The power of auto-encoders

Introduction

With this mini project we wanted to dive into the world of autoencoders. We hope that thanks to it you will understand how they work and what we can do with them.

An autoencoder is a type of neural network that is designed to reconstruct its input data by learning a compressed representation of the input data, known as the latent representation. This latent representation is typically a lower-dimensional space compared to the input data, and it is encoded in a latent vector. The autoencoder has two components: an encoder and a decoder. The encoder takes the input image and maps it to the latent space. The decoder takes the latent vector associated to this image and maps it back to the original space, reconstructing a new image, similar to the input one.

The goal of the auto-encoder is to learn a good latent representation of the input data such that the reconstructed data is as close as possible to the original data.

To clarify, an auto-encoder works like an image compression process.

Image compression : We take an image and we compress it using an encoding scheme that reduces its memory cost. Finally, we can decode the compress version of the image to obtain a new image similar to the original one.

Auto-encoder : We have an input image that pass through the 'encoder', the first part of the auto-encoder. This part is made of neural network layers that output a low dimensionality vector called the latent vector. This vector is a compress version of the input image. From this latent vector, the 'decoder' part tries reconstruct the input image.

When we train an auto-encoder, we fixe the dimension of the latent space, that is the length of the latent vector. At the begining, the output image is totally different from the input image (all the weights of the model have a random value). But, epoch after epoch, the model will build a better low-dimensionality representation of the input image. An auto-encoder uses unsupervised learning. We just give a dataset of images as input. No label are necessary. The precision of the model (or the loss function) is basically the difference between the output and the input images.

In the first part of this project we want to compress images and we test three different configuration using NN and CNN.

We then try to denoise images using the previous compression model.

Finally, we show a nice use of the latent space, by transfering the style of an image to another.

Dataset importation

As in many laboratories and projects, we are using the MNIST dataset. It contains 70000 images of handwritten digits between 0 and 9. The size of the images is 28x28.

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import fetch_openml
import progressbar
import time
from sklearn.model_selection import train_test_split
import keras
from tensorflow.keras.layers import Input, Dense
from tensorflow.keras.models import Model
import copy
from tensorflow.keras.layers import Input, Dense, Conv2D, MaxPooling2D, UpSampli
from tensorflow.keras.models import Model
from tensorflow.keras import backend as K
import functools
import os
from matplotlib import gridspec
import matplotlib.pylab as plt
import numpy as np
import tensorflow as tf
import tensorflow_hub as hub
```

In [ ]:
```
mnist = fetch_openml('mnist_784') #You can also use and test mnist_784 or Fashio
```

Basic handling of the dataset. Nothing really interesting here.

In [ ]:
```
datasetLength = int((len(mnist['data'])))

datasetTargets  = mnist['target'][0:datasetLength]
datasetData = mnist['data'].to_numpy()[:datasetLength]
nb_classes = 10

ratioTest = 0.2

X_train, X_test, y_train, y_test = train_test_split(datasetData, datasetTargets,

#Normalization
X_train = X_train.astype('float32')/255
X_test  = X_test.astype('float32')/255

#Reshaping from the 28x28 images to array of length 784
X_train  = X_train.reshape(len(X_train), 784)
X_test   = X_test.reshape(len(X_test), 784)

#Binary matrix representation for the label (the digit) instead of an integer
Y_train = keras.utils.np_utils.to_categorical(y_train, nb_classes)
Y_test = keras.utils.np_utils.to_categorical(y_test, nb_classes)
```

Here are some images of the dataset :

In [ ]:
```
plt.figure(figsize=(1,1)), plt.axis("off"), plt.gray(), plt.matshow(X_train[42].
plt.figure(figsize=(1,1)), plt.axis("off"), plt.matshow(X_train[85].reshape(28,
plt.figure(figsize=(1,1)), plt.axis("off"), plt.matshow(X_train[64].reshape(28,
plt.figure(figsize=(1,1)), plt.axis("off"), plt.matshow(X_train[850].reshape(28,
```

Out[ ]:
```
(<Figure size 100x100 with 1 Axes>,
 (0.0, 1.0, 0.0, 1.0),
 <matplotlib.image.AxesImage at 0x28ec0590d00>)
```

Auto-encoder using one dense layer

Here we build the auto-encoder. A dense layer is a fully connected layer. We use one dense layer for the encoder part, and one for the decoder part.

In [ ]:

```
# Input data size
original_dim = 784

## Hyperparameters ##

# We will try different size of the latent space. One image will be associated t
# The compression ratio of our model is then directly corelated to this value.
latent_dims = [8,16,32,64] # We use the letter z to refer to the value. '32' is

# Training parameters :
# We have a training dataset with 56000 images (80% of 70000). With a batch size
# The weights of the model are updated after each batch. Thus, one epoch corresp
# The model will be exposed to the whole training dataset 80 times, which corres
# We will see during the next parts of this project that the model is pretty rob

epochs=80
batch_size=250

## Building of each model, layer by layer ##
models = {}
for latent_dim in latent_dims:
    models[latent_dim] = {}

    input_img = Input(shape=(original_dim,))

    encoder_layer = Dense(latent_dim, activation='relu')
    encoded = encoder_layer(input_img)

    decoder_layer = Dense(original_dim, activation='sigmoid')
    decoded = decoder_layer(encoded)

    autoencoder = Model(input_img, decoded)
    autoencoder.compile(optimizer='adam', loss='binary_crossentropy') # you can

    encoder = Model(input_img, encoded)
    models[latent_dim]["encoder"] = encoder

    encoded_input = Input(shape=(latent_dim))
    decoder = Model(encoded_input, decoder_layer(encoded_input))
    models[latent_dim]["decoder"] = decoder

    ## We display the model caracteristics
    autoencoder.summary()

    models[latent_dim]["autoencoder"] = autoencoder
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 784)]             0

 dense (Dense)               (None, 8)                 6280

 dense_1 (Dense)             (None, 784)               7056

=================================================================
Total params: 13,336
Trainable params: 13,336
Non-trainable params: 0
_________________________________________________________________
Model: "model_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_3 (InputLayer)        [(None, 784)]             0

 dense_2 (Dense)             (None, 16)                12560

 dense_3 (Dense)             (None, 784)               13328

=================================================================
Total params: 25,888
Trainable params: 25,888
Non-trainable params: 0
_________________________________________________________________
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
_________________________________________________________________
Model: "model_9"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_7 (InputLayer)        [(None, 784)]             0

 dense_6 (Dense)             (None, 64)                50240

 dense_7 (Dense)             (None, 784)               50960

=================================================================
Total params: 101,200
Trainable params: 101,200
Non-trainable params: 0
_________________________________________________________________
```

Let's train our model!

In a classification situation, we would have fit the image dataset with their label. But here, we see that both subsets (X_train and X_test) appears two times. Indeed, as mentionned before, we are using unsupervised learning. The output of an encoder is an image that should be compared with the input image. This is why we are fiting the datasets with themselves.

In [ ]:
```
for z in models:
    models[z]["logs"] = models[z]["autoencoder"].fit(X_train, X_train,epochs=epo
```

```
Epoch 1/80
224/224 [==============================] - 1s 2ms/step - loss: 0.3403 - val_los
s: 0.2545
Epoch 2/80
224/224 [==============================] - 0s 2ms/step - loss: 0.2334 - val_los
s: 0.2159
Epoch 3/80
224/224 [==============================] - 0s 2ms/step - loss: 0.2024 - val_los
s: 0.1932
Epoch 4/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1876 - val_los
s: 0.1847
Epoch 5/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1819 - val_los
s: 0.1811
Epoch 6/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1791 - val_los
s: 0.1788
Epoch 7/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1772 - val_los
s: 0.1771
Epoch 8/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1756 - val_los
s: 0.1756
Epoch 9/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1743 - val_los
s: 0.1744
Epoch 10/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1732 - val_los
s: 0.1733
Epoch 11/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1722 - val_los
s: 0.1723
Epoch 12/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1713 - val_los
s: 0.1715
Epoch 13/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1705 - val_los
s: 0.1707
Epoch 14/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1698 - val_los
s: 0.1700
Epoch 15/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1692 - val_los
s: 0.1694
Epoch 16/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1686 - val_los
s: 0.1688
Epoch 17/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1681 - val_los
s: 0.1684
Epoch 18/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1677 - val_los
s: 0.1680
Epoch 19/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1673 - val_los
s: 0.1676
Epoch 20/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1670 - val_los
s: 0.1673
```

```
Epoch 21/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1667 - val_los
s: 0.1670
Epoch 22/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1665 - val_los
s: 0.1669
Epoch 23/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1663 - val_los
s: 0.1666
Epoch 24/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1661 - val_los
s: 0.1664
Epoch 25/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1659 - val_los
s: 0.1663
Epoch 26/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1658 - val_los
s: 0.1662
Epoch 27/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1657 - val_los
s: 0.1660
Epoch 28/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1655 - val_los
s: 0.1659
Epoch 29/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1654 - val_los
s: 0.1658
Epoch 30/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1653 - val_los
s: 0.1658
Epoch 31/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1653 - val_los
s: 0.1656
Epoch 32/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1652 - val_los
s: 0.1656
Epoch 33/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1651 - val_los
s: 0.1656
Epoch 34/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1650 - val_los
s: 0.1654
Epoch 35/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1650 - val_los
s: 0.1654
Epoch 36/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1649 - val_los
s: 0.1653
Epoch 37/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1648 - val_los
s: 0.1653
Epoch 38/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1648 - val_los
s: 0.1652
Epoch 39/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1647 - val_los
s: 0.1651
Epoch 40/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1647 - val_los
s: 0.1651
```

