype('float32') / 255.
x_train = x_train.reshape((len(x_train), np.prod(x_train.shape[1:])))
x_test = x_test.reshape((len(x_test), np.prod(x_test.shape[1:])))
```

```
autoencoder.fit(x_train, x_train,
                epochs=50,
                batch_size=256,
                shuffle=True,
                validation_data=(x_test, x_test))
# encode and decode some digits
# note that we take them from the *test* set
encoded_imgs = encoder.predict(x_test)
decoded_imgs = decoder.predict(encoded_imgs)

# use Matplotlib
import matplotlib.pyplot as plt
n = 10 # how many digits we will display
plt.figure(figsize=(20, 4))

for i in range(n):
# display original
  ax = plt.subplot(2, n, i + 1)
  plt.imshow(x_test[i].reshape(28, 28))
  plt.gray()
  ax.get_xaxis().set_visible(False)
  ax.get_yaxis().set_visible(False)
  # display reconstruction
  ax = plt.subplot(2, n, i + 1 + n)
  plt.imshow(decoded_imgs[i].reshape(28, 28))
  plt.gray()
  ax.get_xaxis().set_visible(False)
  ax.get_yaxis().set_visible(False)

plt.show()
```

```
Model: "model_6"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_5 (InputLayer)        [(None, 784)]             0

 dense_4 (Dense)             (None, 32)                25120

 dense_5 (Dense)             (None, 784)               25872

=================================================================
Total params: 50,992
Trainable params: 50,992
Non-trainable params: 0
_________________________________________________________________
```

```
Epoch 1/50
235/235 [==============================] - 2s 6ms/step - loss: 0.2799 -
val_loss: 0.1920
Epoch 2/50
235/235 [==============================] - 1s 5ms/step - loss: 0.1720 -
val_loss: 0.1526
Epoch 3/50
235/235 [==============================] - 1s 5ms/step - loss: 0.1425 -
val_loss: 0.1319
Epoch 4/50
235/235 [==============================] - 1s 5ms/step - loss: 0.1269 -
val_loss: 0.1197
Epoch 5/50
235/235 [==============================] - 1s 5ms/step - loss: 0.1169 -
val_loss: 0.1117
Epoch 6/50
235/235 [==============================] - 1s 5ms/step - loss: 0.1099 -
val_loss: 0.1056
Epoch 7/50
235/235 [==============================] - 1s 5ms/step - loss: 0.1047 -
val_loss: 0.1014
Epoch 8/50
235/235 [==============================] - 1s 5ms/step - loss: 0.1010 -
val_loss: 0.0984
Epoch 9/50
235/235 [==============================] - 1s 5ms/step - loss: 0.0986 -
val_loss: 0.0963
Epoch 10/50
235/235 [==============================] - 1s 5ms/step - loss: 0.0969 -
val_loss: 0.0950
Epoch 11/50
235/235 [==============================] - 1s 5ms/step - loss: 0.0958 -
val_loss: 0.0941
Epoch 12/50
235/235 [==============================] - 1s 5ms/step - loss: 0.0952 -
val_loss: 0.0935
Epoch 13/50
235/235 [==============================] - 1s 5ms/step - loss: 0.0947 -
val_loss: 0.0932
Epoch 14/50
235/235 [==============================] - 1s 5ms/step - loss: 0.0943 -
val_loss: 0.0929
Epoch 15/50
235/235 [==============================] - 1s 5ms/step - loss: 0.0941 -
val_loss: 0.0927
Epoch 16/50
235/235 [==============================] - 1s 5ms/step - loss: 0.0939 -
val_loss: 0.0924
```

```
Epoch 17/50
235/235 [==============================] - 1s 5ms/step - loss: 0.0937 -
val_loss: 0.0924
Epoch 18/50
235/235 [==============================] - 1s 5ms/step - loss: 0.0936 -
val_loss: 0.0923
Epoch 19/50
235/235 [==============================] - 1s 5ms/step - loss: 0.0935 -
val_loss: 0.0922
Epoch 20/50
235/235 [==============================] - 1s 5ms/step - loss: 0.0934 -
val_loss: 0.0922
Epoch 21/50
235/235 [==============================] - 1s 5ms/step - loss: 0.0934 -
val_loss: 0.0921
Epoch 22/50
235/235 [==============================] - 1s 5ms/step - loss: 0.0933 -
val_loss: 0.0919
Epoch 23/50
235/235 [==============================] - 1s 5ms/step - loss: 0.0932 -
val_loss: 0.0919
Epoch 24/50
235/235 [==============================] - 1s 5ms/step - loss: 0.0932 -
val_loss: 0.0919
Epoch 25/50
235/235 [==============================] - 1s 5ms/step - loss: 0.0931 -
val_loss: 0.0919
Epoch 26/50
235/235 [==============================] - 1s 5ms/step - loss: 0.0931 -
val_loss: 0.0919
Epoch 27/50
235/235 [==============================] - 1s 5ms/step - loss: 0.0931 -
val_loss: 0.0918
Epoch 28/50
235/235 [==============================] - 1s 5ms/step - loss: 0.0930 -
val_loss: 0.0918
Epoch 29/50
235/235 [==============================] - 1s 5ms/step - loss: 0.0930 -
val_loss: 0.0917
Epoch 30/50
235/235 [==============================] - 1s 5ms/step - loss: 0.0930 -
val_loss: 0.0918
Epoch 31/50
235/235 [==============================] - 1s 5ms/step - loss: 0.0929 -
val_loss: 0.0917
Epoch 32/50
235/235 [==============================] - 1s 5ms/step - loss: 0.0929 -
val_loss: 0.0917
```

```
Epoch 33/50
235/235 [==============================] - 1s 5ms/step - loss: 0.0929 -
val_loss: 0.0917
Epoch 34/50
235/235 [==============================] - 1s 5ms/step - loss: 0.0929 -
val_loss: 0.0917
Epoch 35/50
235/235 [==============================] - 1s 5ms/step - loss: 0.0929 -
val_loss: 0.0918
Epoch 36/50
235/235 [==============================] - 1s 5ms/step - loss: 0.0928 -
val_loss: 0.0917
Epoch 37/50
235/235 [==============================] - 1s 5ms/step - loss: 0.0928 -
val_loss: 0.0916
Epoch 38/50
235/235 [==============================] - 1s 5ms/step - loss: 0.0928 -
val_loss: 0.0917
Epoch 39/50
235/235 [==============================] - 1s 5ms/step - loss: 0.0928 -
val_loss: 0.0916
Epoch 40/50
235/235 [==============================] - 1s 5ms/step - loss: 0.0928 -
val_loss: 0.0916
Epoch 41/50
235/235 [==============================] - 1s 5ms/step - loss: 0.0927 -
val_loss: 0.0916
Epoch 42/50
235/235 [==============================] - 1s 5ms/step - loss: 0.0927 -
val_loss: 0.0916
Epoch 43/50
235/235 [==============================] - 1s 5ms/step - loss: 0.0927 -
val_loss: 0.0916
Epoch 44/50
235/235 [==============================] - 1s 5ms/step - loss: 0.0927 -
val_loss: 0.0915
Epoch 45/50
235/235 [==============================] - 1s 5ms/step - loss: 0.0927 -
val_loss: 0.0915
Epoch 46/50
235/235 [==============================] - 1s 5ms/step - loss: 0.0927 -
val_loss: 0.0915
Epoch 47/50
235/235 [==============================] - 1s 5ms/step - loss: 0.0927 -
val_loss: 0.0915
Epoch 48/50
235/235 [==============================] - 1s 5ms/step - loss: 0.0927 -
val_loss: 0.0915
```

```
Epoch 49/50
235/235 [==============================] - 1s 5ms/step - loss: 0.0927 -
val_loss: 0.0916
Epoch 50/50
235/235 [==============================] - 1s 5ms/step - loss: 0.0927 -
val_loss: 0.0915
```

Avec **fashion_MINST** : `loss: 0.2813` les résultats sont nettement moins bien, et augmenter le nombre d'epochs n'a pas diminié la perte.

```
[ ]: from tensorflow.keras.layers import Input, Dense
from tensorflow.keras.models import Model
# this is the size of our encoded representations
encoding_dim = 32
# 32 floats -> compression of factor 24.5, assuming the input is 784 floats
# this is our input placeholder
input_img = Input(shape=(784,))
# "encoded" is the encoded representation of the input
encoded = Dense(encoding_dim, activation='relu')(input_img)
# "decoded" is the lossy reconstruction of the input
decoded = Dense(784, activation='sigmoid')(encoded)
# this model maps an input to its reconstruction
autoencoder = Model(input_img, decoded)
# this model maps an input to its encoded representation
encoder = Model(input_img, encoded)
# create a placeholder for an encoded (32-dimensional) input
encoded_input = Input(shape=(encoding_dim,))
# retrieve the last layer of the autoencoder model
decoder_layer = autoencoder.layers[-1]
# create the decoder model
decoder = Model(encoded_input, decoder_layer(encoded_input))
autoencoder.compile(optimizer='adam', loss='binary_crossentropy')
autoencoder.summary()

from tensorflow.keras.datasets import fashion_mnist
import numpy as np

(x_train, _), (x_test, _) = fashion_mnist.load_data()
```

```
x_train = x_train.astype('float32') / 255.
x_test = x_test.astype('float32') / 255.
x_train = x_train.reshape((len(x_train), np.prod(x_train.shape[1:])))
x_test = x_test.reshape((len(x_test), np.prod(x_test.shape[1:])))

autoencoder.fit(x_train, x_train,
                epochs=50,
                batch_size=256,
                shuffle=True,
                validation_data=(x_test, x_test))
# encode and decode some digits
# note that we take them from the *test* set
encoded_imgs = encoder.predict(x_test)
decoded_imgs = decoder.predict(encoded_imgs)

# use Matplotlib
import matplotlib.pyplot as plt
n = 10 # how many digits we will display
plt.figure(figsize=(20, 4))

for i in range(n):
# display original
  ax = plt.subplot(2, n, i + 1)
  plt.imshow(x_test[i].reshape(28, 28))
  plt.gray()
  ax.get_xaxis().set_visible(False)
  ax.get_yaxis().set_visible(False)
  # display reconstruction
  ax = plt.subplot(2, n, i + 1 + n)
  plt.imshow(decoded_imgs[i].reshape(28, 28))
  plt.gray()
  ax.get_xaxis().set_visible(False)
  ax.get_yaxis().set_visible(False)

plt.show()
```

4 3.3 Un auto-encodeur profond

Avec une couche à 32 paramètres dans le NeuralNetwork on peut dire que le taux de compresssion est (taille initiale image) / (couche la plus petite) = 784/32 = 24,5

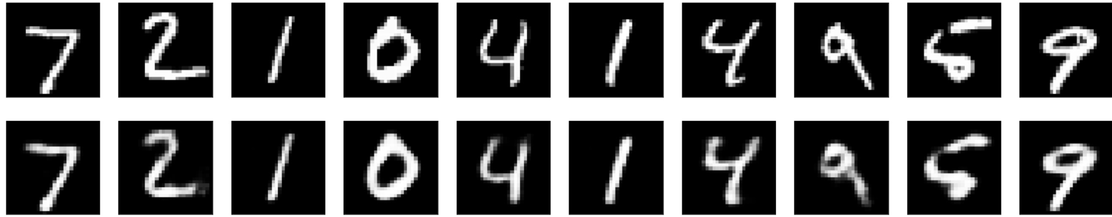
[ ]:
```
from tensorflow.keras.layers import Input, Dense
from tensorflow.keras.models import Model
# this is the size of our encoded representations
encoding_dim = 32
# 32 floats -> compression of factor 24.5, assuming the input is 784 floats
```

```python
# this is our input placeholder
input_img = Input(shape=(784,))
# encoder part
encoded = Dense(128, activation='relu')(input_img)
encoded = Dense(64, activation='relu')(encoded)
encoded = Dense(32, activation='relu')(encoded)
# decoder part
decoded = Dense(64, activation='relu')(encoded)
decoded = Dense(128, activation='relu')(decoded)
decoded = Dense(784, activation='sigmoid')(decoded)

# separate models for autoencoder, encoder and decoder
autoencoder = Model(input_img, decoded)
encoder = Model(input_img, encoded)
# create a placeholder for an encoded (32-dimensional) input
encoded_input = Input(shape=(encoding_dim,))
# create the decoder model
# retrieve the last layer of the autoencoder model
decoder_layer1 = autoencoder.layers[-3]
decoder_layer2 = autoencoder.layers[-2]
decoder_layer3 = autoencoder.layers[-1]
decoder = Model(encoded_input,
decoder_layer3(decoder_layer2(decoder_layer1(encoded_input))))
decoder.summary()
decoder.save('autoenc_deep_decoder.h5')
autoencoder.compile(optimizer='adam', loss='binary_crossentropy')
autoencoder.summary()
autoencoder.save('autoenc_deep.h5')
from tensorflow.keras.datasets import mnist
import numpy as np
(x_train, _), (x_test, _) = mnist.load_data()

x_train = x_train.astype('float32') / 255.
x_test = x_test.astype('float32') / 255.
x_train = x_train.reshape((len(x_train), np.prod(x_train.shape[1:])))
x_test = x_test.reshape((len(x_test), np.prod(x_test.shape[1:])))
autoencoder.fit(x_train, x_train,
                epochs=10,
                batch_size=64,
                shuffle=True,
                validation_data=(x_test, x_test))
# encode and decode some digits
# note that we take them from the *test* set
encoded_imgs = encoder.predict(x_test)
decoded_imgs = decoder.predict(encoded_imgs)
import matplotlib.pyplot as plt
n = 10 # how many digits we will display
```

```
plt.figure(figsize=(20, 4))
for i in range(n):
  # display original
  ax = plt.subplot(2, n, i + 1)
  plt.imshow(x_test[i].reshape(28, 28))
  plt.gray()
  ax.get_xaxis().set_visible(False)
  ax.get_yaxis().set_visible(False)
  # display reconstruction
  ax = plt.subplot(2, n, i + 1 + n)
  plt.imshow(decoded_imgs[i].reshape(28, 28))
  plt.gray()
  ax.get_xaxis().set_visible(False)
  ax.get_yaxis().set_visible(False)
plt.show()
```