```
Epoch 41/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1646 - val_los
s: 0.1651
Epoch 42/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1646 - val_los
s: 0.1650
Epoch 43/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1646 - val_los
s: 0.1650
Epoch 44/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1645 - val_los
s: 0.1650
Epoch 45/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1645 - val_los
s: 0.1649
Epoch 46/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1644 - val_los
s: 0.1649
Epoch 47/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1644 - val_los
s: 0.1649
Epoch 48/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1644 - val_los
s: 0.1648
Epoch 49/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1644 - val_los
s: 0.1648
Epoch 50/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1643 - val_los
s: 0.1648
Epoch 51/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1643 - val_los
s: 0.1648
Epoch 52/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1643 - val_los
s: 0.1647
Epoch 53/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1643 - val_los
s: 0.1647
Epoch 54/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1643 - val_los
s: 0.1647
Epoch 55/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1642 - val_los
s: 0.1647
Epoch 56/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1642 - val_los
s: 0.1647
Epoch 57/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1642 - val_los
s: 0.1647
Epoch 58/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1642 - val_los
s: 0.1646
Epoch 59/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1641 - val_los
s: 0.1646
Epoch 60/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1641 - val_los
s: 0.1646
```

```
Epoch 61/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1641 - val_los
s: 0.1646
Epoch 62/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1641 - val_los
s: 0.1645
Epoch 63/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1641 - val_los
s: 0.1645
Epoch 64/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1641 - val_los
s: 0.1645
Epoch 65/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1641 - val_los
s: 0.1646
Epoch 66/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1640 - val_los
s: 0.1646
Epoch 67/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1640 - val_los
s: 0.1645
Epoch 68/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1640 - val_los
s: 0.1645
Epoch 69/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1640 - val_los
s: 0.1645
Epoch 70/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1640 - val_los
s: 0.1645
Epoch 71/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1640 - val_los
s: 0.1644
Epoch 72/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1640 - val_los
s: 0.1644
Epoch 73/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1640 - val_los
s: 0.1644
Epoch 74/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1640 - val_los
s: 0.1644
Epoch 75/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1640 - val_los
s: 0.1644
Epoch 76/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1639 - val_los
s: 0.1644
Epoch 77/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1639 - val_los
s: 0.1644
Epoch 78/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1639 - val_los
s: 0.1644
Epoch 79/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1639 - val_los
s: 0.1644
Epoch 80/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1639 - val_los
s: 0.1644
```

```
Epoch 1/80
224/224 [==============================] - 1s 2ms/step - loss: 0.3135 - val_los
s: 0.2292
Epoch 2/80
224/224 [==============================] - 0s 2ms/step - loss: 0.2021 - val_los
s: 0.1857
Epoch 3/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1754 - val_los
s: 0.1676
Epoch 4/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1612 - val_los
s: 0.1566
Epoch 5/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1524 - val_los
s: 0.1494
Epoch 6/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1465 - val_los
s: 0.1447
Epoch 7/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1429 - val_los
s: 0.1418
Epoch 8/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1403 - val_los
s: 0.1395
Epoch 9/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1383 - val_los
s: 0.1376
Epoch 10/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1365 - val_los
s: 0.1361
Epoch 11/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1350 - val_los
s: 0.1346
Epoch 12/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1337 - val_los
s: 0.1334
Epoch 13/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1326 - val_los
s: 0.1325
Epoch 14/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1318 - val_los
s: 0.1317
Epoch 15/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1312 - val_los
s: 0.1312
Epoch 16/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1307 - val_los
s: 0.1307
Epoch 17/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1303 - val_los
s: 0.1304
Epoch 18/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1300 - val_los
s: 0.1301
Epoch 19/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1297 - val_los
s: 0.1298
Epoch 20/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1295 - val_los
s: 0.1296
```

```
Epoch 21/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1293 - val_los
s: 0.1294
Epoch 22/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1291 - val_los
s: 0.1293
Epoch 23/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1289 - val_los
s: 0.1291
Epoch 24/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1288 - val_los
s: 0.1290
Epoch 25/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1286 - val_los
s: 0.1289
Epoch 26/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1285 - val_los
s: 0.1288
Epoch 27/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1284 - val_los
s: 0.1286
Epoch 28/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1283 - val_los
s: 0.1285
Epoch 29/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1282 - val_los
s: 0.1285
Epoch 30/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1281 - val_los
s: 0.1283
Epoch 31/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1280 - val_los
s: 0.1282
Epoch 32/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1279 - val_los
s: 0.1281
Epoch 33/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1278 - val_los
s: 0.1281
Epoch 34/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1278 - val_los
s: 0.1280
Epoch 35/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1277 - val_los
s: 0.1279
Epoch 36/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1276 - val_los
s: 0.1279
Epoch 37/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1276 - val_los
s: 0.1277
Epoch 38/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1275 - val_los
s: 0.1277
Epoch 39/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1275 - val_los
s: 0.1277
Epoch 40/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1274 - val_los
s: 0.1276
```

```
Epoch 41/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1274 - val_los
s: 0.1276
Epoch 42/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1273 - val_los
s: 0.1275
Epoch 43/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1273 - val_los
s: 0.1275
Epoch 44/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1272 - val_los
s: 0.1274
Epoch 45/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1272 - val_los
s: 0.1275
Epoch 46/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1271 - val_los
s: 0.1274
Epoch 47/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1271 - val_los
s: 0.1274
Epoch 48/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1271 - val_los
s: 0.1273
Epoch 49/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1270 - val_los
s: 0.1273
Epoch 50/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1270 - val_los
s: 0.1272
Epoch 51/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1270 - val_los
s: 0.1272
Epoch 52/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1269 - val_los
s: 0.1272
Epoch 53/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1269 - val_los
s: 0.1272
Epoch 54/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1269 - val_los
s: 0.1271
Epoch 55/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1268 - val_los
s: 0.1271
Epoch 56/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1268 - val_los
s: 0.1270
Epoch 57/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1268 - val_los
s: 0.1270
Epoch 58/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1267 - val_los
s: 0.1270
Epoch 59/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1267 - val_los
s: 0.1270
Epoch 60/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1267 - val_los
s: 0.1269
```

```
Epoch 61/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1267 - val_los
s: 0.1269
Epoch 62/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1266 - val_los
s: 0.1269
Epoch 63/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1266 - val_los
s: 0.1269
Epoch 64/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1266 - val_los
s: 0.1269
Epoch 65/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1266 - val_los
s: 0.1268
Epoch 66/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1266 - val_los
s: 0.1268
Epoch 67/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1265 - val_los
s: 0.1268
Epoch 68/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1265 - val_los
s: 0.1268
Epoch 69/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1265 - val_los
s: 0.1268
Epoch 70/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1265 - val_los
s: 0.1268
Epoch 71/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1264 - val_los
s: 0.1267
Epoch 72/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1264 - val_los
s: 0.1267
Epoch 73/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1264 - val_los
s: 0.1267
Epoch 74/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1264 - val_los
s: 0.1266
Epoch 75/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1264 - val_los
s: 0.1266
Epoch 76/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1264 - val_los
s: 0.1266
Epoch 77/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1263 - val_los
s: 0.1266
Epoch 78/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1263 - val_los
s: 0.1266
Epoch 79/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1263 - val_los
s: 0.1266
Epoch 80/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1263 - val_los
s: 0.1265
```

```
Epoch 1/80
224/224 [==============================] - 1s 2ms/step - loss: 0.2804 - val_los
s: 0.1965
Epoch 2/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1744 - val_los
s: 0.1586
Epoch 3/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1468 - val_los
s: 0.1375
Epoch 4/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1303 - val_los
s: 0.1251
Epoch 5/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1201 - val_los
s: 0.1165
Epoch 6/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1128 - val_los
s: 0.1104
Epoch 7/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1076 - val_los
s: 0.1059
Epoch 8/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1037 - val_los
s: 0.1025
Epoch 9/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1007 - val_los
s: 0.0999
Epoch 10/80
224/224 [==============================] - 0s 2ms/step - loss: 0.0985 - val_los
s: 0.0980
Epoch 11/80
224/224 [==============================] - 0s 2ms/step - loss: 0.0970 - val_los
s: 0.0966
Epoch 12/80
224/224 [==============================] - 0s 2ms/step - loss: 0.0959 - val_los
s: 0.0957
Epoch 13/80
224/224 [==============================] - 0s 2ms/step - loss: 0.0952 - val_los
s: 0.0952
Epoch 14/80
224/224 [==============================] - 0s 2ms/step - loss: 0.0947 - val_los
s: 0.0948
Epoch 15/80
224/224 [==============================] - 0s 2ms/step - loss: 0.0943 - val_los
s: 0.0945
Epoch 16/80
224/224 [==============================] - 0s 2ms/step - loss: 0.0941 - val_los
s: 0.0943
Epoch 17/80
224/224 [==============================] - 0s 2ms/step - loss: 0.0939 - val_los
s: 0.0940
Epoch 18/80
224/224 [==============================] - 0s 2ms/step - loss: 0.0937 - val_los
s: 0.0939
Epoch 19/80
224/224 [==============================] - 0s 2ms/step - loss: 0.0936 - val_los
s: 0.0938
Epoch 20/80
224/224 [==============================] - 0s 2ms/step - loss: 0.0934 - val_los
s: 0.0938
```

```
Epoch 21/80
224/224 [==============================] - 0s 2ms/step - loss: 0.0933 - val_los
s: 0.0936
Epoch 22/80
224/224 [==============================] - 0s 2ms/step - loss: 0.0932 - val_los
s: 0.0936
Epoch 23/80
224/224 [==============================] - 0s 2ms/step - loss: 0.0932 - val_los
s: 0.0935
Epoch 24/80
224/224 [==============================] - 0s 2ms/step - loss: 0.0931 - val_los
s: 0.0934
Epoch 25/80
224/224 [==============================] - 0s 2ms/step - loss: 0.0930 - val_los
s: 0.0934
Epoch 26/80
224/224 [==============================] - 0s 2ms/step - loss: 0.0930 - val_los
s: 0.0934
Epoch 27/80
224/224 [==============================] - 0s 2ms/step - loss: 0.0929 - val_los
s: 0.0932
Epoch 28/80
224/224 [==============================] - 0s 2ms/step - loss: 0.0929 - val_los
s: 0.0932
Epoch 29/80
224/224 [==============================] - 0s 2ms/step - loss: 0.0928 - val_los
s: 0.0932
Epoch 30/80
224/224 [==============================] - 0s 2ms/step - loss: 0.0928 - val_los
s: 0.0931
Epoch 31/80
224/224 [==============================] - 0s 2ms/step - loss: 0.0928 - val_los
s: 0.0931
Epoch 32/80
224/224 [==============================] - 0s 2ms/step - loss: 0.0927 - val_los
s: 0.0931
Epoch 33/80
224/224 [==============================] - 0s 2ms/step - loss: 0.0927 - val_los
s: 0.0930
Epoch 34/80
224/224 [==============================] - 0s 2ms/step - loss: 0.0927 - val_los
s: 0.0930
Epoch 35/80
224/224 [==============================] - 0s 2ms/step - loss: 0.0926 - val_los
s: 0.0930
Epoch 36/80
224/224 [==============================] - 0s 2ms/step - loss: 0.0926 - val_los
s: 0.0930
Epoch 37/80
224/224 [==============================] - 0s 2ms/step - loss: 0.0926 - val_los
s: 0.0930
Epoch 38/80
224/224 [==============================] - 0s 2ms/step - loss: 0.0926 - val_los
s: 0.0930
Epoch 39/80
224/224 [==============================] - 0s 2ms/step - loss: 0.0926 - val_los
s: 0.0929
Epoch 40/80
224/224 [==============================] - 0s 2ms/step - loss: 0.0926 - val_los
s: 0.0930
```

```
Epoch 41/80
224/224 [==============================] - 0s 2ms/step - loss: 0.0925 - val_los
s: 0.0930
Epoch 42/80
224/224 [==============================] - 0s 2ms/step - loss: 0.0925 - val_los
s: 0.0929
Epoch 43/80
224/224 [==============================] - 0s 2ms/step - loss: 0.0925 - val_los
s: 0.0929
Epoch 44/80
224/224 [==============================] - 0s 2ms/step - loss: 0.0925 - val_los
s: 0.0929
Epoch 45/80
224/224 [==============================] - 0s 2ms/step - loss: 0.0925 - val_los
s: 0.0929
Epoch 46/80
224/224 [==============================] - 0s 2ms/step - loss: 0.0925 - val_los
s: 0.0928
Epoch 47/80
224/224 [==============================] - 0s 2ms/step - loss: 0.0924 - val_los
s: 0.0929
Epoch 48/80
224/224 [==============================] - 0s 2ms/step - loss: 0.0924 - val_los
s: 0.0928
Epoch 49/80
224/224 [==============================] - 0s 2ms/step - loss: 0.0924 - val_los
s: 0.0928
Epoch 50/80
224/224 [==============================] - 0s 2ms/step - loss: 0.0924 - val_los
s: 0.0929
Epoch 51/80
224/224 [==============================] - 0s 2ms/step - loss: 0.0924 - val_los
s: 0.0929
Epoch 52/80
224/224 [==============================] - 0s 2ms/step - loss: 0.0924 - val_los
s: 0.0928
Epoch 53/80
224/224 [==============================] - 0s 2ms/step - loss: 0.0924 - val_los
s: 0.0928
Epoch 54/80
224/224 [==============================] - 0s 2ms/step - loss: 0.0924 - val_los
s: 0.0928
Epoch 55/80
224/224 [==============================] - 0s 2ms/step - loss: 0.0924 - val_los
s: 0.0928
Epoch 56/80
224/224 [==============================] - 0s 2ms/step - loss: 0.0924 - val_los
s: 0.0928
Epoch 57/80
224/224 [==============================] - 0s 2ms/step - loss: 0.0924 - val_los
s: 0.0928
Epoch 58/80
224/224 [==============================] - 1s 2ms/step - loss: 0.0924 - val_los
s: 0.0928
Epoch 59/80
224/224 [==============================] - 0s 2ms/step - loss: 0.0924 - val_los
s: 0.0929
Epoch 60/80
224/224 [==============================] - 0s 2ms/step - loss: 0.0923 - val_los
s: 0.0928
```

```
Epoch 61/80
224/224 [==============================] - 0s 2ms/step - loss: 0.0923 - val_los
s: 0.0928
Epoch 62/80
224/224 [==============================] - 0s 2ms/step - loss: 0.0923 - val_los
s: 0.0927
Epoch 63/80
224/224 [==============================] - 0s 2ms/step - loss: 0.0923 - val_los
s: 0.0928
Epoch 64/80
224/224 [==============================] - 0s 2ms/step - loss: 0.0923 - val_los
s: 0.0928
Epoch 65/80
224/224 [==============================] - 0s 2ms/step - loss: 0.0923 - val_los
s: 0.0928
Epoch 66/80
224/224 [==============================] - 0s 2ms/step - loss: 0.0923 - val_los
s: 0.0928
Epoch 67/80
224/224 [==============================] - 0s 2ms/step - loss: 0.0923 - val_los
s: 0.0928
Epoch 68/80
224/224 [==============================] - 0s 2ms/step - loss: 0.0923 - val_los
s: 0.0927
Epoch 69/80
224/224 [==============================] - 0s 2ms/step - loss: 0.0923 - val_los
s: 0.0927
Epoch 70/80
224/224 [==============================] - 0s 2ms/step - loss: 0.0923 - val_los
s: 0.0927
Epoch 71/80
224/224 [==============================] - 0s 2ms/step - loss: 0.0923 - val_los
s: 0.0927
Epoch 72/80
224/224 [==============================] - 0s 2ms/step - loss: 0.0923 - val_los
s: 0.0927
Epoch 73/80
224/224 [==============================] - 0s 2ms/step - loss: 0.0923 - val_los
s: 0.0927
Epoch 74/80
224/224 [==============================] - 0s 2ms/step - loss: 0.0923 - val_los
s: 0.0927
Epoch 75/80
224/224 [==============================] - 0s 2ms/step - loss: 0.0923 - val_los
s: 0.0927
Epoch 76/80
224/224 [==============================] - 0s 2ms/step - loss: 0.0922 - val_los
s: 0.0928
Epoch 77/80
224/224 [==============================] - 0s 2ms/step - loss: 0.0922 - val_los
s: 0.0927
Epoch 78/80
224/224 [==============================] - 0s 2ms/step - loss: 0.0922 - val_los
s: 0.0927
Epoch 79/80
224/224 [==============================] - 0s 2ms/step - loss: 0.0922 - val_los
s: 0.0927
Epoch 80/80
224/224 [==============================] - 0s 2ms/step - loss: 0.0922 - val_los
s: 0.0927
```