```
Model: "model_11"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_8 (InputLayer)        [(None, 32)]              0

 dense_9 (Dense)             (None, 64)                2112

 dense_10 (Dense)            (None, 128)               8320

 dense_11 (Dense)            (None, 784)               101136

=================================================================
Total params: 111,568
Trainable params: 111,568
Non-trainable params: 0
_________________________________________________________________
WARNING:tensorflow:Compiled the loaded model, but the compiled metrics have yet
to be built. `model.compile_metrics` will be empty until you train or evaluate
the model.
Model: "model_9"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_7 (InputLayer)        [(None, 784)]             0

 dense_6 (Dense)             (None, 128)               100480

 dense_7 (Dense)             (None, 64)                8256

 dense_8 (Dense)             (None, 32)                2080
```

```
 dense_9 (Dense)             (None, 64)                2112

 dense_10 (Dense)            (None, 128)               8320

 dense_11 (Dense)            (None, 784)               101136

=================================================================
Total params: 222,384
Trainable params: 222,384
Non-trainable params: 0
_________________________________________________________________
Epoch 1/10
938/938 [==============================] - 6s 5ms/step - loss: 0.1694 -
val_loss: 0.1238
Epoch 2/10
938/938 [==============================] - 5s 5ms/step - loss: 0.1171 -
val_loss: 0.1092
Epoch 3/10
938/938 [==============================] - 5s 5ms/step - loss: 0.1063 -
val_loss: 0.1017
Epoch 4/10
938/938 [==============================] - 5s 5ms/step - loss: 0.1008 -
val_loss: 0.0972
Epoch 5/10
938/938 [==============================] - 5s 5ms/step - loss: 0.0972 -
val_loss: 0.0949
Epoch 6/10
938/938 [==============================] - 5s 5ms/step - loss: 0.0946 -
val_loss: 0.0927
Epoch 7/10
938/938 [==============================] - 5s 5ms/step - loss: 0.0925 -
val_loss: 0.0907
Epoch 8/10
938/938 [==============================] - 5s 5ms/step - loss: 0.0910 -
val_loss: 0.0898
Epoch 9/10
938/938 [==============================] - 5s 5ms/step - loss: 0.0899 -
val_loss: 0.0894
Epoch 10/10
938/938 [==============================] - 5s 5ms/step - loss: 0.0890 -
val_loss: 0.0878
```



5 3.3.2 Exercice

Modifions les paramètres avec epochs=50, et batch_size=256.

Les résultats nous donnent une perte plus faible de 10% par rapport à 10 epochs et un bach_size de 64 : `loss: 0.0835 - val_loss: 0.0841`

[ ]:
```
from tensorflow.keras.layers import Input, Dense
from tensorflow.keras.models import Model
# this is the size of our encoded representations
encoding_dim = 32
# 32 floats -> compression of factor 24.5, assuming the input is 784 floats
# this is our input placeholder
input_img = Input(shape=(784,))
# encoder part
encoded = Dense(128, activation='relu')(input_img)
encoded = Dense(64, activation='relu')(encoded)
encoded = Dense(32, activation='relu')(encoded)
# decoder part
decoded = Dense(64, activation='relu')(encoded)
decoded = Dense(128, activation='relu')(decoded)
decoded = Dense(784, activation='sigmoid')(decoded)

# separate models for autoencoder, encoder and decoder
autoencoder = Model(input_img, decoded)
encoder = Model(input_img, encoded)
# create a placeholder for an encoded (32-dimensional) input
encoded_input = Input(shape=(encoding_dim,))
# create the decoder model
# retrieve the last layer of the autoencoder model
decoder_layer1 = autoencoder.layers[-3]
decoder_layer2 = autoencoder.layers[-2]
decoder_layer3 = autoencoder.layers[-1]
decoder = Model(encoded_input,
decoder_layer3(decoder_layer2(decoder_layer1(encoded_input))))
decoder.summary()
decoder.save('autoenc_deep_decoder.h5')
autoencoder.compile(optimizer='adam', loss='binary_crossentropy')
```

```python
autoencoder.summary()
autoencoder.save('autoenc_deep.h5')

from tensorflow.keras.datasets import mnist
import numpy as np
(x_train, _), (x_test, _) = mnist.load_data()

x_train = x_train.astype('float32') / 255.
x_test = x_test.astype('float32') / 255.
x_train = x_train.reshape((len(x_train), np.prod(x_train.shape[1:])))
x_test = x_test.reshape((len(x_test), np.prod(x_test.shape[1:])))
autoencoder.fit(x_train, x_train,
                epochs=50,
                batch_size=256,
                shuffle=True,
                validation_data=(x_test, x_test))
# encode and decode some digits
# note that we take them from the *test* set
encoded_imgs = encoder.predict(x_test)
decoded_imgs = decoder.predict(encoded_imgs)
import matplotlib.pyplot as plt
n = 10 # how many digits we will display
plt.figure(figsize=(20, 4))
for i in range(n):
  # display original
  ax = plt.subplot(2, n, i + 1)
  plt.imshow(x_test[i].reshape(28, 28))
  plt.gray()
  ax.get_xaxis().set_visible(False)
  ax.get_yaxis().set_visible(False)
  # display reconstruction
  ax = plt.subplot(2, n, i + 1 + n)
  plt.imshow(decoded_imgs[i].reshape(28, 28))
  plt.gray()
  ax.get_xaxis().set_visible(False)
  ax.get_yaxis().set_visible(False)
plt.show()
```

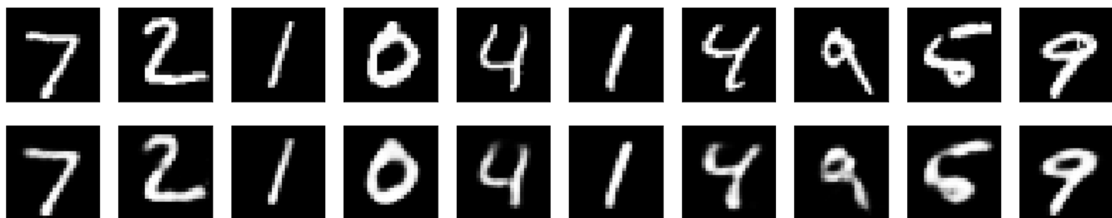
```
Model: "model_14"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_10 (InputLayer)       [(None, 32)]              0

 dense_15 (Dense)            (None, 64)                2112

 dense_16 (Dense)            (None, 128)               8320
```

```
 dense_17 (Dense)            (None, 784)              101136

=================================================================
Total params: 111,568
Trainable params: 111,568
Non-trainable params: 0
_________________________________________________________________
WARNING:tensorflow:Compiled the loaded model, but the compiled metrics have yet
to be built. `model.compile_metrics` will be empty until you train or evaluate
the model.
Model: "model_12"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_9 (InputLayer)        [(None, 784)]             0

 dense_12 (Dense)            (None, 128)               100480

 dense_13 (Dense)            (None, 64)                8256

 dense_14 (Dense)            (None, 32)                2080

 dense_15 (Dense)            (None, 64)                2112

 dense_16 (Dense)            (None, 128)               8320

 dense_17 (Dense)            (None, 784)               101136

=================================================================
Total params: 222,384
Trainable params: 222,384
Non-trainable params: 0
_________________________________________________________________
Epoch 1/50
235/235 [==============================] - 2s 7ms/step - loss: 0.2482 -
val_loss: 0.1665
Epoch 2/50
235/235 [==============================] - 1s 6ms/step - loss: 0.1511 -
val_loss: 0.1387
Epoch 3/50
235/235 [==============================] - 1s 6ms/step - loss: 0.1339 -
val_loss: 0.1272
Epoch 4/50
235/235 [==============================] - 1s 6ms/step - loss: 0.1239 -
val_loss: 0.1188
Epoch 5/50
235/235 [==============================] - 1s 6ms/step - loss: 0.1178 -
```

```
val_loss: 0.1139
Epoch 6/50
235/235 [==============================] - 1s 6ms/step - loss: 0.1136 -
val_loss: 0.1106
Epoch 7/50
235/235 [==============================] - 1s 6ms/step - loss: 0.1103 -
val_loss: 0.1079
Epoch 8/50
235/235 [==============================] - 1s 6ms/step - loss: 0.1074 -
val_loss: 0.1048
Epoch 9/50
235/235 [==============================] - 1s 6ms/step - loss: 0.1052 -
val_loss: 0.1028
Epoch 10/50
235/235 [==============================] - 1s 6ms/step - loss: 0.1033 -
val_loss: 0.1015
Epoch 11/50
235/235 [==============================] - 1s 6ms/step - loss: 0.1018 -
val_loss: 0.1001
Epoch 12/50
235/235 [==============================] - 1s 6ms/step - loss: 0.1005 -
val_loss: 0.0986
Epoch 13/50
235/235 [==============================] - 1s 6ms/step - loss: 0.0993 -
val_loss: 0.0979
Epoch 14/50
235/235 [==============================] - 1s 6ms/step - loss: 0.0982 -
val_loss: 0.0970
Epoch 15/50
235/235 [==============================] - 1s 6ms/step - loss: 0.0972 -
val_loss: 0.0958
Epoch 16/50
235/235 [==============================] - 1s 6ms/step - loss: 0.0963 -
val_loss: 0.0954
Epoch 17/50
235/235 [==============================] - 1s 6ms/step - loss: 0.0955 -
val_loss: 0.0941
Epoch 18/50
235/235 [==============================] - 1s 6ms/step - loss: 0.0948 -
val_loss: 0.0938
Epoch 19/50
235/235 [==============================] - 1s 6ms/step - loss: 0.0941 -
val_loss: 0.0932
Epoch 20/50
235/235 [==============================] - 1s 6ms/step - loss: 0.0936 -
val_loss: 0.0923
Epoch 21/50
235/235 [==============================] - 1s 6ms/step - loss: 0.0929 -
```

```
val_loss: 0.0922
Epoch 22/50
235/235 [==============================] - 1s 6ms/step - loss: 0.0924 -
val_loss: 0.0912
Epoch 23/50
235/235 [==============================] - 1s 6ms/step - loss: 0.0918 -
val_loss: 0.0907
Epoch 24/50
235/235 [==============================] - 1s 6ms/step - loss: 0.0913 -
val_loss: 0.0904
Epoch 25/50
235/235 [==============================] - 1s 6ms/step - loss: 0.0908 -
val_loss: 0.0899
Epoch 26/50
235/235 [==============================] - 1s 6ms/step - loss: 0.0905 -
val_loss: 0.0897
Epoch 27/50
235/235 [==============================] - 1s 6ms/step - loss: 0.0901 -
val_loss: 0.0894
Epoch 28/50
235/235 [==============================] - 1s 6ms/step - loss: 0.0897 -
val_loss: 0.0890
Epoch 29/50
235/235 [==============================] - 1s 6ms/step - loss: 0.0895 -
val_loss: 0.0886
Epoch 30/50
235/235 [==============================] - 1s 6ms/step - loss: 0.0892 -
val_loss: 0.0881
Epoch 31/50
235/235 [==============================] - 1s 6ms/step - loss: 0.0889 -
val_loss: 0.0882
Epoch 32/50
235/235 [==============================] - 1s 6ms/step - loss: 0.0887 -
val_loss: 0.0880
Epoch 33/50
235/235 [==============================] - 1s 6ms/step - loss: 0.0885 -
val_loss: 0.0879
Epoch 34/50
235/235 [==============================] - 2s 6ms/step - loss: 0.0883 -
val_loss: 0.0877
Epoch 35/50
235/235 [==============================] - 1s 6ms/step - loss: 0.0880 -
val_loss: 0.0871
Epoch 36/50
235/235 [==============================] - 1s 6ms/step - loss: 0.0879 -
val_loss: 0.0870
Epoch 37/50
235/235 [==============================] - 1s 6ms/step - loss: 0.0875 -
```

```
val_loss: 0.0869
Epoch 38/50
235/235 [==============================] - 1s 6ms/step - loss: 0.0874 -
val_loss: 0.0868
Epoch 39/50
235/235 [==============================] - 1s 6ms/step - loss: 0.0872 -
val_loss: 0.0869
Epoch 40/50
235/235 [==============================] - 1s 6ms/step - loss: 0.0871 -
val_loss: 0.0864
Epoch 41/50
235/235 [==============================] - 1s 6ms/step - loss: 0.0868 -
val_loss: 0.0863
Epoch 42/50
235/235 [==============================] - 1s 6ms/step - loss: 0.0867 -
val_loss: 0.0864
Epoch 43/50
235/235 [==============================] - 1s 6ms/step - loss: 0.0866 -
val_loss: 0.0859
Epoch 44/50
235/235 [==============================] - 1s 6ms/step - loss: 0.0864 -
val_loss: 0.0860
Epoch 45/50
235/235 [==============================] - 1s 6ms/step - loss: 0.0863 -
val_loss: 0.0859
Epoch 46/50
235/235 [==============================] - 1s 6ms/step - loss: 0.0862 -
val_loss: 0.0857
Epoch 47/50
235/235 [==============================] - 1s 6ms/step - loss: 0.0861 -
val_loss: 0.0856
Epoch 48/50
235/235 [==============================] - 1s 6ms/step - loss: 0.0859 -
val_loss: 0.0856
Epoch 49/50
235/235 [==============================] - 1s 6ms/step - loss: 0.0859 -
val_loss: 0.0852
Epoch 50/50
235/235 [==============================] - 1s 6ms/step - loss: 0.0858 -
val_loss: 0.0850
```



Avec **fashion_MINST** : `loss: 0.2735` les résultats sont nettement moins bons, et augmenter le nombre d'epochs n'a pas diminié la perte.