```
Epoch 1/80
224/224 [==============================] - 1s 2ms/step - loss: 0.2486 - val_los
s: 0.1659
Epoch 2/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1463 - val_los
s: 0.1316
Epoch 3/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1210 - val_los
s: 0.1128
Epoch 4/80
224/224 [==============================] - 0s 2ms/step - loss: 0.1059 - val_los
s: 0.1005
Epoch 5/80
224/224 [==============================] - 0s 2ms/step - loss: 0.0960 - val_los
s: 0.0927
Epoch 6/80
224/224 [==============================] - 0s 2ms/step - loss: 0.0896 - val_los
s: 0.0874
Epoch 7/80
224/224 [==============================] - 0s 2ms/step - loss: 0.0851 - val_los
s: 0.0835
Epoch 8/80
224/224 [==============================] - 0s 2ms/step - loss: 0.0819 - val_los
s: 0.0809
Epoch 9/80
224/224 [==============================] - 1s 2ms/step - loss: 0.0797 - val_los
s: 0.0791
Epoch 10/80
224/224 [==============================] - 1s 2ms/step - loss: 0.0781 - val_los
s: 0.0778
Epoch 11/80
224/224 [==============================] - 1s 2ms/step - loss: 0.0770 - val_los
s: 0.0769
Epoch 12/80
224/224 [==============================] - 1s 2ms/step - loss: 0.0762 - val_los
s: 0.0761
Epoch 13/80
224/224 [==============================] - 1s 2ms/step - loss: 0.0756 - val_los
s: 0.0756
Epoch 14/80
224/224 [==============================] - 1s 2ms/step - loss: 0.0751 - val_los
s: 0.0752
Epoch 15/80
224/224 [==============================] - 0s 2ms/step - loss: 0.0747 - val_los
s: 0.0750
Epoch 16/80
224/224 [==============================] - 0s 2ms/step - loss: 0.0745 - val_los
s: 0.0746
Epoch 17/80
224/224 [==============================] - 0s 2ms/step - loss: 0.0742 - val_los
s: 0.0745
Epoch 18/80
224/224 [==============================] - 0s 2ms/step - loss: 0.0740 - val_los
s: 0.0743
Epoch 19/80
224/224 [==============================] - 0s 2ms/step - loss: 0.0739 - val_los
s: 0.0741
Epoch 20/80
224/224 [==============================] - 1s 2ms/step - loss: 0.0737 - val_los
s: 0.0740
```

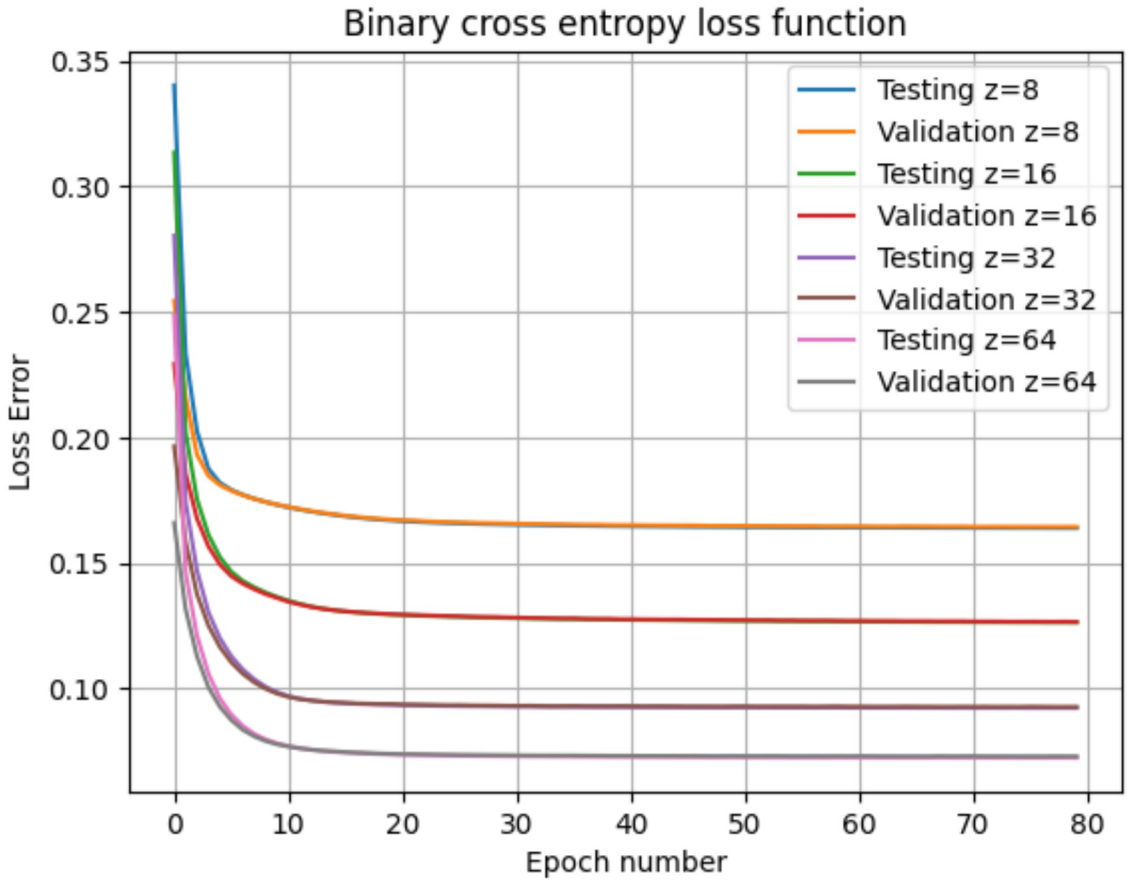
```
Epoch 21/80
224/224 [==============================] - 1s 2ms/step - loss: 0.0736 - val_los
s: 0.0738
Epoch 22/80
224/224 [==============================] - 1s 2ms/step - loss: 0.0735 - val_los
s: 0.0738
Epoch 23/80
224/224 [==============================] - 1s 2ms/step - loss: 0.0734 - val_los
s: 0.0737
Epoch 24/80
224/224 [==============================] - 1s 2ms/step - loss: 0.0734 - val_los
s: 0.0736
Epoch 25/80
224/224 [==============================] - 0s 2ms/step - loss: 0.0733 - val_los
s: 0.0736
Epoch 26/80
224/224 [==============================] - 0s 2ms/step - loss: 0.0732 - val_los
s: 0.0735
Epoch 27/80
224/224 [==============================] - 0s 2ms/step - loss: 0.0732 - val_los
s: 0.0735
Epoch 28/80
224/224 [==============================] - 0s 2ms/step - loss: 0.0731 - val_los
s: 0.0735
Epoch 29/80
224/224 [==============================] - 0s 2ms/step - loss: 0.0731 - val_los
s: 0.0734
Epoch 30/80
224/224 [==============================] - 0s 2ms/step - loss: 0.0730 - val_los
s: 0.0734
Epoch 31/80
224/224 [==============================] - 1s 2ms/step - loss: 0.0730 - val_los
s: 0.0733
Epoch 32/80
224/224 [==============================] - 1s 2ms/step - loss: 0.0730 - val_los
s: 0.0733
Epoch 33/80
224/224 [==============================] - 1s 2ms/step - loss: 0.0729 - val_los
s: 0.0733
Epoch 34/80
224/224 [==============================] - 1s 2ms/step - loss: 0.0729 - val_los
s: 0.0733
Epoch 35/80
224/224 [==============================] - 1s 2ms/step - loss: 0.0729 - val_los
s: 0.0733
Epoch 36/80
224/224 [==============================] - 1s 2ms/step - loss: 0.0729 - val_los
s: 0.0733
Epoch 37/80
224/224 [==============================] - 1s 2ms/step - loss: 0.0728 - val_los
s: 0.0732
Epoch 38/80
224/224 [==============================] - 1s 2ms/step - loss: 0.0728 - val_los
s: 0.0732
Epoch 39/80
224/224 [==============================] - 1s 2ms/step - loss: 0.0728 - val_los
s: 0.0732
Epoch 40/80
224/224 [==============================] - 0s 2ms/step - loss: 0.0728 - val_los
s: 0.0732
```

```
Epoch 41/80
224/224 [==============================] - 1s 2ms/step - loss: 0.0728 - val_los
s: 0.0732
Epoch 42/80
224/224 [==============================] - 1s 2ms/step - loss: 0.0727 - val_los
s: 0.0731
Epoch 43/80
224/224 [==============================] - 1s 2ms/step - loss: 0.0727 - val_los
s: 0.0731
Epoch 44/80
224/224 [==============================] - 1s 2ms/step - loss: 0.0727 - val_los
s: 0.0731
Epoch 45/80
224/224 [==============================] - 1s 2ms/step - loss: 0.0727 - val_los
s: 0.0731
Epoch 46/80
224/224 [==============================] - 1s 2ms/step - loss: 0.0727 - val_los
s: 0.0731
Epoch 47/80
224/224 [==============================] - 0s 2ms/step - loss: 0.0727 - val_los
s: 0.0731
Epoch 48/80
224/224 [==============================] - 1s 2ms/step - loss: 0.0726 - val_los
s: 0.0731
Epoch 49/80
224/224 [==============================] - 1s 2ms/step - loss: 0.0726 - val_los
s: 0.0730
Epoch 50/80
224/224 [==============================] - 1s 2ms/step - loss: 0.0726 - val_los
s: 0.0731
Epoch 51/80
224/224 [==============================] - 1s 2ms/step - loss: 0.0726 - val_los
s: 0.0730
Epoch 52/80
224/224 [==============================] - 1s 2ms/step - loss: 0.0726 - val_los
s: 0.0730
Epoch 53/80
224/224 [==============================] - 1s 2ms/step - loss: 0.0726 - val_los
s: 0.0731
Epoch 54/80
224/224 [==============================] - 1s 2ms/step - loss: 0.0726 - val_los
s: 0.0731
Epoch 55/80
224/224 [==============================] - 1s 2ms/step - loss: 0.0726 - val_los
s: 0.0730
Epoch 56/80
224/224 [==============================] - 1s 2ms/step - loss: 0.0726 - val_los
s: 0.0730
Epoch 57/80
224/224 [==============================] - 1s 2ms/step - loss: 0.0726 - val_los
s: 0.0730
Epoch 58/80
224/224 [==============================] - 1s 2ms/step - loss: 0.0726 - val_los
s: 0.0730
Epoch 59/80
224/224 [==============================] - 1s 2ms/step - loss: 0.0725 - val_los
s: 0.0730
Epoch 60/80
224/224 [==============================] - 1s 2ms/step - loss: 0.0725 - val_los
s: 0.0730
```