```
[ ]: from tensorflow.keras.layers import Input, Dense
from tensorflow.keras.models import Model
# this is the size of our encoded representations
encoding_dim = 32
# 32 floats -> compression of factor 24.5, assuming the input is 784 floats
# this is our input placeholder
input_img = Input(shape=(784,))
# encoder part
encoded = Dense(128, activation='relu')(input_img)
encoded = Dense(64, activation='relu')(encoded)
encoded = Dense(32, activation='relu')(encoded)
# decoder part
decoded = Dense(64, activation='relu')(encoded)
decoded = Dense(128, activation='relu')(decoded)
decoded = Dense(784, activation='sigmoid')(decoded)

# separate models for autoencoder, encoder and decoder
autoencoder = Model(input_img, decoded)
encoder = Model(input_img, encoded)
# create a placeholder for an encoded (32-dimensional) input
encoded_input = Input(shape=(encoding_dim,))
# create the decoder model
# retrieve the last layer of the autoencoder model
decoder_layer1 = autoencoder.layers[-3]
decoder_layer2 = autoencoder.layers[-2]
decoder_layer3 = autoencoder.layers[-1]
decoder = Model(encoded_input,
decoder_layer3(decoder_layer2(decoder_layer1(encoded_input))))
decoder.summary()
decoder.save('autoenc_deep_decoder.h5')
autoencoder.compile(optimizer='adam', loss='binary_crossentropy')
autoencoder.summary()
autoencoder.save('autoenc_deep.h5')

from tensorflow.keras.datasets import fashion_mnist
import numpy as np
(x_train, _), (x_test, _) = fashion_mnist.load_data()

x_train = x_train.astype('float32') / 255.
x_test = x_test.astype('float32') / 255.
x_train = x_train.reshape((len(x_train), np.prod(x_train.shape[1:])))
x_test = x_test.reshape((len(x_test), np.prod(x_test.shape[1:])))
```

```python
autoencoder.fit(x_train, x_train,
                epochs=50,
                batch_size=256,
                shuffle=True,
                validation_data=(x_test, x_test))
# encode and decode some digits
# note that we take them from the *test* set
encoded_imgs = encoder.predict(x_test)
decoded_imgs = decoder.predict(encoded_imgs)
import matplotlib.pyplot as plt
n = 10 # how many digits we will display
plt.figure(figsize=(20, 4))
for i in range(n):
  # display original
  ax = plt.subplot(2, n, i + 1)
  plt.imshow(x_test[i].reshape(28, 28))
  plt.gray()
  ax.get_xaxis().set_visible(False)
  ax.get_yaxis().set_visible(False)
  # display reconstruction
  ax = plt.subplot(2, n, i + 1 + n)
  plt.imshow(decoded_imgs[i].reshape(28, 28))
  plt.gray()
  ax.get_xaxis().set_visible(False)
  ax.get_yaxis().set_visible(False)
plt.show()
```

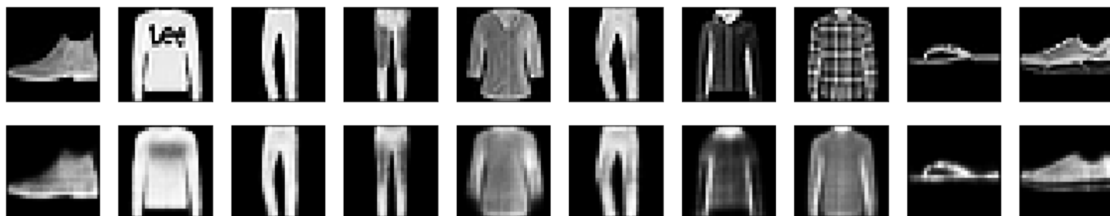
```
Model: "model_17"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_12 (InputLayer)       [(None, 32)]              0

 dense_21 (Dense)            (None, 64)                2112

 dense_22 (Dense)            (None, 128)               8320

 dense_23 (Dense)            (None, 784)               101136

=================================================================
Total params: 111,568
Trainable params: 111,568
Non-trainable params: 0
_________________________________________________________________
WARNING:tensorflow:Compiled the loaded model, but the compiled metrics have yet
to be built. `model.compile_metrics` will be empty until you train or evaluate
the model.
```

```
Model: "model_15"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 input_11 (InputLayer)       [(None, 784)]             0         
                                                                 
 dense_18 (Dense)            (None, 128)               100480    
                                                                 
 dense_19 (Dense)            (None, 64)                8256      
                                                                 
 dense_20 (Dense)            (None, 32)                2080      
                                                                 
 dense_21 (Dense)            (None, 64)                2112      
                                                                 
 dense_22 (Dense)            (None, 128)               8320      
                                                                 
 dense_23 (Dense)            (None, 784)               101136    
                                                                 
=================================================================
Total params: 222,384
Trainable params: 222,384
Non-trainable params: 0
_________________________________________________________________
Epoch 1/50
235/235 [==============================] - 2s 7ms/step - loss: 0.3764 -
val_loss: 0.3176
Epoch 2/50
235/235 [==============================] - 1s 6ms/step - loss: 0.3096 -
val_loss: 0.3063
Epoch 3/50
235/235 [==============================] - 1s 6ms/step - loss: 0.3007 -
val_loss: 0.2998
Epoch 4/50
235/235 [==============================] - 1s 6ms/step - loss: 0.2960 -
val_loss: 0.2963
Epoch 5/50
235/235 [==============================] - 1s 6ms/step - loss: 0.2931 -
val_loss: 0.2943
Epoch 6/50
235/235 [==============================] - 1s 6ms/step - loss: 0.2910 -
val_loss: 0.2922
Epoch 7/50
235/235 [==============================] - 1s 6ms/step - loss: 0.2892 -
val_loss: 0.2904
Epoch 8/50
235/235 [==============================] - 1s 6ms/step - loss: 0.2876 -
val_loss: 0.2891
Epoch 9/50
```

```
235/235 [==============================] - 1s 6ms/step - loss: 0.2863 -
val_loss: 0.2882
Epoch 10/50
235/235 [==============================] - 2s 6ms/step - loss: 0.2852 -
val_loss: 0.2870
Epoch 11/50
235/235 [==============================] - 1s 6ms/step - loss: 0.2842 -
val_loss: 0.2860
Epoch 12/50
235/235 [==============================] - 2s 6ms/step - loss: 0.2834 -
val_loss: 0.2852
Epoch 13/50
235/235 [==============================] - 2s 6ms/step - loss: 0.2826 -
val_loss: 0.2845
Epoch 14/50
235/235 [==============================] - 1s 6ms/step - loss: 0.2818 -
val_loss: 0.2852
Epoch 15/50
235/235 [==============================] - 1s 6ms/step - loss: 0.2812 -
val_loss: 0.2836
Epoch 16/50
235/235 [==============================] - 1s 6ms/step - loss: 0.2805 -
val_loss: 0.2826
Epoch 17/50
235/235 [==============================] - 1s 6ms/step - loss: 0.2801 -
val_loss: 0.2821
Epoch 18/50
235/235 [==============================] - 1s 6ms/step - loss: 0.2796 -
val_loss: 0.2821
Epoch 19/50
235/235 [==============================] - 1s 6ms/step - loss: 0.2791 -
val_loss: 0.2812
Epoch 20/50
235/235 [==============================] - 1s 6ms/step - loss: 0.2787 -
val_loss: 0.2809
Epoch 21/50
235/235 [==============================] - 1s 6ms/step - loss: 0.2783 -
val_loss: 0.2805
Epoch 22/50
235/235 [==============================] - 1s 6ms/step - loss: 0.2780 -
val_loss: 0.2803
Epoch 23/50
235/235 [==============================] - 1s 6ms/step - loss: 0.2777 -
val_loss: 0.2799
Epoch 24/50
235/235 [==============================] - 1s 6ms/step - loss: 0.2774 -
val_loss: 0.2796
Epoch 25/50
```

```
235/235 [==============================] - 1s 6ms/step - loss: 0.2772 -
val_loss: 0.2793
Epoch 26/50
235/235 [==============================] - 1s 6ms/step - loss: 0.2769 -
val_loss: 0.2794
Epoch 27/50
235/235 [==============================] - 1s 6ms/step - loss: 0.2766 -
val_loss: 0.2788
Epoch 28/50
235/235 [==============================] - 1s 6ms/step - loss: 0.2764 -
val_loss: 0.2789
Epoch 29/50
235/235 [==============================] - 1s 6ms/step - loss: 0.2762 -
val_loss: 0.2783
Epoch 30/50
235/235 [==============================] - 1s 6ms/step - loss: 0.2759 -
val_loss: 0.2783
Epoch 31/50
235/235 [==============================] - 1s 6ms/step - loss: 0.2758 -
val_loss: 0.2780
Epoch 32/50
235/235 [==============================] - 1s 6ms/step - loss: 0.2755 -
val_loss: 0.2786
Epoch 33/50
235/235 [==============================] - 1s 6ms/step - loss: 0.2753 -
val_loss: 0.2778
Epoch 34/50
235/235 [==============================] - 1s 6ms/step - loss: 0.2751 -
val_loss: 0.2775
Epoch 35/50
235/235 [==============================] - 1s 6ms/step - loss: 0.2749 -
val_loss: 0.2774
Epoch 36/50
235/235 [==============================] - 1s 6ms/step - loss: 0.2747 -
val_loss: 0.2772
Epoch 37/50
235/235 [==============================] - 1s 6ms/step - loss: 0.2746 -
val_loss: 0.2773
Epoch 38/50
235/235 [==============================] - 2s 6ms/step - loss: 0.2744 -
val_loss: 0.2770
Epoch 39/50
235/235 [==============================] - 2s 6ms/step - loss: 0.2742 -
val_loss: 0.2766
Epoch 40/50
235/235 [==============================] - 2s 7ms/step - loss: 0.2740 -
val_loss: 0.2765
Epoch 41/50
```

```
235/235 [==============================] - 1s 6ms/step - loss: 0.2738 -
val_loss: 0.2763
Epoch 42/50
235/235 [==============================] - 1s 6ms/step - loss: 0.2737 -
val_loss: 0.2761
Epoch 43/50
235/235 [==============================] - 1s 6ms/step - loss: 0.2736 -
val_loss: 0.2761
Epoch 44/50
235/235 [==============================] - 1s 6ms/step - loss: 0.2735 -
val_loss: 0.2761
Epoch 45/50
235/235 [==============================] - 1s 6ms/step - loss: 0.2733 -
val_loss: 0.2758
Epoch 46/50
235/235 [==============================] - 1s 6ms/step - loss: 0.2733 -
val_loss: 0.2756
Epoch 47/50
235/235 [==============================] - 1s 6ms/step - loss: 0.2731 -
val_loss: 0.2756
Epoch 48/50
235/235 [==============================] - 1s 6ms/step - loss: 0.2731 -
val_loss: 0.2756
Epoch 49/50
235/235 [==============================] - 1s 6ms/step - loss: 0.2729 -
val_loss: 0.2754
Epoch 50/50
235/235 [==============================] - 1s 6ms/step - loss: 0.2728 -
val_loss: 0.2753
```



6 3.4 Auto-encodeur convolutif

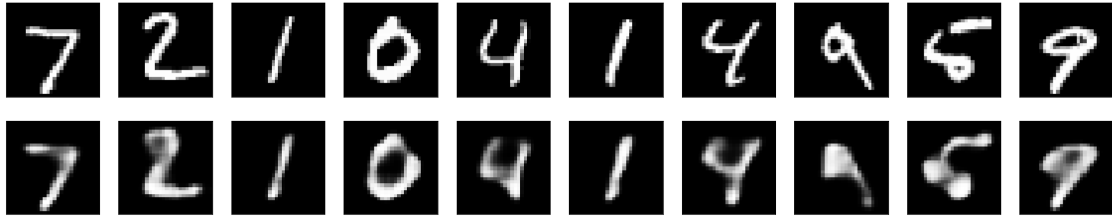
[ ]:
```
from tensorflow.keras.layers import Input, Dense, Conv2D, MaxPooling2D,␣
↪UpSampling2D
from tensorflow.keras.models import Model
from tensorflow.keras import backend as K
input_img = Input(shape=(28, 28, 1))
```

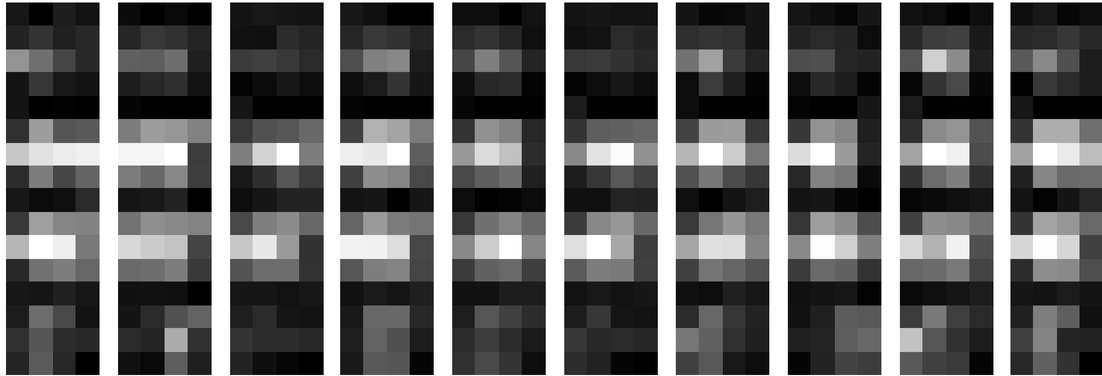
```
# adapt this if using `channels_first` image data format
x = Conv2D(8, (3, 3), activation='relu', padding='same')(input_img)
x = MaxPooling2D((2, 2), padding='same')(x)
x = Conv2D(4, (3, 3), activation='relu', padding='same')(x)
x = MaxPooling2D((2, 2), padding='same')(x)
x = Conv2D(4, (3, 3), activation='relu', padding='same')(x)
encoded = MaxPooling2D((2, 2), padding='same')(x)
# at this point the representation is (4, 4, 8) i.e. 128-dimensional
x = Conv2D(4, (3, 3), activation='relu', padding='same')(encoded)
x = UpSampling2D((2, 2))(x)
x = Conv2D(4, (3, 3), activation='relu', padding='same')(x)
x = UpSampling2D((2, 2))(x)
x = Conv2D(8, (3, 3), activation='relu')(x)
x = UpSampling2D((2, 2))(x)
decoded = Conv2D(1, (3, 3), activation='sigmoid', padding='same')(x)
autoencoder = Model(input_img, decoded)
autoencoder.compile(optimizer='adam', loss='binary_crossentropy')
autoencoder.summary()

from tensorflow.keras.datasets import mnist
import numpy as np
(x_train, _), (x_test, _) = mnist.load_data()
x_train = x_train.astype('float32') / 255.
x_test = x_test.astype('float32') / 255.
x_train = np.reshape(x_train, (len(x_train), 28, 28, 1))
# adapt this if using `channels_first` image data format
x_test = np.reshape(x_test, (len(x_test), 28, 28, 1))
# adapt this if using `channels_first` image data format
autoencoder.fit(x_train, x_train,
                epochs=10,
                batch_size=64,
                shuffle=True,
                validation_data=(x_test, x_test),verbose=1)
autoencoder.save('autoenc_conv.h5')