```
Epoch 61/80
224/224 [==============================] - 1s 2ms/step - loss: 0.0725 - val_los
s: 0.0730
Epoch 62/80
224/224 [==============================] - 1s 2ms/step - loss: 0.0725 - val_los
s: 0.0730
Epoch 63/80
224/224 [==============================] - 1s 2ms/step - loss: 0.0725 - val_los
s: 0.0730
Epoch 64/80
224/224 [==============================] - 1s 2ms/step - loss: 0.0725 - val_los
s: 0.0730
Epoch 65/80
224/224 [==============================] - 1s 2ms/step - loss: 0.0725 - val_los
s: 0.0729
Epoch 66/80
224/224 [==============================] - 1s 2ms/step - loss: 0.0725 - val_los
s: 0.0729
Epoch 67/80
224/224 [==============================] - 1s 2ms/step - loss: 0.0725 - val_los
s: 0.0729
Epoch 68/80
224/224 [==============================] - 1s 2ms/step - loss: 0.0725 - val_los
s: 0.0730
Epoch 69/80
224/224 [==============================] - 1s 2ms/step - loss: 0.0725 - val_los
s: 0.0730
Epoch 70/80
224/224 [==============================] - 1s 2ms/step - loss: 0.0725 - val_los
s: 0.0730
Epoch 71/80
224/224 [==============================] - 1s 2ms/step - loss: 0.0725 - val_los
s: 0.0730
Epoch 72/80
224/224 [==============================] - 1s 2ms/step - loss: 0.0725 - val_los
s: 0.0730
Epoch 73/80
224/224 [==============================] - 1s 2ms/step - loss: 0.0725 - val_los
s: 0.0730
Epoch 74/80
224/224 [==============================] - 1s 2ms/step - loss: 0.0725 - val_los
s: 0.0730
Epoch 75/80
224/224 [==============================] - 1s 2ms/step - loss: 0.0725 - val_los
s: 0.0729
Epoch 76/80
224/224 [==============================] - 1s 2ms/step - loss: 0.0724 - val_los
s: 0.0729
Epoch 77/80
224/224 [==============================] - 1s 2ms/step - loss: 0.0724 - val_los
s: 0.0729
Epoch 78/80
224/224 [==============================] - 1s 2ms/step - loss: 0.0725 - val_los
s: 0.0730
Epoch 79/80
224/224 [==============================] - 1s 2ms/step - loss: 0.0724 - val_los
s: 0.0729
Epoch 80/80
224/224 [==============================] - 1s 2ms/step - loss: 0.0724 - val_los
s: 0.0729
```

We display the evolution of the loss function (the precision of the model) during the training. z corresponds to the dimension of the latent space.

In [ ]:
```
for z in models:
    plt.plot(models[z]["logs"].history['loss'], label='Testing z='+str(z))
    plt.plot(models[z]["logs"].history['val_loss'], label='Validation z='+str(z)

plt.title('Binary cross entropy loss function')
plt.ylabel('Loss Error')
plt.xlabel('Epoch number')
plt.legend(loc="upper right")

plt.grid()
```



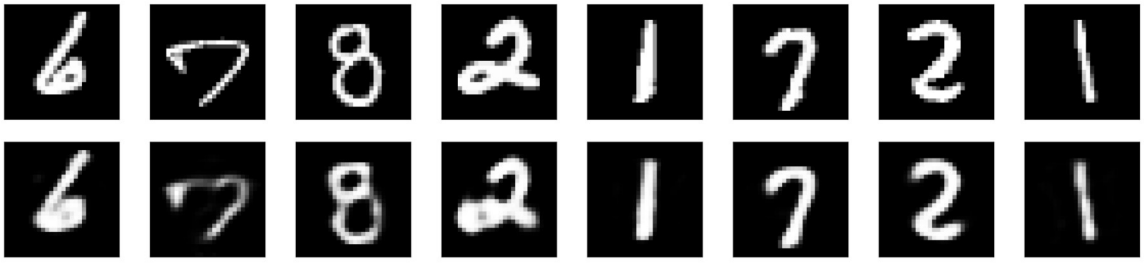
Firstly, we see that our model is robust as the precision on the validation dataset is very similar with the one of the testing dataset, no matter the value z. Thus, we don't have to worry about overfitting. Secondly, as expected, the more the compress version of an image use data, the more the resulted decoded version is good. Otherwise, a video in 480p on youtube could have a better quality than in 1080p :D

From now on, we will use z=32.

Let's test our model on the test dataset and display some input and output images.

In [ ]:
```
decoded_imgs = models[32]["autoencoder"].predict(copy.deepcopy(X_test))



plt.figure(figsize=(18, 4))
for i in range(8):
  # show reference images (input)
  ax = plt.subplot(2, 8, i + 1)
  plt.imshow(copy.deepcopy(X_test[i]).reshape(28, 28))
  ax.get_xaxis().set_visible(False)
  ax.get_yaxis().set_visible(False)

  # show decoded images (output)
  ax = plt.subplot(2, 8, i + 1 + 8)
  plt.imshow(copy.deepcopy(decoded_imgs[i]).reshape(28, 28))
  ax.get_xaxis().set_visible(False)
  ax.get_yaxis().set_visible(False)
plt.show()
```

```
438/438 [==============================] - 0s 554us/step
```

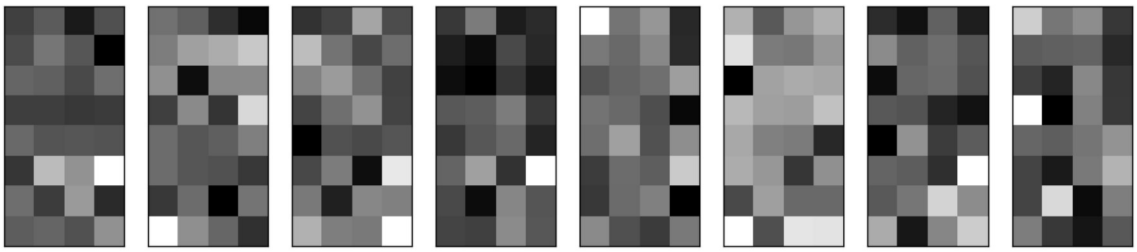


The results are pretty good. The input data have a dimension of 28x28 = 784. The latent space here have a dimensionality of 32. So, this model achieves a compression of 1-32/784 = 96%

For curiosity only, we can display the latent vector associated to those test images. For the same reason the file representing a compress image does not represent anything from a human perspective, the latent space of the autoencoder does not carry visual information.

In [ ]:
```
encoded_imgs = models[32]["encoder"].predict(X_test)

plt.figure(figsize=(16, 8))
for i in range(8):
    ax = plt.subplot(2, 10, i + 1)
    plt.imshow(encoded_imgs[i].reshape(8,4))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)
plt.show()
```

```
438/438 [==============================] - 0s 474us/step
```



We can improve the precision of our model without changing the compression rate by adding some dense layers.

Our new model will have three dense layers of size 128, 64 and 32.

In [ ]:
```
# Input data size
original_dim = 784
intermediate_dim0 = 128
intermediate_dim1 = 64
latent_dims = 32

epochs=80
batch_size=250


input_img = Input(shape=(original_dim,))

encoded = Dense(intermediate_dim0, activation='relu')(input_img)
encoded = Dense(intermediate_dim1, activation='relu')(encoded)
encoded = Dense(latent_dim, activation='relu')(encoded)

decoded = Dense(intermediate_dim1, activation='sigmoid')(encoded)
decoded = Dense(intermediate_dim0, activation='sigmoid')(encoded)
decoded = Dense(original_dim, activation='sigmoid')(encoded)

autoencoder3 = Model(input_img, decoded)
autoencoder3.compile(optimizer='adam', loss='binary_crossentropy') # you can als

encoder3 = Model(input_img, encoded)

encoded_input = Input(shape=(latent_dim,))
decoder3 = Model(encoded_input, decoder_layer(encoded_input))

autoencoder3.summary()
```

```
Model: "model_12"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_9 (InputLayer)        [(None, 784)]             0