# encode and decode some digits
# note that we take them from the *test* set
decoded_imgs = autoencoder.predict(x_test)
import matplotlib.pyplot as plt
n = 10 # how many digits we will display
plt.figure(figsize=(20, 4))
for i in range(n):
  # display original
  ax = plt.subplot(2, n, i + 1)
  plt.imshow(x_test[i].reshape(28, 28))
  plt.gray()
  ax.get_xaxis().set_visible(False)
```

```
  ax.get_yaxis().set_visible(False)
  # display reconstruction
  ax = plt.subplot(2, n, i + 1 + n)
  plt.imshow(decoded_imgs[i].reshape(28, 28))
  plt.gray()
  ax.get_xaxis().set_visible(False)
  ax.get_yaxis().set_visible(False)
plt.show()

encoder = Model(input_img, encoded)
encoded_imgs = encoder.predict(x_test)
plt.figure(figsize=(10, 4), dpi=100)
for i in range(n):
  ax = plt.subplot(1, n, i + 1)
  plt.imshow(encoded_imgs[i].reshape(4,4*4).T) # 8 => 4
  plt.gray()
  ax.set_axis_off()
plt.show()
```

```
Model: "model_18"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_13 (InputLayer)       [(None, 28, 28, 1)]       0

 conv2d (Conv2D)             (None, 28, 28, 8)         80

 max_pooling2d (MaxPooling2D  (None, 14, 14, 8)        0
 )

 conv2d_1 (Conv2D)           (None, 14, 14, 4)         292

 max_pooling2d_1 (MaxPooling  (None, 7, 7, 4)          0
 2D)

 conv2d_2 (Conv2D)           (None, 7, 7, 4)           148

 max_pooling2d_2 (MaxPooling  (None, 4, 4, 4)          0
 2D)

 conv2d_3 (Conv2D)           (None, 4, 4, 4)           148

 up_sampling2d (UpSampling2D  (None, 8, 8, 4)          0
 )

 conv2d_4 (Conv2D)           (None, 8, 8, 4)           148
```

```
 up_sampling2d_1 (UpSampling  (None, 16, 16, 4)        0
 2D)

 conv2d_5 (Conv2D)           (None, 14, 14, 8)         296

 up_sampling2d_2 (UpSampling  (None, 28, 28, 8)        0
 2D)

 conv2d_6 (Conv2D)           (None, 28, 28, 1)         73

=================================================================
Total params: 1,185
Trainable params: 1,185
Non-trainable params: 0
_________________________________________________________________
Epoch 1/10
938/938 [==============================] - 17s 9ms/step - loss: 0.2186 -
val_loss: 0.1616
Epoch 2/10
938/938 [==============================] - 8s 9ms/step - loss: 0.1538 -
val_loss: 0.1468
Epoch 3/10
938/938 [==============================] - 9s 9ms/step - loss: 0.1440 -
val_loss: 0.1402
Epoch 4/10
938/938 [==============================] - 8s 9ms/step - loss: 0.1388 -
val_loss: 0.1361
Epoch 5/10
938/938 [==============================] - 8s 9ms/step - loss: 0.1352 -
val_loss: 0.1334
Epoch 6/10
938/938 [==============================] - 8s 9ms/step - loss: 0.1325 -
val_loss: 0.1304
Epoch 7/10
938/938 [==============================] - 9s 9ms/step - loss: 0.1306 -
val_loss: 0.1289
Epoch 8/10
938/938 [==============================] - 9s 9ms/step - loss: 0.1294 -
val_loss: 0.1278
Epoch 9/10
938/938 [==============================] - 8s 9ms/step - loss: 0.1283 -
val_loss: 0.1269
Epoch 10/10
938/938 [==============================] - 8s 9ms/step - loss: 0.1273 -
val_loss: 0.1259
```





7 3.4.2 Exercice

Modifions les paramètres avec `epochs=40`, et `batch_size=256`.

Les résultats nous donnent une perte plus faible de 10% par rapport à 10 epochs et un bach_size de 64 : `loss: 0.1212 - val_loss: 0.1201`

[ ]:
```
from tensorflow.keras.layers import Input, Dense, Conv2D, MaxPooling2D,␣
 ↪UpSampling2D
from tensorflow.keras.models import Model
from tensorflow.keras import backend as K
input_img = Input(shape=(28, 28, 1))
# adapt this if using `channels_first` image data format 8=>64,4=>32
x = Conv2D(64, (3, 3), activation='relu', padding='same')(input_img)
x = MaxPooling2D((2, 2), padding='same')(x)
x = Conv2D(32, (3, 3), activation='relu', padding='same')(x)
x = MaxPooling2D((2, 2), padding='same')(x)
x = Conv2D(32, (3, 3), activation='relu', padding='same')(x)
encoded = MaxPooling2D((2, 2), padding='same')(x)
# at this point the representation is (4, 4, 8) i.e. 128-dimensional
x = Conv2D(32, (3, 3), activation='relu', padding='same')(encoded)
x = UpSampling2D((2, 2))(x)
x = Conv2D(32, (3, 3), activation='relu', padding='same')(x)
```

```
x = UpSampling2D((2, 2))(x)
x = Conv2D(64, (3, 3), activation='relu')(x)
x = UpSampling2D((2, 2))(x)
decoded = Conv2D(1, (3, 3), activation='sigmoid', padding='same')(x)
autoencoder = Model(input_img, decoded)
autoencoder.compile(optimizer='adam', loss='binary_crossentropy')
autoencoder.summary()

from tensorflow.keras.datasets import mnist
import numpy as np
(x_train, _), (x_test, _) = mnist.load_data()
x_train = x_train.astype('float32') / 255.
x_test = x_test.astype('float32') / 255.
x_train = np.reshape(x_train, (len(x_train), 28, 28, 1))
# adapt this if using `channels_first` image data format
x_test = np.reshape(x_test, (len(x_test), 28, 28, 1))
# adapt this if using `channels_first` image data format
autoencoder.fit(x_train, x_train,
                epochs=40,
                batch_size=256,
                shuffle=True,
                validation_data=(x_test, x_test),verbose=1)
autoencoder.save('autoenc_conv.h5')

# encode and decode some digits
# note that we take them from the *test* set
decoded_imgs = autoencoder.predict(x_test)
import matplotlib.pyplot as plt
n = 10 # how many digits we will display
plt.figure(figsize=(20, 4))
for i in range(n):
  # display original
  ax = plt.subplot(2, n, i + 1)
  plt.imshow(x_test[i].reshape(28, 28))
  plt.gray()
  ax.get_xaxis().set_visible(False)
  ax.get_yaxis().set_visible(False)
  # display reconstruction
  ax = plt.subplot(2, n, i + 1 + n)
  plt.imshow(decoded_imgs[i].reshape(28, 28))
  plt.gray()
  ax.get_xaxis().set_visible(False)
  ax.get_yaxis().set_visible(False)
plt.show()

encoder = Model(input_img, encoded)
encoded_imgs = encoder.predict(x_test)
```

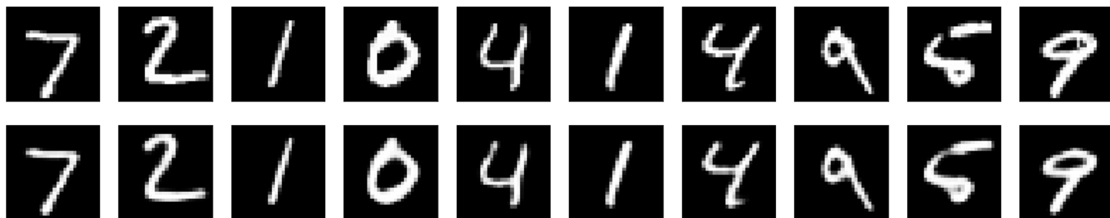
```python
plt.figure(figsize=(10, 4), dpi=100)
for i in range(n):
  ax = plt.subplot(1, n, i + 1)
  plt.imshow(encoded_imgs[i].reshape(8,8*8).T) # 8 => 4  # 8,8*8
  plt.gray()
  ax.set_axis_off()
plt.show()
```

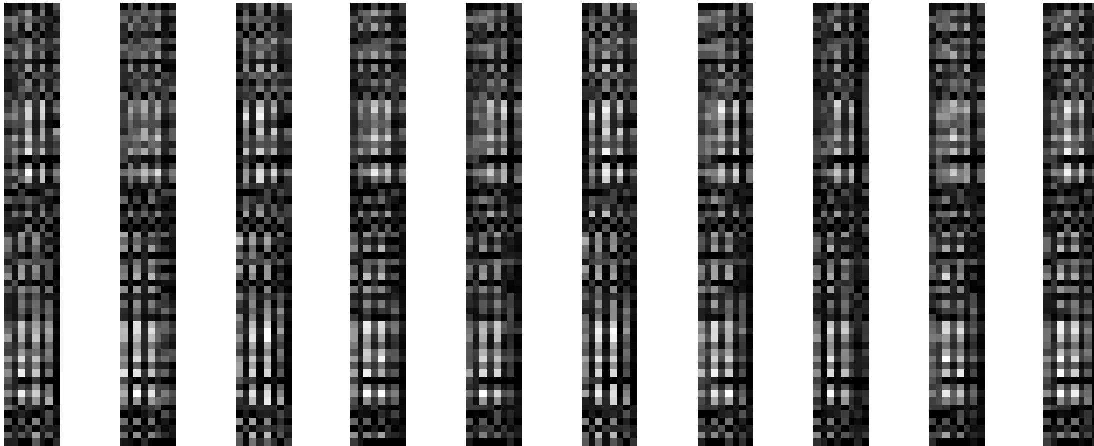
```
Model: "model_22"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_15 (InputLayer)       [(None, 28, 28, 1)]       0

 conv2d_14 (Conv2D)          (None, 28, 28, 64)        640

 max_pooling2d_6 (MaxPooling  (None, 14, 14, 64)       0
 2D)

 conv2d_15 (Conv2D)          (None, 14, 14, 32)        18464

 max_pooling2d_7 (MaxPooling  (None, 7, 7, 32)         0
 2D)

 conv2d_16 (Conv2D)          (None, 7, 7, 32)          9248

 max_pooling2d_8 (MaxPooling  (None, 4, 4, 32)         0
 2D)

 conv2d_17 (Conv2D)          (None, 4, 4, 32)          9248

 up_sampling2d_6 (UpSampling  (None, 8, 8, 32)         0
 2D)

 conv2d_18 (Conv2D)          (None, 8, 8, 32)          9248

 up_sampling2d_7 (UpSampling  (None, 16, 16, 32)       0
 2D)

 conv2d_19 (Conv2D)          (None, 14, 14, 64)        18496

 up_sampling2d_8 (UpSampling  (None, 28, 28, 64)       0
 2D)

 conv2d_20 (Conv2D)          (None, 28, 28, 1)         577

=================================================================
```

```
Total params: 65,921
Trainable params: 65,921
Non-trainable params: 0
_________________________________________________________________
Epoch 1/40
235/235 [==============================] - 11s 42ms/step - loss: 0.1811 -
val_loss: 0.1159
Epoch 2/40
235/235 [==============================] - 10s 41ms/step - loss: 0.1074 -
val_loss: 0.1009
Epoch 3/40
235/235 [==============================] - 9s 38ms/step - loss: 0.0966 -
val_loss: 0.0928
Epoch 4/40
235/235 [==============================] - 9s 38ms/step - loss: 0.0911 -
val_loss: 0.0881
Epoch 5/40
235/235 [==============================] - 9s 38ms/step - loss: 0.0877 -
val_loss: 0.0868
Epoch 6/40
235/235 [==============================] - 9s 38ms/step - loss: 0.0854 -
val_loss: 0.0836
Epoch 7/40
235/235 [==============================] - 9s 38ms/step - loss: 0.0836 -
val_loss: 0.0819
Epoch 8/40
235/235 [==============================] - 9s 38ms/step - loss: 0.0823 -
val_loss: 0.0813
Epoch 9/40
235/235 [==============================] - 9s 38ms/step - loss: 0.0813 -
val_loss: 0.0804
Epoch 10/40
235/235 [==============================] - 9s 38ms/step - loss: 0.0802 -
val_loss: 0.0796
Epoch 11/40
235/235 [==============================] - 9s 38ms/step - loss: 0.0795 -
val_loss: 0.0786
Epoch 12/40
235/235 [==============================] - 9s 38ms/step - loss: 0.0788 -
val_loss: 0.0781
Epoch 13/40
235/235 [==============================] - 9s 38ms/step - loss: 0.0781 -
val_loss: 0.0774
Epoch 14/40
235/235 [==============================] - 9s 38ms/step - loss: 0.0776 -
val_loss: 0.0763
Epoch 15/40
235/235 [==============================] - 9s 38ms/step - loss: 0.0772 -
```