 dense_8 (Dense)             (None, 128)               100480

 dense_9 (Dense)             (None, 64)                8256

 dense_10 (Dense)            (None, 64)                4160

 dense_13 (Dense)            (None, 784)               50960

=================================================================
Total params: 163,856
Trainable params: 163,856
Non-trainable params: 0
_________________________________________________________________
```

In [ ]:
```
logs3 = autoencoder3.fit(X_train, X_train,epochs=epochs,batch_size=batch_size,sh
```

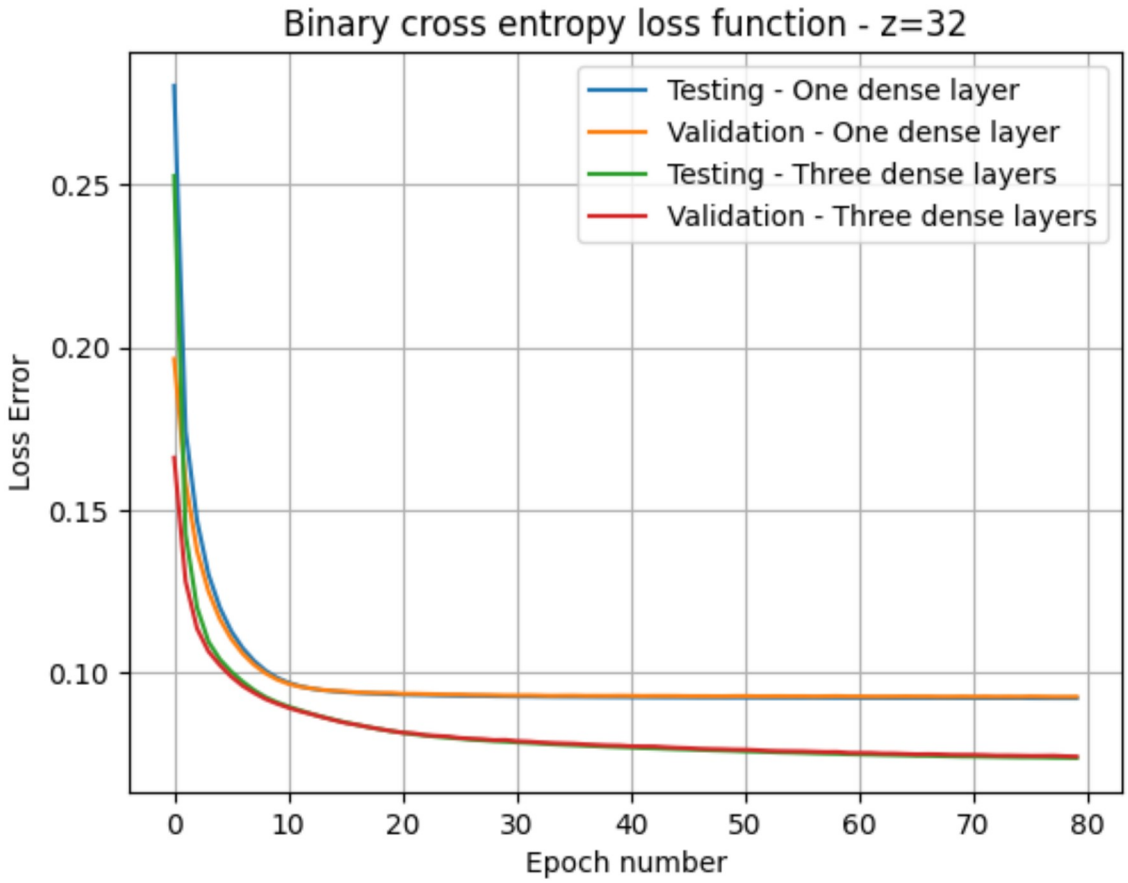
```
Epoch 1/80
224/224 [==============================] - 1s 3ms/step - loss: 0.2527 - val_los
s: 0.1661
Epoch 2/80
224/224 [==============================] - 1s 3ms/step - loss: 0.1426 - val_los
s: 0.1281
Epoch 3/80
224/224 [==============================] - 1s 3ms/step - loss: 0.1201 - val_los
s: 0.1137
Epoch 4/80
224/224 [==============================] - 1s 3ms/step - loss: 0.1096 - val_los
s: 0.1066
Epoch 5/80
224/224 [==============================] - 1s 3ms/step - loss: 0.1042 - val_los
s: 0.1024
Epoch 6/80
224/224 [==============================] - 1s 3ms/step - loss: 0.1004 - val_los
s: 0.0988
Epoch 7/80
224/224 [==============================] - 1s 3ms/step - loss: 0.0973 - val_los
s: 0.0960
Epoch 8/80
224/224 [==============================] - 1s 3ms/step - loss: 0.0947 - val_los
s: 0.0939
Epoch 9/80
224/224 [==============================] - 1s 3ms/step - loss: 0.0925 - val_los
s: 0.0919
Epoch 10/80
224/224 [==============================] - 1s 3ms/step - loss: 0.0910 - val_los
s: 0.0906
Epoch 11/80
224/224 [==============================] - 1s 3ms/step - loss: 0.0896 - val_los
s: 0.0893
Epoch 12/80
224/224 [==============================] - 1s 3ms/step - loss: 0.0885 - val_los
s: 0.0882
Epoch 13/80
224/224 [==============================] - 1s 3ms/step - loss: 0.0874 - val_los
s: 0.0873
Epoch 14/80
224/224 [==============================] - 1s 3ms/step - loss: 0.0865 - val_los
s: 0.0864
Epoch 15/80
224/224 [==============================] - 1s 3ms/step - loss: 0.0856 - val_los
s: 0.0855
Epoch 16/80
224/224 [==============================] - 1s 3ms/step - loss: 0.0848 - val_los
s: 0.0846
Epoch 17/80
224/224 [==============================] - 1s 3ms/step - loss: 0.0840 - val_los
s: 0.0840
Epoch 18/80
224/224 [==============================] - 1s 3ms/step - loss: 0.0833 - val_los
s: 0.0833
Epoch 19/80
224/224 [==============================] - 1s 3ms/step - loss: 0.0827 - val_los
s: 0.0827
Epoch 20/80
224/224 [==============================] - 1s 3ms/step - loss: 0.0821 - val_los
s: 0.0821
```

```
Epoch 21/80
224/224 [==============================] - 1s 3ms/step - loss: 0.0816 - val_los
s: 0.0816
Epoch 22/80
224/224 [==============================] - 1s 3ms/step - loss: 0.0811 - val_los
s: 0.0814
Epoch 23/80
224/224 [==============================] - 1s 3ms/step - loss: 0.0808 - val_los
s: 0.0809
Epoch 24/80
224/224 [==============================] - 1s 3ms/step - loss: 0.0804 - val_los
s: 0.0806
Epoch 25/80
224/224 [==============================] - 1s 3ms/step - loss: 0.0801 - val_los
s: 0.0804
Epoch 26/80
224/224 [==============================] - 1s 3ms/step - loss: 0.0798 - val_los
s: 0.0800
Epoch 27/80
224/224 [==============================] - 1s 3ms/step - loss: 0.0795 - val_los
s: 0.0798
Epoch 28/80
224/224 [==============================] - 1s 3ms/step - loss: 0.0793 - val_los
s: 0.0796
Epoch 29/80
224/224 [==============================] - 1s 3ms/step - loss: 0.0790 - val_los
s: 0.0793
Epoch 30/80
224/224 [==============================] - 1s 3ms/step - loss: 0.0789 - val_los
s: 0.0794
Epoch 31/80
224/224 [==============================] - 1s 3ms/step - loss: 0.0787 - val_los
s: 0.0790
Epoch 32/80
224/224 [==============================] - 1s 3ms/step - loss: 0.0785 - val_los
s: 0.0789
Epoch 33/80
224/224 [==============================] - 1s 3ms/step - loss: 0.0783 - val_los
s: 0.0786
Epoch 34/80
224/224 [==============================] - 1s 3ms/step - loss: 0.0781 - val_los
s: 0.0785
Epoch 35/80
224/224 [==============================] - 1s 3ms/step - loss: 0.0779 - val_los
s: 0.0783
Epoch 36/80
224/224 [==============================] - 1s 3ms/step - loss: 0.0778 - val_los
s: 0.0782
Epoch 37/80
224/224 [==============================] - 1s 3ms/step - loss: 0.0776 - val_los
s: 0.0780
Epoch 38/80
224/224 [==============================] - 1s 3ms/step - loss: 0.0775 - val_los
s: 0.0779
Epoch 39/80
224/224 [==============================] - 1s 3ms/step - loss: 0.0773 - val_los
s: 0.0777
Epoch 40/80
224/224 [==============================] - 1s 3ms/step - loss: 0.0772 - val_los
s: 0.0777
```

```
Epoch 41/80
224/224 [==============================] - 1s 3ms/step - loss: 0.0771 - val_los
s: 0.0774
Epoch 42/80
224/224 [==============================] - 1s 3ms/step - loss: 0.0769 - val_los
s: 0.0774
Epoch 43/80
224/224 [==============================] - 1s 3ms/step - loss: 0.0768 - val_los
s: 0.0774
Epoch 44/80
224/224 [==============================] - 1s 3ms/step - loss: 0.0767 - val_los
s: 0.0772
Epoch 45/80
224/224 [==============================] - 1s 3ms/step - loss: 0.0766 - val_los
s: 0.0770
Epoch 46/80
224/224 [==============================] - 1s 3ms/step - loss: 0.0765 - val_los
s: 0.0769
Epoch 47/80
224/224 [==============================] - 1s 3ms/step - loss: 0.0763 - val_los
s: 0.0767
Epoch 48/80
224/224 [==============================] - 1s 3ms/step - loss: 0.0762 - val_los
s: 0.0766
Epoch 49/80
224/224 [==============================] - 1s 3ms/step - loss: 0.0761 - val_los
s: 0.0765
Epoch 50/80
224/224 [==============================] - 1s 3ms/step - loss: 0.0760 - val_los
s: 0.0764
Epoch 51/80
224/224 [==============================] - 1s 3ms/step - loss: 0.0759 - val_los
s: 0.0764
Epoch 52/80
224/224 [==============================] - 1s 3ms/step - loss: 0.0758 - val_los
s: 0.0763
Epoch 53/80
224/224 [==============================] - 1s 3ms/step - loss: 0.0757 - val_los
s: 0.0761
Epoch 54/80
224/224 [==============================] - 1s 3ms/step - loss: 0.0756 - val_los
s: 0.0760
Epoch 55/80
224/224 [==============================] - 1s 3ms/step - loss: 0.0755 - val_los
s: 0.0759
Epoch 56/80
224/224 [==============================] - 1s 3ms/step - loss: 0.0754 - val_los
s: 0.0759
Epoch 57/80
224/224 [==============================] - 1s 3ms/step - loss: 0.0753 - val_los
s: 0.0758
Epoch 58/80
224/224 [==============================] - 1s 3ms/step - loss: 0.0752 - val_los
s: 0.0757
Epoch 59/80
224/224 [==============================] - 1s 3ms/step - loss: 0.0751 - val_los
s: 0.0757
Epoch 60/80
224/224 [==============================] - 1s 3ms/step - loss: 0.0750 - val_los
s: 0.0754
```

```
Epoch 61/80
224/224 [==============================] - 1s 3ms/step - loss: 0.0749 - val_los
s: 0.0754
Epoch 62/80
224/224 [==============================] - 1s 3ms/step - loss: 0.0749 - val_los
s: 0.0754
Epoch 63/80
224/224 [==============================] - 1s 3ms/step - loss: 0.0748 - val_los
s: 0.0752
Epoch 64/80
224/224 [==============================] - 1s 3ms/step - loss: 0.0747 - val_los
s: 0.0752
Epoch 65/80
224/224 [==============================] - 1s 3ms/step - loss: 0.0746 - val_los
s: 0.0751
Epoch 66/80
224/224 [==============================] - 1s 3ms/step - loss: 0.0745 - val_los
s: 0.0750
Epoch 67/80
224/224 [==============================] - 1s 3ms/step - loss: 0.0745 - val_los
s: 0.0750
Epoch 68/80
224/224 [==============================] - 1s 3ms/step - loss: 0.0744 - val_los
s: 0.0750
Epoch 69/80
224/224 [==============================] - 1s 3ms/step - loss: 0.0744 - val_los
s: 0.0748
Epoch 70/80
224/224 [==============================] - 1s 3ms/step - loss: 0.0743 - val_los
s: 0.0748
Epoch 71/80
224/224 [==============================] - 1s 3ms/step - loss: 0.0743 - val_los
s: 0.0748
Epoch 72/80
224/224 [==============================] - 1s 3ms/step - loss: 0.0742 - val_los
s: 0.0747
Epoch 73/80
224/224 [==============================] - 1s 3ms/step - loss: 0.0742 - val_los
s: 0.0746
Epoch 74/80
224/224 [==============================] - 1s 3ms/step - loss: 0.0741 - val_los
s: 0.0746
Epoch 75/80
224/224 [==============================] - 1s 3ms/step - loss: 0.0741 - val_los
s: 0.0746
Epoch 76/80
224/224 [==============================] - 1s 3ms/step - loss: 0.0740 - val_los
s: 0.0745
Epoch 77/80
224/224 [==============================] - 1s 3ms/step - loss: 0.0740 - val_los
s: 0.0745
Epoch 78/80
224/224 [==============================] - 1s 3ms/step - loss: 0.0739 - val_los
s: 0.0745
Epoch 79/80
224/224 [==============================] - 1s 3ms/step - loss: 0.0739 - val_los
s: 0.0744
Epoch 80/80
224/224 [==============================] - 1s 3ms/step - loss: 0.0738 - val_los
s: 0.0743
```

In [ ]:
```
plt.plot(models[32]["logs"].history['loss'], label='Testing - One dense layer')
plt.plot(models[32]["logs"].history['val_loss'], label='Validation - One dense l

plt.plot(logs3.history['loss'], label='Testing - Three dense layers')
plt.plot(logs3.history['val_loss'], label='Validation - Three dense layers')

plt.title('Binary cross entropy loss function - z=32')
plt.ylabel('Loss Error')
plt.xlabel('Epoch number')
plt.legend(loc="upper right")

plt.grid()
```



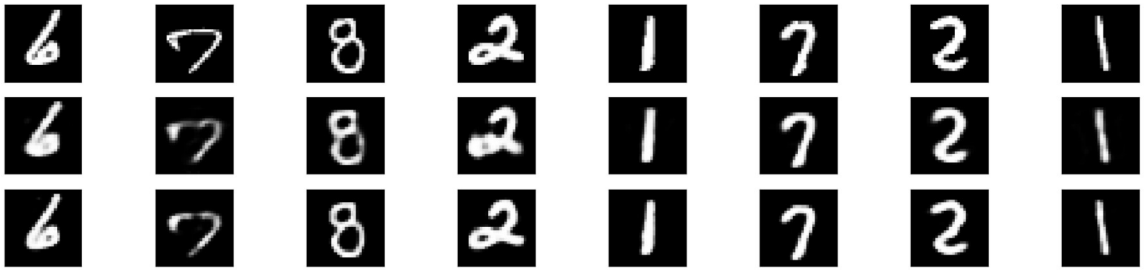
As we can see, the addition of two layers allows our model to reach a better precision without changing the dimensionality of the latent space.

We also see that the new model take more time to converge. Indeed, with one layer, the total number of parameters that can be train (weights) is 50992. In comparison, with three layers, this number is equal to 163856.

We now display a comparison between the one layer model outputs, and the three layer ones, for the same input images.

In [ ]:
```
decoded_imgs = models[32]["autoencoder"].predict(copy.deepcopy(X_test))
decoded_imgs3 = autoencoder3.predict(copy.deepcopy(X_test))

plt.figure(figsize=(18, 4))
for i in range(8):
  # show reference images (input)
  ax = plt.subplot(3, 8, i + 1)
  plt.imshow(copy.deepcopy(X_test[i]).reshape(28, 28))
  ax.get_xaxis().set_visible(False)
  ax.get_yaxis().set_visible(False)

  # show decoded images (output) - One Layer
  ax = plt.subplot(3, 8, i + 1 + 8)
  plt.imshow(copy.deepcopy(decoded_imgs[i]).reshape(28, 28))
  ax.get_xaxis().set_visible(False)
  ax.get_yaxis().set_visible(False)

  # show decoded images (output) - Three Layers
  ax = plt.subplot(3, 8, i + 1 + 2*8)
  plt.imshow(copy.deepcopy(decoded_imgs3[i]).reshape(28, 28))
  ax.get_xaxis().set_visible(False)
  ax.get_yaxis().set_visible(False)

plt.show()
```

```
438/438 [==============================] - 0s 524us/step
438/438 [==============================] - 0s 611us/step
```



Our result are very good. However, for such task, it is common to use convolutional layers (CNN). We decided to build a third model based on this idea.

Auto-encoder with two convolutional layers

Let's build a auto-encoder from a CNN. We use the following characteristics :

- Two convolutional layers composed of 32 features with a kernel of size (3,3) followed by :
  - MaxPooling2D (2,2) for encoder (divide by two the dimensionality)
  - UpSampling2D (2,2) for the decoder (double the dimensionality)
- One last layer of 1 feature with a (3,3) kernel and a sigmoid activation function

The target of this project is to focus on the auto-encoder. We don't show the different tests we made to tune the layers of this CNN model as it is not linked to the understanding of auto-encoder or latent space. Indeed, we just want to show that in such tasks, including the following denoising process, the use of convolutional layer is very relevant.

In [ ]:

```
latent_dims = 32

epochs=80
batch_size=250

width = 28

X_trainIm = X_train.reshape(X_train.shape[0], width, width, 1)
X_testIm  = X_test.reshape(X_test.shape[0], width, width, 1)

input_img = Input(shape=(width,width,1))

encoded = Conv2D(latent_dims,(3,3),activation='relu',padding='same')(input_img)
encoded = MaxPooling2D(pool_size=(2,2),padding='same')(encoded)
encoded = Conv2D(latent_dims,(3,3),activation='relu',padding='same')(encoded)
encoded = MaxPooling2D(pool_size=(2,2),padding='same')(encoded)

decoded = Conv2D(latent_dims,(3,3),activation='relu',padding='same')(encoded)
decoded = UpSampling2D((2,2))(decoded)
decoded = Conv2D(latent_dims,(3,3),activation='relu',padding='same')(decoded)
decoded = UpSampling2D((2,2))(decoded)
decoded = Conv2D(1,(3,3),activation='sigmoid',padding='same')(decoded)

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

encoderConv = Model(input_img, encoded)

encoded_input = Input(shape=(latent_dim,))
decoderConv = Model(encoded_input, decoder_layer(encoded_input))

autoencoderConv.summary()
```

```
Model: "model_30"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_21 (InputLayer)       [(None, 28, 28, 1)]       0

 conv2d_25 (Conv2D)          (None, 28, 28, 32)        320

 max_pooling2d_10 (MaxPoolin  (None, 14, 14, 32)       0
 g2D)

 conv2d_26 (Conv2D)          (None, 14, 14, 32)        9248

 max_pooling2d_11 (MaxPoolin  (None, 7, 7, 32)         0
 g2D)

 conv2d_27 (Conv2D)          (None, 7, 7, 32)          9248

 up_sampling2d_10 (UpSamplin  (None, 14, 14, 32)       0
 g2D)

 conv2d_28 (Conv2D)          (None, 14, 14, 32)        9248

 up_sampling2d_11 (UpSamplin  (None, 28, 28, 32)       0
 g2D)

 conv2d_29 (Conv2D)          (None, 28, 28, 1)         289

=================================================================
Total params: 28,353
Trainable params: 28,353
Non-trainable params: 0
_________________________________________________________________
```

In [ ]:
```
logsConv = autoencoderConv.fit(X_trainIm, X_trainIm, epochs=epochs, batch_size=b
                validation_data=(X_testIm, X_testIm))
```