```
val_loss: 0.0766
Epoch 16/40
235/235 [==============================] - 9s 37ms/step - loss: 0.0767 -
val_loss: 0.0759
Epoch 17/40
235/235 [==============================] - 9s 38ms/step - loss: 0.0763 -
val_loss: 0.0787
Epoch 18/40
235/235 [==============================] - 9s 38ms/step - loss: 0.0761 -
val_loss: 0.0751
Epoch 19/40
235/235 [==============================] - 9s 37ms/step - loss: 0.0756 -
val_loss: 0.0747
Epoch 20/40
235/235 [==============================] - 9s 37ms/step - loss: 0.0752 -
val_loss: 0.0744
Epoch 21/40
235/235 [==============================] - 9s 37ms/step - loss: 0.0750 -
val_loss: 0.0749
Epoch 22/40
235/235 [==============================] - 9s 37ms/step - loss: 0.0747 -
val_loss: 0.0739
Epoch 23/40
235/235 [==============================] - 9s 37ms/step - loss: 0.0745 -
val_loss: 0.0746
Epoch 24/40
235/235 [==============================] - 9s 37ms/step - loss: 0.0742 -
val_loss: 0.0735
Epoch 25/40
235/235 [==============================] - 9s 37ms/step - loss: 0.0740 -
val_loss: 0.0734
Epoch 26/40
235/235 [==============================] - 9s 37ms/step - loss: 0.0738 -
val_loss: 0.0739
Epoch 27/40
235/235 [==============================] - 9s 39ms/step - loss: 0.0736 -
val_loss: 0.0734
Epoch 28/40
235/235 [==============================] - 9s 38ms/step - loss: 0.0734 -
val_loss: 0.0743
Epoch 29/40
235/235 [==============================] - 9s 38ms/step - loss: 0.0731 -
val_loss: 0.0728
Epoch 30/40
235/235 [==============================] - 9s 37ms/step - loss: 0.0730 -
val_loss: 0.0728
Epoch 31/40
235/235 [==============================] - 9s 37ms/step - loss: 0.0729 -
```

```
val_loss: 0.0722
Epoch 32/40
235/235 [==============================] - 9s 37ms/step - loss: 0.0727 -
val_loss: 0.0719
Epoch 33/40
235/235 [==============================] - 9s 37ms/step - loss: 0.0725 -
val_loss: 0.0717
Epoch 34/40
235/235 [==============================] - 9s 37ms/step - loss: 0.0725 -
val_loss: 0.0716
Epoch 35/40
235/235 [==============================] - 9s 37ms/step - loss: 0.0723 -
val_loss: 0.0716
Epoch 36/40
235/235 [==============================] - 9s 37ms/step - loss: 0.0721 -
val_loss: 0.0713
Epoch 37/40
235/235 [==============================] - 9s 37ms/step - loss: 0.0719 -
val_loss: 0.0713
Epoch 38/40
235/235 [==============================] - 9s 37ms/step - loss: 0.0719 -
val_loss: 0.0713
Epoch 39/40
235/235 [==============================] - 9s 37ms/step - loss: 0.0718 -
val_loss: 0.0711
Epoch 40/40
235/235 [==============================] - 9s 38ms/step - loss: 0.0717 -
val_loss: 0.0718
```





Avec **fashion_MINST** : `loss: 0.2923` les résultats sont nettement moins bons, et augmenter le nombre d'epochs n'a pas diminié la perte.

[ ]:
```
from tensorflow.keras.layers import Input, Dense, Conv2D, MaxPooling2D,␣
↪UpSampling2D
from tensorflow.keras.models import Model
from tensorflow.keras import backend as K
input_img = Input(shape=(28, 28, 1))
# adapt this if using `channels_first` image data format: 8=>128, 4=64
x = Conv2D(64, (3, 3), activation='relu', padding='same')(input_img)
x = MaxPooling2D((2, 2), padding='same')(x)
x = Conv2D(32, (3, 3), activation='relu', padding='same')(x)
x = MaxPooling2D((2, 2), padding='same')(x)
x = Conv2D(32, (3, 3), activation='relu', padding='same')(x)
encoded = MaxPooling2D((2, 2), padding='same')(x)
# at this point the representation is (4, 4, 8) i.e. 128-dimensional
x = Conv2D(32, (3, 3), activation='relu', padding='same')(encoded)
x = UpSampling2D((2, 2))(x)
x = Conv2D(32, (3, 3), activation='relu', padding='same')(x)
x = UpSampling2D((2, 2))(x)
x = Conv2D(64, (3, 3), activation='relu')(x)
x = UpSampling2D((2, 2))(x)
decoded = Conv2D(1, (3, 3), activation='sigmoid', padding='same')(x)
autoencoder = Model(input_img, decoded)
autoencoder.compile(optimizer='adam', loss='binary_crossentropy')
autoencoder.summary()

from tensorflow.keras.datasets import fashion_mnist
import numpy as np
(x_train, _), (x_test, _) = fashion_mnist.load_data()
x_train = x_train.astype('float32') / 255.
```

```python
x_test = x_test.astype('float32') / 255.
x_train = np.reshape(x_train, (len(x_train), 28, 28, 1))
# adapt this if using `channels_first` image data format
x_test = np.reshape(x_test, (len(x_test), 28, 28, 1))
# adapt this if using `channels_first` image data format
autoencoder.fit(x_train, x_train,
                epochs=40,
                batch_size=256,
                shuffle=True,
                validation_data=(x_test, x_test),verbose=1)
autoencoder.save('autoenc_conv_fashion.h5')

# encode and decode some digits
# note that we take them from the *test* set
decoded_imgs = autoencoder.predict(x_test)
import matplotlib.pyplot as plt
n = 10 # how many digits we will display
plt.figure(figsize=(20, 4))
for i in range(n):
  # display original
  ax = plt.subplot(2, n, i + 1)
  plt.imshow(x_test[i].reshape(28, 28))
  plt.gray()
  ax.get_xaxis().set_visible(False)
  ax.get_yaxis().set_visible(False)
  # display reconstruction
  ax = plt.subplot(2, n, i + 1 + n)
  plt.imshow(decoded_imgs[i].reshape(28, 28))
  plt.gray()
  ax.get_xaxis().set_visible(False)
  ax.get_yaxis().set_visible(False)
plt.show()

encoder = Model(input_img, encoded)
encoded_imgs = encoder.predict(x_test)
plt.figure(figsize=(10, 4), dpi=100)
for i in range(n):
  ax = plt.subplot(1, n, i + 1)
  plt.imshow(encoded_imgs[i].reshape(8,8*8)) # 4,4*4).T) # 8 => 4
  plt.gray()
  ax.set_axis_off()
plt.show()
```

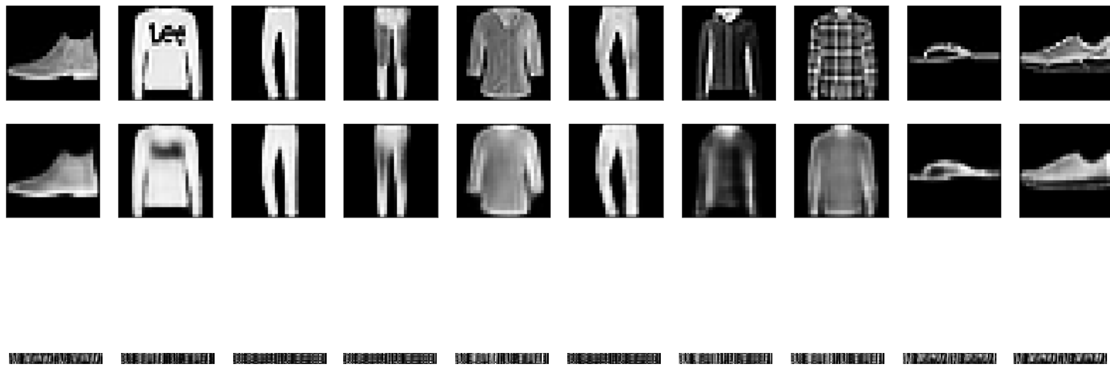
```
Model: "model_24"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
```

```
 input_16 (InputLayer)       [(None, 28, 28, 1)]       0

 conv2d_21 (Conv2D)          (None, 28, 28, 64)        640

 max_pooling2d_9 (MaxPooling  (None, 14, 14, 64)       0
 2D)

 conv2d_22 (Conv2D)          (None, 14, 14, 32)        18464

 max_pooling2d_10 (MaxPoolin  (None, 7, 7, 32)         0
 g2D)

 conv2d_23 (Conv2D)          (None, 7, 7, 32)          9248

 max_pooling2d_11 (MaxPoolin  (None, 4, 4, 32)         0
 g2D)

 conv2d_24 (Conv2D)          (None, 4, 4, 32)          9248

 up_sampling2d_9 (UpSampling  (None, 8, 8, 32)         0
 2D)

 conv2d_25 (Conv2D)          (None, 8, 8, 32)          9248

 up_sampling2d_10 (UpSamplin  (None, 16, 16, 32)       0
 g2D)

 conv2d_26 (Conv2D)          (None, 14, 14, 64)        18496

 up_sampling2d_11 (UpSamplin  (None, 28, 28, 64)       0
 g2D)

 conv2d_27 (Conv2D)          (None, 28, 28, 1)         577

=================================================================
Total params: 65,921
Trainable params: 65,921
Non-trainable params: 0
_________________________________________________________________
Epoch 1/40
235/235 [==============================] - 11s 42ms/step - loss: 0.3372 -
val_loss: 0.2993
Epoch 2/40
235/235 [==============================] - 10s 41ms/step - loss: 0.2900 -
val_loss: 0.2873
Epoch 3/40
235/235 [==============================] - 9s 40ms/step - loss: 0.2837 -
val_loss: 0.2833
```

```
Epoch 4/40
235/235 [==============================] - 9s 38ms/step - loss: 0.2799 -
val_loss: 0.2804
Epoch 5/40
235/235 [==============================] - 9s 37ms/step - loss: 0.2774 -
val_loss: 0.2786
Epoch 6/40
235/235 [==============================] - 9s 37ms/step - loss: 0.2754 -
val_loss: 0.2764
Epoch 7/40
235/235 [==============================] - 9s 36ms/step - loss: 0.2740 -
val_loss: 0.2752
Epoch 8/40
235/235 [==============================] - 9s 36ms/step - loss: 0.2725 -
val_loss: 0.2748
Epoch 9/40
235/235 [==============================] - 9s 38ms/step - loss: 0.2715 -
val_loss: 0.2728
Epoch 10/40
235/235 [==============================] - 9s 37ms/step - loss: 0.2706 -
val_loss: 0.2727
Epoch 11/40
235/235 [==============================] - 9s 37ms/step - loss: 0.2699 -
val_loss: 0.2712
Epoch 12/40
235/235 [==============================] - 9s 36ms/step - loss: 0.2691 -
val_loss: 0.2708
Epoch 13/40
235/235 [==============================] - 9s 37ms/step - loss: 0.2685 -
val_loss: 0.2702
Epoch 14/40
235/235 [==============================] - 9s 36ms/step - loss: 0.2681 -
val_loss: 0.2696
Epoch 15/40
235/235 [==============================] - 9s 37ms/step - loss: 0.2674 -
val_loss: 0.2689
Epoch 16/40
235/235 [==============================] - 9s 37ms/step - loss: 0.2671 -
val_loss: 0.2689
Epoch 17/40
235/235 [==============================] - 9s 37ms/step - loss: 0.2665 -
val_loss: 0.2682
Epoch 18/40
235/235 [==============================] - 9s 37ms/step - loss: 0.2662 -
val_loss: 0.2680
Epoch 19/40
235/235 [==============================] - 9s 37ms/step - loss: 0.2657 -
val_loss: 0.2685
```

```
Epoch 20/40
235/235 [==============================] - 9s 37ms/step - loss: 0.2655 -
val_loss: 0.2673
Epoch 21/40
235/235 [==============================] - 9s 39ms/step - loss: 0.2652 -
val_loss: 0.2671
Epoch 22/40
235/235 [==============================] - 9s 38ms/step - loss: 0.2649 -
val_loss: 0.2666
Epoch 23/40
235/235 [==============================] - 9s 38ms/step - loss: 0.2646 -
val_loss: 0.2665
Epoch 24/40
235/235 [==============================] - 9s 37ms/step - loss: 0.2644 -
val_loss: 0.2679
Epoch 25/40
235/235 [==============================] - 9s 37ms/step - loss: 0.2641 -
val_loss: 0.2659
Epoch 26/40
235/235 [==============================] - 9s 37ms/step - loss: 0.2639 -
val_loss: 0.2659
Epoch 27/40
235/235 [==============================] - 9s 38ms/step - loss: 0.2637 -
val_loss: 0.2654
Epoch 28/40
235/235 [==============================] - 9s 38ms/step - loss: 0.2635 -
val_loss: 0.2652
Epoch 29/40
235/235 [==============================] - 9s 37ms/step - loss: 0.2632 -
val_loss: 0.2651
Epoch 30/40
235/235 [==============================] - 9s 37ms/step - loss: 0.2631 -
val_loss: 0.2648
Epoch 31/40
235/235 [==============================] - 9s 37ms/step - loss: 0.2628 -
val_loss: 0.2646
Epoch 32/40
235/235 [==============================] - 9s 37ms/step - loss: 0.2629 -
val_loss: 0.2645
Epoch 33/40
235/235 [==============================] - 9s 37ms/step - loss: 0.2624 -
val_loss: 0.2646
Epoch 34/40
235/235 [==============================] - 9s 37ms/step - loss: 0.2624 -
val_loss: 0.2644
Epoch 35/40
235/235 [==============================] - 9s 37ms/step - loss: 0.2622 -
val_loss: 0.2643
```

```
Epoch 36/40
235/235 [==============================] - 9s 38ms/step - loss: 0.2621 -
val_loss: 0.2639
Epoch 37/40
235/235 [==============================] - 9s 37ms/step - loss: 0.2618 -
val_loss: 0.2639
Epoch 38/40
235/235 [==============================] - 9s 37ms/step - loss: 0.2619 -
val_loss: 0.2636
Epoch 39/40
235/235 [==============================] - 9s 37ms/step - loss: 0.2616 -
val_loss: 0.2647
Epoch 40/40
235/235 [==============================] - 9s 38ms/step - loss: 0.2616 -
val_loss: 0.2635
```



8 3.5 Application au débruitage d'image

Première execution avec un facteur de bruit de 50%.

```
[ ]: from tensorflow.keras.layers import Input, Dense, Conv2D, MaxPooling2D,␣
     ↪UpSampling2D
     from tensorflow.keras.models import Model
     import sys

     input_img = Input(shape=(28, 28, 1)) # adapt this if using `channels_first`␣
     ↪image data format

     x = Conv2D(8, (3, 3), activation='relu', padding='same')(input_img)
     x = MaxPooling2D((2, 2), padding='same')(x)
     x = Conv2D(4, (3, 3), activation='relu', padding='same')(x)
     x = MaxPooling2D((2, 2), padding='same')(x)
     x = Conv2D(4, (3, 3), activation='relu', padding='same')(x)
```

```
encoded = MaxPooling2D((2, 2), padding='same')(x)

# at this point the representation is (4, 4, 8) i.e. 128-dimensional
x = Conv2D(4, (3, 3), activation='relu', padding='same')(encoded)
x = UpSampling2D((2, 2))(x)
x = Conv2D(4, (3, 3), activation='relu', padding='same')(x)
x = UpSampling2D((2, 2))(x)
x = Conv2D(8, (3, 3), activation='relu')(x)
x = UpSampling2D((2, 2))(x)
decoded = Conv2D(1, (3, 3), activation='sigmoid', padding='same')(x)

autoencoder = Model(input_img, decoded)
autoencoder.compile(optimizer='adam', loss='binary_crossentropy')
autoencoder.summary()

from tensorflow.keras.datasets import mnist
import numpy as np
(x_train, _), (x_test, _) = mnist.load_data()
x_train = x_train.astype('float32') / 255.
x_test = x_test.astype('float32') / 255.
x_train = np.reshape(x_train, (len(x_train), 28, 28, 1)) # adapt this if using␣
↪`channels_first` image data format
x_test = np.reshape(x_test, (len(x_test), 28, 28, 1)) # adapt this if using␣
↪`channels_first` image data format

noise_factor = 0.5 # set as argument
x_train_noisy = x_train + noise_factor * np.random.normal(loc=0.0, scale=1.0,␣
↪size=x_train.shape)
x_test_noisy = x_test + noise_factor * np.random.normal(loc=0.0, scale=1.0,␣
↪size=x_test.shape)
x_train_noisy = np.clip(x_train_noisy, 0., 1.)
x_test_noisy = np.clip(x_test_noisy, 0., 1.)
autoencoder.fit(x_train_noisy, x_train,
                epochs=10,
                batch_size=64,
                shuffle=True,
                validation_data=(x_test, x_test),verbose=1)

autoencoder.save('autoenc_conv_denoise.h5')
# Recreate the exact same model purely from the file
new_model = keras.models.load_model('autoenc_conv_denoise.h5')
# encode and decode some digits
# note that we take them from the *test* set
#decoded_imgs = autoencoder.predict(x_test)
decoded_imgs = new_model.predict(x_test)
import matplotlib.pyplot as plt
n = 10
```

```
plt.figure(figsize=(10, 4), dpi=100)
for i in range(n):
  # display noisy
  ax = plt.subplot(2, n, i + 1)
  plt.imshow(x_test_noisy[i].reshape(28, 28))
  plt.gray()
  ax.set_axis_off()
  # display reconstruction
  ax = plt.subplot(2, n, i + n + 1)
  plt.imshow(decoded_imgs[i].reshape(28, 28))
  plt.gray()
  ax.set_axis_off()
plt.show()
```

Avec **fashion__MINST** : `loss: 0.3212` les résultats sont nettement moins bien, et augmenter le nombre d'epochs n'a pas diminié la perte.

```
[ ]: from tensorflow.keras.layers import Input, Dense, Conv2D, MaxPooling2D,␣
     ↪UpSampling2D
from tensorflow.keras.models import Model
import sys

input_img = Input(shape=(28, 28, 1)) # adapt this if using `channels_first`␣
 ↪image data format

x = Conv2D(8, (3, 3), activation='relu', padding='same')(input_img)
x = MaxPooling2D((2, 2), padding='same')(x)
x = Conv2D(4, (3, 3), activation='relu', padding='same')(x)
x = MaxPooling2D((2, 2), padding='same')(x)
x = Conv2D(4, (3, 3), activation='relu', padding='same')(x)
encoded = MaxPooling2D((2, 2), padding='same')(x)

# at this point the representation is (4, 4, 8) i.e. 128-dimensional
x = Conv2D(4, (3, 3), activation='relu', padding='same')(encoded)
x = UpSampling2D((2, 2))(x)
x = Conv2D(4, (3, 3), activation='relu', padding='same')(x)
x = UpSampling2D((2, 2))(x)
x = Conv2D(8, (3, 3), activation='relu')(x)
x = UpSampling2D((2, 2))(x)
decoded = Conv2D(1, (3, 3), activation='sigmoid', padding='same')(x)

autoencoder = Model(input_img, decoded)
autoencoder.compile(optimizer='adam', loss='binary_crossentropy')
autoencoder.summary()

from tensorflow.keras.datasets import fashion_mnist
import numpy as np
```

```python
(x_train, _), (x_test, _) = fashion_mnist.load_data()
x_train = x_train.astype('float32') / 255.
x_test = x_test.astype('float32') / 255.
x_train = np.reshape(x_train, (len(x_train), 28, 28, 1)) # adapt this if using␣
↪`channels_first` image data format
x_test = np.reshape(x_test, (len(x_test), 28, 28, 1)) # adapt this if using␣
↪`channels_first` image data format

noise_factor = 0.5 # set as argument
x_train_noisy = x_train + noise_factor * np.random.normal(loc=0.0, scale=1.0,␣
↪size=x_train.shape)
x_test_noisy = x_test + noise_factor * np.random.normal(loc=0.0, scale=1.0,␣
↪size=x_test.shape)
x_train_noisy = np.clip(x_train_noisy, 0., 1.)
x_test_noisy = np.clip(x_test_noisy, 0., 1.)
autoencoder.fit(x_train_noisy, x_train,
                epochs=10,
                batch_size=64,
                shuffle=True,
                validation_data=(x_test, x_test),verbose=1)

autoencoder.save('autoenc_conv_denoise.h5')
# Recreate the exact same model purely from the file
new_model = keras.models.load_model('autoenc_conv_denoise.h5')
# encode and decode some digits
# note that we take them from the *test* set
#decoded_imgs = autoencoder.predict(x_test)
decoded_imgs = new_model.predict(x_test)
import matplotlib.pyplot as plt
n = 10
plt.figure(figsize=(10, 4), dpi=100)
for i in range(n):
  # display noisy
  ax = plt.subplot(2, n, i + 1)
  plt.imshow(x_test_noisy[i].reshape(28, 28))
  plt.gray()
  ax.set_axis_off()
  # display reconstruction
  ax = plt.subplot(2, n, i + n + 1)
  plt.imshow(decoded_imgs[i].reshape(28, 28))
  plt.gray()
  ax.set_axis_off()
plt.show()
```

Et `loss: 0.3160` avec 50 epochs et un batch_size de 256

[ ]:
```
from tensorflow.keras.layers import Input, Dense, Conv2D, MaxPooling2D,␣
↪UpSampling2D
from tensorflow.keras.models import Model
import sys

input_img = Input(shape=(28, 28, 1)) # adapt this if using `channels_first`␣
↪image data format

x = Conv2D(8, (3, 3), activation='relu', padding='same')(input_img)
x = MaxPooling2D((2, 2), padding='same')(x)
x = Conv2D(4, (3, 3), activation='relu', padding='same')(x)
x = MaxPooling2D((2, 2), padding='same')(x)
x = Conv2D(4, (3, 3), activation='relu', padding='same')(x)
encoded = MaxPooling2D((2, 2), padding='same')(x)

# at this point the representation is (4, 4, 8) i.e. 128-dimensional
x = Conv2D(4, (3, 3), activation='relu', padding='same')(encoded)
x = UpSampling2D((2, 2))(x)
x = Conv2D(4, (3, 3), activation='relu', padding='same')(x)
x = UpSampling2D((2, 2))(x)
x = Conv2D(8, (3, 3), activation='relu')(x)
x = UpSampling2D((2, 2))(x)
decoded = Conv2D(1, (3, 3), activation='sigmoid', padding='same')(x)

autoencoder = Model(input_img, decoded)
autoencoder.compile(optimizer='adam', loss='binary_crossentropy')
autoencoder.summary()

from tensorflow.keras.datasets import fashion_mnist
import numpy as np
(x_train, _), (x_test, _) = fashion_mnist.load_data()
x_train = x_train.astype('float32') / 255.
x_test = x_test.astype('float32') / 255.
x_train = np.reshape(x_train, (len(x_train), 28, 28, 1)) # adapt this if using␣
↪`channels_first` image data format
x_test = np.reshape(x_test, (len(x_test), 28, 28, 1)) # adapt this if using␣
↪`channels_first` image data format

noise_factor = 0.5 # set as argument
x_train_noisy = x_train + noise_factor * np.random.normal(loc=0.0, scale=1.0,␣
↪size=x_train.shape)
x_test_noisy = x_test + noise_factor * np.random.normal(loc=0.0, scale=1.0,␣
↪size=x_test.shape)
x_train_noisy = np.clip(x_train_noisy, 0., 1.)
x_test_noisy = np.clip(x_test_noisy, 0., 1.)
autoencoder.fit(x_train_noisy, x_train,
                epochs=50,
```

```
                 batch_size=256,
                 shuffle=True,
                 validation_data=(x_test, x_test),verbose=1)

autoencoder.save('autoenc_conv_denoise.h5')
# Recreate the exact same model purely from the file
new_model = keras.models.load_model('autoenc_conv_denoise.h5')
# encode and decode some digits
# note that we take them from the *test* set
#decoded_imgs = autoencoder.predict(x_test)
decoded_imgs = new_model.predict(x_test)
import matplotlib.pyplot as plt
n = 10
plt.figure(figsize=(10, 4), dpi=100)
for i in range(n):
  # display noisy
  ax = plt.subplot(2, n, i + 1)
  plt.imshow(x_test_noisy[i].reshape(28, 28))
  plt.gray()
  ax.set_axis_off()
  # display reconstruction
  ax = plt.subplot(2, n, i + n + 1)
  plt.imshow(decoded_imgs[i].reshape(28, 28))
  plt.gray()
  ax.set_axis_off()
plt.show()
```

9 3.5.2 Exercices

Le paramètre noise_factor permet de préciser le taux de bruitage.

- Lancez l'entraînement avec les valeurs plus importantes : 0.8, 0.9, voir 0.95 est observez le résultat de débruitage.

0.8 : loss: 0.1815, certains chiffres ne sont pas lisibles (2,5,9,7).

0.9 : loss: 0.1884, légèrement moins bien que précedement.

0.95 : loss: 0.1916, légèrement moins bien que précedement.

[ ]:
```
noise_factor = 0.8 # set as argument

x_train_noisy = x_train + noise_factor * np.random.normal(loc=0.0, scale=1.0,␣
 ↪size=x_train.shape)
x_test_noisy = x_test + noise_factor * np.random.normal(loc=0.0, scale=1.0,␣
 ↪size=x_test.shape)
x_train_noisy = np.clip(x_train_noisy, 0., 1.)
x_test_noisy = np.clip(x_test_noisy, 0., 1.)
```

```
autoencoder.fit(x_train_noisy, x_train,
                epochs=10,
                batch_size=64,
                shuffle=True,
                validation_data=(x_test, x_test),verbose=1)

autoencoder.save('autoenc_conv_denoise.h5')
# Recreate the exact same model purely from the file
new_model = keras.models.load_model('autoenc_conv_denoise.h5')
# encode and decode some digits
# note that we take them from the *test* set
#decoded_imgs = autoencoder.predict(x_test)
decoded_imgs = new_model.predict(x_test)
import matplotlib.pyplot as plt
n = 10
plt.figure(figsize=(10, 4), dpi=100)
for i in range(n):
  # display noisy
  ax = plt.subplot(2, n, i + 1)
  plt.imshow(x_test_noisy[i].reshape(28, 28))
  plt.gray()
  ax.set_axis_off()
  # display reconstruction
  ax = plt.subplot(2, n, i + n + 1)
  plt.imshow(decoded_imgs[i].reshape(28, 28))
  plt.gray()
  ax.set_axis_off()
plt.show()
```

[ ]:
```
noise_factor = 0.9 # set as argument

x_train_noisy = x_train + noise_factor * np.random.normal(loc=0.0, scale=1.0,␣
 ↪size=x_train.shape)
x_test_noisy = x_test + noise_factor * np.random.normal(loc=0.0, scale=1.0,␣
 ↪size=x_test.shape)
x_train_noisy = np.clip(x_train_noisy, 0., 1.)
x_test_noisy = np.clip(x_test_noisy, 0., 1.)
autoencoder.fit(x_train_noisy, x_train,
                epochs=10,
                batch_size=64,
                shuffle=True,
                validation_data=(x_test, x_test),verbose=1)

autoencoder.save('autoenc_conv_denoise.h5')
# Recreate the exact same model purely from the file
new_model = keras.models.load_model('autoenc_conv_denoise.h5')
# encode and decode some digits
```

```
# note that we take them from the *test* set
#decoded_imgs = autoencoder.predict(x_test)
decoded_imgs = new_model.predict(x_test)
import matplotlib.pyplot as plt
n = 10
plt.figure(figsize=(10, 4), dpi=100)
for i in range(n):
  # display noisy
  ax = plt.subplot(2, n, i + 1)
  plt.imshow(x_test_noisy[i].reshape(28, 28))
  plt.gray()
  ax.set_axis_off()
  # display reconstruction
  ax = plt.subplot(2, n, i + n + 1)
  plt.imshow(decoded_imgs[i].reshape(28, 28))
  plt.gray()
  ax.set_axis_off()
plt.show()
```

[ ]:

```
noise_factor = 0.95 # set as argument

x_train_noisy = x_train + noise_factor * np.random.normal(loc=0.0, scale=1.0,␣
 ↪size=x_train.shape)
x_test_noisy = x_test + noise_factor * np.random.normal(loc=0.0, scale=1.0,␣
 ↪size=x_test.shape)
x_train_noisy = np.clip(x_train_noisy, 0., 1.)
x_test_noisy = np.clip(x_test_noisy, 0., 1.)
autoencoder.fit(x_train_noisy, x_train,
                epochs=10,
                batch_size=64,
                shuffle=True,
                validation_data=(x_test, x_test),verbose=1)

autoencoder.save('autoenc_conv_denoise.h5')
# Recreate the exact same model purely from the file
new_model = keras.models.load_model('autoenc_conv_denoise.h5')
# encode and decode some digits
# note that we take them from the *test* set
#decoded_imgs = autoencoder.predict(x_test)
decoded_imgs = new_model.predict(x_test)
import matplotlib.pyplot as plt
n = 10
plt.figure(figsize=(10, 4), dpi=100)
for i in range(n):
  # display noisy
  ax = plt.subplot(2, n, i + 1)
  plt.imshow(x_test_noisy[i].reshape(28, 28))
```

```
    plt.gray()
    ax.set_axis_off()
    # display reconstruction
    ax = plt.subplot(2, n, i + n + 1)
    plt.imshow(decoded_imgs[i].reshape(28, 28))
    plt.gray()
    ax.set_axis_off()
plt.show()
```

10 3.5.3 Sujet final – inférence - avec modèles entraînés sur Tesla

Le programme suivant est une application complète permettant d'entraîner et de sauvegarder les nouveaux modèles avec ou sans débruitage sur deux datasets MNIST : chiffres (digit) et vêtements (fashion).

Notre application permet également de charger les modèles externes pour leurs inférence sur les mêmes datasets.

[ ]:
```
from tensorflow import keras
from tensorflow.keras.layers import Input, Dense, Conv2D, MaxPooling2D,␣
↪UpSampling2D
from tensorflow.keras.models import Model
from tensorflow.keras.datasets import fashion_mnist
import sys

ds=int(input("Give dataset[1,2,3,4] - digit,fashion,digit+noise,fashion+noise:
↪"))
ti=int(input("train or inference [1,2]: "))
mn=input("model_name to train/save or load/inference, .h5 will be added: ")

if ti==1:
  wf=int(input("Give width_factor to be multiplied by 8: "))
  ef=int(input("Give epoch_factor to be multiplied by 10: "))
  bf=int(input("Give batch_factor to be multiplied by 64: "))
if ds>2:
  nf=float(input("Give the noise_factor [0.0 to 0.99]: "))
if ti==1:
  input_img = Input(shape=(28, 28, 1)) # adapt this if using `channels_first` ␣
↪image data format
  x = Conv2D(wf*8, (3, 3), activation='relu', padding='same')(input_img)
  x = MaxPooling2D((2, 2), padding='same')(x)
  x = Conv2D(wf*4, (3, 3), activation='relu', padding='same')(x)
  x = MaxPooling2D((2, 2), padding='same')(x)
  x = Conv2D(wf*4, (3, 3), activation='relu', padding='same')(x)
  encoded = MaxPooling2D((2, 2), padding='same')(x)
  # at this point the representation is (4, 4, 8) i.e. 128-dimensional
  x = Conv2D(wf*4, (3, 3), activation='relu', padding='same')(encoded)
```

```
  x = UpSampling2D((2, 2))(x)
  x = Conv2D(wf*4, (3, 3), activation='relu', padding='same')(x)
  x = UpSampling2D((2, 2))(x)
  x = Conv2D(wf*8, (3, 3), activation='relu')(x)
  x = UpSampling2D((2, 2))(x)
  decoded = Conv2D(1, (3, 3), activation='sigmoid', padding='same')(x)
  autoencoder = Model(input_img, decoded)
  autoencoder.compile(optimizer='adam', loss='binary_crossentropy')
  autoencoder.summary()

from tensorflow.keras.datasets import mnist
import numpy as np
if ds==1 or ds==3:
  (x_train, _), (x_test, _) = mnist.load_data()
if ds==2 or ds==4:
  (x_train, _), (x_test, _) = fashion_mnist.load_data()
x_train = x_train.astype('float32') / 255.
x_test = x_test.astype('float32') / 255.

x_train = np.reshape(x_train, (len(x_train), 28, 28, 1)) # adapt this if using␣
 ↪`channels_first` image data format
x_test = np.reshape(x_test, (len(x_test), 28, 28, 1)) # adapt this if using␣
 ↪`channels_first` image data format
if ds>2:
  x_train_noisy = x_train + nf * np.random.normal(loc=0.0, scale=1.0,␣
 ↪size=x_train.shape)
  x_test_noisy = x_test + nf * np.random.normal(loc=0.0, scale=1.0, size=x_test.
 ↪shape)
  x_train_noisy = np.clip(x_train_noisy, 0., 1.)
  x_test_noisy = np.clip(x_test_noisy, 0., 1.)
if ti==1 and ds<3:
  autoencoder.fit(x_train, x_train,␣
 ↪epochs=ef*10,batch_size=bf*64,shuffle=True,validation_data=(x_test,␣
 ↪x_test),verbose=1)
  autoencoder.save(mn+'.h5')
if ti==1 and ds>2:
  autoencoder.fit(x_train_noisy, x_train,␣
 ↪epochs=ef*10,batch_size=bf*64,shuffle=True,validation_data=(x_test,␣
 ↪x_test),verbose=1)
  autoencoder.save(mn+'.h5')
if ti==2:
  autoencoder = keras.models.load_model(mn+'.h5')
# encode and decode some digits
# note that we take them from the *test* set
# decoded_imgs = autoencoder.predict(x_test)
if ds<3:
```

```
  decoded_imgs = autoencoder.predict(x_test)
if ds>2:
  decoded_imgs = autoencoder.predict(x_test_noisy)

import matplotlib.pyplot as plt
n = 10 # how many digits we will display
plt.figure(figsize=(20, 4))
for i in range(n):
  # display original
  ax = plt.subplot(2, n, i + 1)
  if ds<3:
    plt.imshow(x_test[i].reshape(28, 28))
  if ds>2:
    plt.imshow(x_test_noisy[i].reshape(28, 28))

  plt.gray()
  ax.get_xaxis().set_visible(False)
  ax.get_yaxis().set_visible(False)
  # display reconstruction
  ax = plt.subplot(2, n, i + 1 + n)
  plt.imshow(decoded_imgs[i].reshape(28, 28))
  plt.gray()
  ax.get_xaxis().set_visible(False)
  ax.get_yaxis().set_visible(False)
plt.show()

if ti==1:
  encoder = Model(input_img, encoded)
  encoded_imgs = encoder.predict(x_test)
  plt.figure(figsize=(10, 4), dpi=100)
  for i in range(n):
    ax = plt.subplot(1, n, i + 1)
    plt.imshow(encoded_imgs[i].reshape(4,4*4*wf).T) # nano max 4
    plt.gray()
    ax.set_axis_off()
plt.show()
```

11 3.5.3.3 Exercices

Avec l'application ci-dessus : 1. entraînement de modèles MINST et fashion_MINST sur la Tesla (10 epochs, chunk_size 64, width_factor 8)

2. débruitage des images MISNT et fashion_MISNT (noise factor 0.8)

[ ]:
```
from keras.layers import Input, Dense, Conv2D, MaxPooling2D, UpSampling2D
from keras.models import Model
from tensorflow import keras
```

```python
from keras.datasets import fashion_mnist
import sys

ds=int(input("Give dataset[1,2,3,4] - digit,fashion,digit+noise,fashion+noise:␣
↪"))
ti=int(input("train or inference [1,2]: "))
mn=input("model_name to train/save or load/inference, .h5 will be added: ")

if ti==1:
  wf=int(input("Give width_factor to be multiplied by 8: "))
  ef=int(input("Give epoch_factor to be multiplied by 10: "))
  bf=int(input("Give batch_factor to be multiplied by 64: "))
if ds>2:
  nf=float(input("Give the noise_factor [0.0 to 0.99]: "))
if ti==1:
  input_img = Input(shape=(28, 28, 1)) # adapt this if using `channels_first` ␣
↪image data format
  x = Conv2D(wf*8, (3, 3), activation='relu', padding='same')(input_img)
  x = MaxPooling2D((2, 2), padding='same')(x)
  x = Conv2D(wf*4, (3, 3), activation='relu', padding='same')(x)
  x = MaxPooling2D((2, 2), padding='same')(x)
  x = Conv2D(wf*4, (3, 3), activation='relu', padding='same')(x)
  encoded = MaxPooling2D((2, 2), padding='same')(x)
  # at this point the representation is (4, 4, 8) i.e. 128-dimensional
  x = Conv2D(wf*4, (3, 3), activation='relu', padding='same')(encoded)
  x = UpSampling2D((2, 2))(x)
  x = Conv2D(wf*4, (3, 3), activation='relu', padding='same')(x)
  x = UpSampling2D((2, 2))(x)
  x = Conv2D(wf*8, (3, 3), activation='relu')(x)
  x = UpSampling2D((2, 2))(x)
  decoded = Conv2D(1, (3, 3), activation='sigmoid', padding='same')(x)
  autoencoder = Model(input_img, decoded)
  autoencoder.compile(optimizer='adam', loss='binary_crossentropy')
  autoencoder.summary()

from keras.datasets import mnist
import numpy as np
if ds==1 or ds==3:
  (x_train, _), (x_test, _) = mnist.load_data()
if ds==2 or ds==4:
  (x_train, _), (x_test, _) = fashion_mnist.load_data()
x_train = x_train.astype('float32') / 255.
x_test = x_test.astype('float32') / 255.

x_train = np.reshape(x_train, (len(x_train), 28, 28, 1)) # adapt this if using␣
↪`channels_first` image data format
```

```
x_test = np.reshape(x_test, (len(x_test), 28, 28, 1)) # adapt this if using␣
 ↪`channels_first` image data format
if ds>2:
  x_train_noisy = x_train + nf * np.random.normal(loc=0.0, scale=1.0,␣
 ↪size=x_train.shape)
  x_test_noisy = x_test + nf * np.random.normal(loc=0.0, scale=1.0, size=x_test.
 ↪shape)
  x_train_noisy = np.clip(x_train_noisy, 0., 1.)
  x_test_noisy = np.clip(x_test_noisy, 0., 1.)
if ti==1 and ds<3:
  autoencoder.fit(x_train, x_train,␣
 ↪epochs=ef*10,batch_size=bf*64,shuffle=True,validation_data=(x_test,␣
 ↪x_test),verbose=1)
  autoencoder.save(mn+'.h5')
if ti==1 and ds>2:
  autoencoder.fit(x_train_noisy, x_train,␣
 ↪epochs=ef*10,batch_size=bf*64,shuffle=True,validation_data=(x_test,␣
 ↪x_test),verbose=1)
  autoencoder.save(mn+'.h5')
if ti==2:
  autoencoder = keras.models.load_model(mn+'.h5')
# encode and decode some digits
# note that we take them from the *test* set
# decoded_imgs = autoencoder.predict(x_test)
if ds<3:
  decoded_imgs = autoencoder.predict(x_test)
if ds>2:
  decoded_imgs = autoencoder.predict(x_test_noisy)

import matplotlib.pyplot as plt
n = 10 # how many digits we will display
plt.figure(figsize=(20, 4))
for i in range(n):
  # display original
  ax = plt.subplot(2, n, i + 1)
  if ds<3:
    plt.imshow(x_test[i].reshape(28, 28))
  if ds>2:
    plt.imshow(x_test_noisy[i].reshape(28, 28))

  plt.gray()
  ax.get_xaxis().set_visible(False)
  ax.get_yaxis().set_visible(False)
  # display reconstruction
  ax = plt.subplot(2, n, i + 1 + n)
  plt.imshow(decoded_imgs[i].reshape(28, 28))
  plt.gray()
```

```
  ax.get_xaxis().set_visible(False)
  ax.get_yaxis().set_visible(False)
plt.show()

if ti==1:
  encoder = Model(input_img, encoded)
  encoded_imgs = encoder.predict(x_test)
  plt.figure(figsize=(10, 4), dpi=100)
  for i in range(n):
    ax = plt.subplot(1, n, i + 1)
    plt.imshow(encoded_imgs[i].reshape(4,4*4*wf).T) # nano max 4
    plt.gray()
    ax.set_axis_off()
plt.show()
```

Trainning on MINST_digit

model_name to train/save or load/inference, .h5 will be added: autoenc_conv_MINST_tesla_1

Give width_factor to be multiplied by 8: 1

Give epoch_factor to be multiplied by 10: 1

Give batch_factor to be multiplied by 64: 1

We obtain : loss: 0.1197

[ ]:
```
from keras.layers import Input, Dense, Conv2D, MaxPooling2D, UpSampling2D
from keras.models import Model
from tensorflow import keras
from keras.datasets import fashion_mnist
import sys

ds=int(input("Give dataset[1,2,3,4] - digit,fashion,digit+noise,fashion+noise:
 ↪"))
ti=int(input("train or inference [1,2]: "))
mn=input("model_name to train/save or load/inference, .h5 will be added: ")

if ti==1:
  wf=int(input("Give width_factor to be multiplied by 8: "))
  ef=int(input("Give epoch_factor to be multiplied by 10: "))
  bf=int(input("Give batch_factor to be multiplied by 64: "))
if ds>2:
  nf=float(input("Give the noise_factor [0.0 to 0.99]: "))
if ti==1:
  input_img = Input(shape=(28, 28, 1)) # adapt this if using `channels_first` ␣
 ↪image data format
  x = Conv2D(wf*8, (3, 3), activation='relu', padding='same')(input_img)
  x = MaxPooling2D((2, 2), padding='same')(x)
  x = Conv2D(wf*4, (3, 3), activation='relu', padding='same')(x)
```

```
  x = MaxPooling2D((2, 2), padding='same')(x)
  x = Conv2D(wf*4, (3, 3), activation='relu', padding='same')(x)
  encoded = MaxPooling2D((2, 2), padding='same')(x)
  # at this point the representation is (4, 4, 8) i.e. 128-dimensional
  x = Conv2D(wf*4, (3, 3), activation='relu', padding='same')(encoded)
  x = UpSampling2D((2, 2))(x)
  x = Conv2D(wf*4, (3, 3), activation='relu', padding='same')(x)
  x = UpSampling2D((2, 2))(x)
  x = Conv2D(wf*8, (3, 3), activation='relu')(x)
  x = UpSampling2D((2, 2))(x)
  decoded = Conv2D(1, (3, 3), activation='sigmoid', padding='same')(x)
  autoencoder = Model(input_img, decoded)
  autoencoder.compile(optimizer='adam', loss='binary_crossentropy')
  autoencoder.summary()

from keras.datasets import mnist
import numpy as np
if ds==1 or ds==3:
  (x_train, _), (x_test, _) = mnist.load_data()
if ds==2 or ds==4:
  (x_train, _), (x_test, _) = fashion_mnist.load_data()
x_train = x_train.astype('float32') / 255.
x_test = x_test.astype('float32') / 255.

x_train = np.reshape(x_train, (len(x_train), 28, 28, 1)) # adapt this if using␣
↪`channels_first` image data format
x_test = np.reshape(x_test, (len(x_test), 28, 28, 1)) # adapt this if using␣
↪`channels_first` image data format
if ds>2:
  x_train_noisy = x_train + nf * np.random.normal(loc=0.0, scale=1.0,␣
↪size=x_train.shape)
  x_test_noisy = x_test + nf * np.random.normal(loc=0.0, scale=1.0, size=x_test.
↪shape)
  x_train_noisy = np.clip(x_train_noisy, 0., 1.)
  x_test_noisy = np.clip(x_test_noisy, 0., 1.)
if ti==1 and ds<3:
  autoencoder.fit(x_train, x_train,␣
↪epochs=ef*10,batch_size=bf*64,shuffle=True,validation_data=(x_test,␣
↪x_test),verbose=1)
  autoencoder.save(mn+'.h5')
if ti==1 and ds>2:
  autoencoder.fit(x_train_noisy, x_train,␣
↪epochs=ef*10,batch_size=bf*64,shuffle=True,validation_data=(x_test,␣
↪x_test),verbose=1)
  autoencoder.save(mn+'.h5')
if ti==2:
```

```
  autoencoder = keras.models.load_model(mn+'.h5')
# encode and decode some digits
# note that we take them from the *test* set
# decoded_imgs = autoencoder.predict(x_test)
if ds<3:
  decoded_imgs = autoencoder.predict(x_test)
if ds>2:
  decoded_imgs = autoencoder.predict(x_test_noisy)

import matplotlib.pyplot as plt
n = 10 # how many digits we will display
plt.figure(figsize=(20, 4))
for i in range(n):
  # display original
  ax = plt.subplot(2, n, i + 1)
  if ds<3:
    plt.imshow(x_test[i].reshape(28, 28))
  if ds>2:
    plt.imshow(x_test_noisy[i].reshape(28, 28))

  plt.gray()
  ax.get_xaxis().set_visible(False)
  ax.get_yaxis().set_visible(False)
  # display reconstruction
  ax = plt.subplot(2, n, i + 1 + n)
  plt.imshow(decoded_imgs[i].reshape(28, 28))
  plt.gray()
  ax.get_xaxis().set_visible(False)
  ax.get_yaxis().set_visible(False)
plt.show()

if ti==1:
  encoder = Model(input_img, encoded)
  encoded_imgs = encoder.predict(x_test)
  plt.figure(figsize=(10, 4), dpi=100)
  for i in range(n):
    ax = plt.subplot(1, n, i + 1)
    plt.imshow(encoded_imgs[i].reshape(4,4*4*wf).T) # nano max 4
    plt.gray()
    ax.set_axis_off()
plt.show()
```

Trainning on MINST_fashion model_name to train/save or load/inference, .h5 will be added: autoenc_conv_fashionMINST_tesla_1

Give width_factor to be multiplied by 8: 1 Give epoch_factor to be multiplied by 10: 1 Give batch_factor to be multiplied by 64: 1

We obtain : loss: 0.2987

Donc la perte est bien plus grande avec le dataset fashion.

[ ]:
```
from keras.layers import Input, Dense, Conv2D, MaxPooling2D, UpSampling2D
from keras.models import Model
from tensorflow import keras
from keras.datasets import fashion_mnist
import sys

ds=int(input("Give dataset[1,2,3,4] - digit,fashion,digit+noise,fashion+noise:
 ↪"))
ti=int(input("train or inference [1,2]: "))
mn=input("model_name to train/save or load/inference, .h5 will be added: ")

if ti==1:
  wf=int(input("Give width_factor to be multiplied by 8: "))
  ef=int(input("Give epoch_factor to be multiplied by 10: "))
  bf=int(input("Give batch_factor to be multiplied by 64: "))
if ds>2:
  nf=float(input("Give the noise_factor [0.0 to 0.99]: "))
if ti==1:
  input_img = Input(shape=(28, 28, 1)) # adapt this if using `channels_first` ␣
 ↪image data format
  x = Conv2D(wf*8, (3, 3), activation='relu', padding='same')(input_img)
  x = MaxPooling2D((2, 2), padding='same')(x)
  x = Conv2D(wf*4, (3, 3), activation='relu', padding='same')(x)
  x = MaxPooling2D((2, 2), padding='same')(x)
  x = Conv2D(wf*4, (3, 3), activation='relu', padding='same')(x)
  encoded = MaxPooling2D((2, 2), padding='same')(x)
  # at this point the representation is (4, 4, 8) i.e. 128-dimensional
  x = Conv2D(wf*4, (3, 3), activation='relu', padding='same')(encoded)
  x = UpSampling2D((2, 2))(x)
  x = Conv2D(wf*4, (3, 3), activation='relu', padding='same')(x)
  x = UpSampling2D((2, 2))(x)
  x = Conv2D(wf*8, (3, 3), activation='relu')(x)
  x = UpSampling2D((2, 2))(x)
  decoded = Conv2D(1, (3, 3), activation='sigmoid', padding='same')(x)
  autoencoder = Model(input_img, decoded)
  autoencoder.compile(optimizer='adam', loss='binary_crossentropy')
  autoencoder.summary()

from keras.datasets import mnist
import numpy as np
if ds==1 or ds==3:
  (x_train, _), (x_test, _) = mnist.load_data()
if ds==2 or ds==4:
  (x_train, _), (x_test, _) = fashion_mnist.load_data()
```

```
x_train = x_train.astype('float32') / 255.
x_test = x_test.astype('float32') / 255.

x_train = np.reshape(x_train, (len(x_train), 28, 28, 1)) # adapt this if using `channels_first` image data format
x_test = np.reshape(x_test, (len(x_test), 28, 28, 1)) # adapt this if using `channels_first` image data format
if ds>2:
  x_train_noisy = x_train + nf * np.random.normal(loc=0.0, scale=1.0, size=x_train.shape)
  x_test_noisy = x_test + nf * np.random.normal(loc=0.0, scale=1.0, size=x_test.shape)
  x_train_noisy = np.clip(x_train_noisy, 0., 1.)
  x_test_noisy = np.clip(x_test_noisy, 0., 1.)
if ti==1 and ds<3:
  autoencoder.fit(x_train, x_train, epochs=ef*10,batch_size=bf*64,shuffle=True,validation_data=(x_test, x_test),verbose=1)
  autoencoder.save(mn+'.h5')
if ti==1 and ds>2:
  autoencoder.fit(x_train_noisy, x_train, epochs=ef*10,batch_size=bf*64,shuffle=True,validation_data=(x_test, x_test),verbose=1)
  autoencoder.save(mn+'.h5')
if ti==2:
  autoencoder = keras.models.load_model(mn+'.h5')
# encode and decode some digits
# note that we take them from the *test* set
# decoded_imgs = autoencoder.predict(x_test)
if ds<3:
  decoded_imgs = autoencoder.predict(x_test)
if ds>2:
  decoded_imgs = autoencoder.predict(x_test_noisy)

import matplotlib.pyplot as plt
n = 10 # how many digits we will display
plt.figure(figsize=(20, 4))
for i in range(n):
  # display original
  ax = plt.subplot(2, n, i + 1)
  if ds<3:
    plt.imshow(x_test[i].reshape(28, 28))
  if ds>2:
    plt.imshow(x_test_noisy[i].reshape(28, 28))

  plt.gray()
```

```
  ax.get_xaxis().set_visible(False)
  ax.get_yaxis().set_visible(False)
  # display reconstruction
  ax = plt.subplot(2, n, i + 1 + n)
  plt.imshow(decoded_imgs[i].reshape(28, 28))
  plt.gray()
  ax.get_xaxis().set_visible(False)
  ax.get_yaxis().set_visible(False)
plt.show()

if ti==1:
  encoder = Model(input_img, encoded)
  encoded_imgs = encoder.predict(x_test)
  plt.figure(figsize=(10, 4), dpi=100)
  for i in range(n):
    ax = plt.subplot(1, n, i + 1)
    plt.imshow(encoded_imgs[i].reshape(4,4*4*wf).T) # nano max 4
    plt.gray()
    ax.set_axis_off()
plt.show()
```

Avec **fashion_MINST** : Paramètres avec `epochs=50`, et `batch_size=256`, `width_size=80` et un `noise_factor=0.8`

Nous obtenons `loss: 0.3017`

Ce qui est une perte assez équivalente à celle pour la configuration : `epochs=10`, et `batch_size=64`, `width_size=8` et un `noise_factor=0.5`

[ ]:
```
from keras.layers import Input, Dense, Conv2D, MaxPooling2D, UpSampling2D
from keras.models import Model
from tensorflow import keras
from keras.datasets import fashion_mnist
import sys

ds=int(input("Give dataset[1,2,3,4] - digit,fashion,digit+noise,fashion+noise:
 ↪"))
ti=int(input("train or inference [1,2]: "))
mn=input("model_name to train/save or load/inference, .h5 will be added: ")

if ti==1:
  wf=int(input("Give width_factor to be multiplied by 8: "))
  ef=int(input("Give epoch_factor to be multiplied by 10: "))
  bf=int(input("Give batch_factor to be multiplied by 64: "))
if ds>2:
  nf=float(input("Give the noise_factor [0.0 to 0.99]: "))
if ti==1:
```

```
  input_img = Input(shape=(28, 28, 1)) # adapt this if using `channels_first` ␣
↪image data format
  x = Conv2D(wf*8, (3, 3), activation='relu', padding='same')(input_img)
  x = MaxPooling2D((2, 2), padding='same')(x)
  x = Conv2D(wf*4, (3, 3), activation='relu', padding='same')(x)
  x = MaxPooling2D((2, 2), padding='same')(x)
  x = Conv2D(wf*4, (3, 3), activation='relu', padding='same')(x)
  encoded = MaxPooling2D((2, 2), padding='same')(x)
  # at this point the representation is (4, 4, 8) i.e. 128-dimensional
  x = Conv2D(wf*4, (3, 3), activation='relu', padding='same')(encoded)
  x = UpSampling2D((2, 2))(x)
  x = Conv2D(wf*4, (3, 3), activation='relu', padding='same')(x)
  x = UpSampling2D((2, 2))(x)
  x = Conv2D(wf*8, (3, 3), activation='relu')(x)
  x = UpSampling2D((2, 2))(x)
  decoded = Conv2D(1, (3, 3), activation='sigmoid', padding='same')(x)
  autoencoder = Model(input_img, decoded)
  autoencoder.compile(optimizer='adam', loss='binary_crossentropy')
  autoencoder.summary()

from keras.datasets import mnist
import numpy as np
if ds==1 or ds==3:
  (x_train, _), (x_test, _) = mnist.load_data()
if ds==2 or ds==4:
  (x_train, _), (x_test, _) = fashion_mnist.load_data()
x_train = x_train.astype('float32') / 255.
x_test = x_test.astype('float32') / 255.

x_train = np.reshape(x_train, (len(x_train), 28, 28, 1)) # adapt this if using␣
↪`channels_first` image data format
x_test = np.reshape(x_test, (len(x_test), 28, 28, 1)) # adapt this if using␣
↪`channels_first` image data format
if ds>2:
  x_train_noisy = x_train + nf * np.random.normal(loc=0.0, scale=1.0,␣
↪size=x_train.shape)
  x_test_noisy = x_test + nf * np.random.normal(loc=0.0, scale=1.0, size=x_test.
↪shape)
  x_train_noisy = np.clip(x_train_noisy, 0., 1.)
  x_test_noisy = np.clip(x_test_noisy, 0., 1.)
if ti==1 and ds<3:
  autoencoder.fit(x_train, x_train,␣
↪epochs=ef*10,batch_size=bf*64,shuffle=True,validation_data=(x_test,␣
↪x_test),verbose=1)
  autoencoder.save(mn+'.h5')
if ti==1 and ds>2:
```

```
  autoencoder.fit(x_train_noisy, x_train,␣
↪epochs=ef*10,batch_size=bf*64,shuffle=True,validation_data=(x_test,␣
↪x_test),verbose=1)
  autoencoder.save(mn+'.h5')
if ti==2:
  autoencoder = keras.models.load_model(mn+'.h5')
# encode and decode some digits
# note that we take them from the *test* set
# decoded_imgs = autoencoder.predict(x_test)
if ds<3:
  decoded_imgs = autoencoder.predict(x_test)
if ds>2:
  decoded_imgs = autoencoder.predict(x_test_noisy)

import matplotlib.pyplot as plt
n = 10 # how many digits we will display
plt.figure(figsize=(20, 4))
for i in range(n):
  # display original
  ax = plt.subplot(2, n, i + 1)
  if ds<3:
    plt.imshow(x_test[i].reshape(28, 28))
  if ds>2:
    plt.imshow(x_test_noisy[i].reshape(28, 28))

  plt.gray()
  ax.get_xaxis().set_visible(False)
  ax.get_yaxis().set_visible(False)
  # display reconstruction
  ax = plt.subplot(2, n, i + 1 + n)
  plt.imshow(decoded_imgs[i].reshape(28, 28))
  plt.gray()
  ax.get_xaxis().set_visible(False)
  ax.get_yaxis().set_visible(False)
plt.show()

if ti==1:
  encoder = Model(input_img, encoded)
  encoded_imgs = encoder.predict(x_test)
  plt.figure(figsize=(10, 4), dpi=100)
  for i in range(n):
    ax = plt.subplot(1, n, i + 1)
    plt.imshow(encoded_imgs[i].reshape(4,4*4*wf).T) # nano max 4
    plt.gray()
    ax.set_axis_off()
plt.show()
```