```
Epoch 1/80
224/224 [==============================] - 32s 139ms/step - loss: 0.1462 - val_l
oss: 0.0828
Epoch 2/80
224/224 [==============================] - 31s 137ms/step - loss: 0.0789 - val_l
oss: 0.0760
Epoch 3/80
224/224 [==============================] - 31s 139ms/step - loss: 0.0744 - val_l
oss: 0.0731
Epoch 4/80
224/224 [==============================] - 31s 138ms/step - loss: 0.0724 - val_l
oss: 0.0716
Epoch 5/80
224/224 [==============================] - 31s 138ms/step - loss: 0.0711 - val_l
oss: 0.0707
Epoch 6/80
224/224 [==============================] - 31s 136ms/step - loss: 0.0703 - val_l
oss: 0.0702
Epoch 7/80
224/224 [==============================] - 30s 134ms/step - loss: 0.0696 - val_l
oss: 0.0694
Epoch 8/80
224/224 [==============================] - 31s 141ms/step - loss: 0.0691 - val_l
oss: 0.0689
Epoch 9/80
224/224 [==============================] - 31s 140ms/step - loss: 0.0686 - val_l
oss: 0.0686
Epoch 10/80
224/224 [==============================] - 31s 138ms/step - loss: 0.0683 - val_l
oss: 0.0681
Epoch 11/80
224/224 [==============================] - 31s 137ms/step - loss: 0.0679 - val_l
oss: 0.0678
Epoch 12/80
224/224 [==============================] - 31s 137ms/step - loss: 0.0676 - val_l
oss: 0.0675
Epoch 13/80
224/224 [==============================] - 31s 137ms/step - loss: 0.0674 - val_l
oss: 0.0673
Epoch 14/80
224/224 [==============================] - 30s 135ms/step - loss: 0.0671 - val_l
oss: 0.0672
Epoch 15/80
224/224 [==============================] - 31s 138ms/step - loss: 0.0669 - val_l
oss: 0.0670
Epoch 16/80
224/224 [==============================] - 31s 138ms/step - loss: 0.0667 - val_l
oss: 0.0666
Epoch 17/80
224/224 [==============================] - 31s 137ms/step - loss: 0.0665 - val_l
oss: 0.0665
Epoch 18/80
224/224 [==============================] - 31s 139ms/step - loss: 0.0663 - val_l
oss: 0.0663
Epoch 19/80
224/224 [==============================] - 31s 138ms/step - loss: 0.0662 - val_l
oss: 0.0661
Epoch 20/80
224/224 [==============================] - 31s 137ms/step - loss: 0.0660 - val_l
oss: 0.0661
```

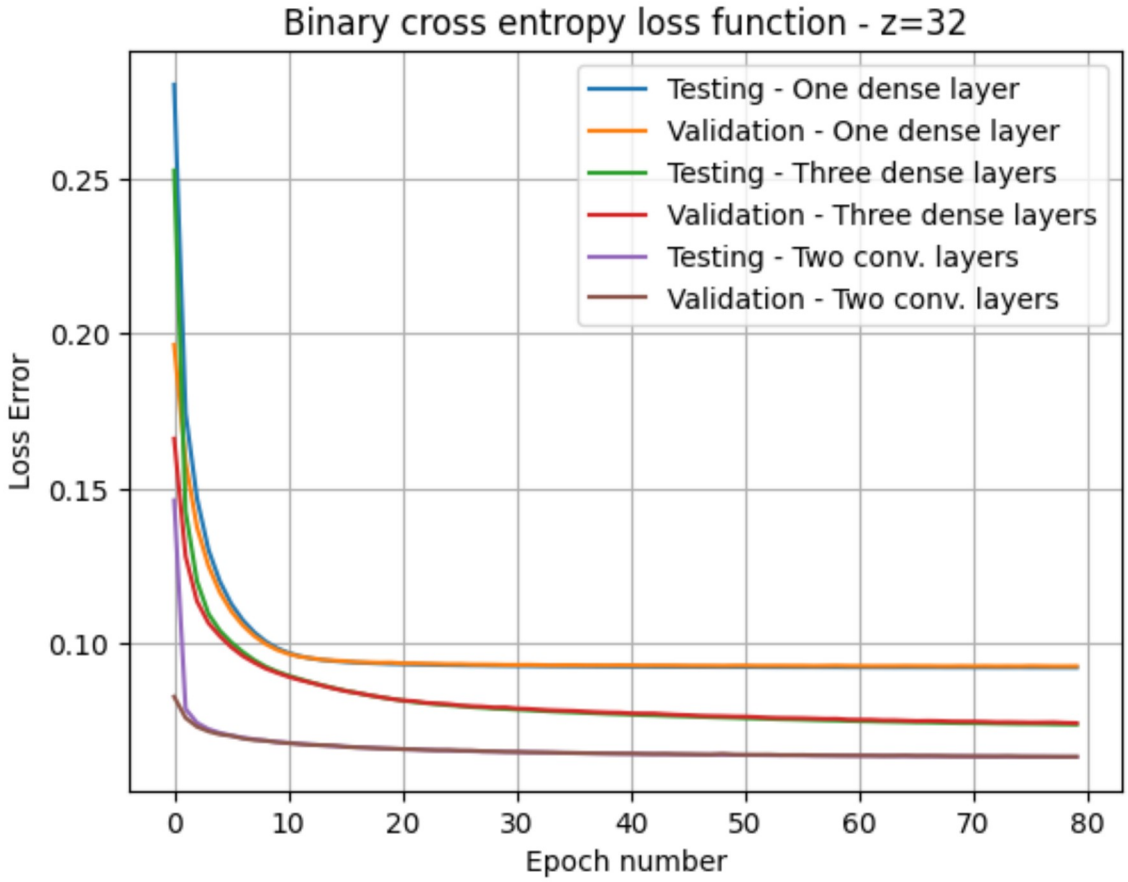
```
Epoch 21/80
224/224 [==============================] - 31s 137ms/step - loss: 0.0659 - val_l
oss: 0.0659
Epoch 22/80
224/224 [==============================] - 31s 138ms/step - loss: 0.0658 - val_l
oss: 0.0658
Epoch 23/80
224/224 [==============================] - 31s 141ms/step - loss: 0.0657 - val_l
oss: 0.0657
Epoch 24/80
224/224 [==============================] - 31s 140ms/step - loss: 0.0655 - val_l
oss: 0.0656
Epoch 25/80
224/224 [==============================] - 32s 143ms/step - loss: 0.0655 - val_l
oss: 0.0656
Epoch 26/80
224/224 [==============================] - 32s 143ms/step - loss: 0.0653 - val_l
oss: 0.0654
Epoch 27/80
224/224 [==============================] - 32s 141ms/step - loss: 0.0653 - val_l
oss: 0.0654
Epoch 28/80
224/224 [==============================] - 32s 141ms/step - loss: 0.0652 - val_l
oss: 0.0652
Epoch 29/80
224/224 [==============================] - 32s 141ms/step - loss: 0.0651 - val_l
oss: 0.0652
Epoch 30/80
224/224 [==============================] - 32s 141ms/step - loss: 0.0650 - val_l
oss: 0.0650
Epoch 31/80
224/224 [==============================] - 32s 141ms/step - loss: 0.0649 - val_l
oss: 0.0650
Epoch 32/80
224/224 [==============================] - 32s 142ms/step - loss: 0.0649 - val_l
oss: 0.0649
Epoch 33/80
224/224 [==============================] - 32s 142ms/step - loss: 0.0648 - val_l
oss: 0.0649
Epoch 34/80
224/224 [==============================] - 32s 141ms/step - loss: 0.0648 - val_l
oss: 0.0648
Epoch 35/80
224/224 [==============================] - 32s 142ms/step - loss: 0.0647 - val_l
oss: 0.0647
Epoch 36/80
224/224 [==============================] - 32s 141ms/step - loss: 0.0646 - val_l
oss: 0.0647
Epoch 37/80
224/224 [==============================] - 32s 141ms/step - loss: 0.0646 - val_l
oss: 0.0646
Epoch 38/80
224/224 [==============================] - 32s 143ms/step - loss: 0.0645 - val_l
oss: 0.0646
Epoch 39/80
224/224 [==============================] - 31s 137ms/step - loss: 0.0645 - val_l
oss: 0.0645
Epoch 40/80
224/224 [==============================] - 31s 137ms/step - loss: 0.0644 - val_l
oss: 0.0645
```

```
Epoch 41/80
224/224 [==============================] - 31s 137ms/step - loss: 0.0644 - val_l
oss: 0.0645
Epoch 42/80
224/224 [==============================] - 31s 137ms/step - loss: 0.0643 - val_l
oss: 0.0644
Epoch 43/80
224/224 [==============================] - 31s 137ms/step - loss: 0.0643 - val_l
oss: 0.0643
Epoch 44/80
224/224 [==============================] - 31s 137ms/step - loss: 0.0642 - val_l
oss: 0.0644
Epoch 45/80
224/224 [==============================] - 31s 140ms/step - loss: 0.0642 - val_l
oss: 0.0643
Epoch 46/80
224/224 [==============================] - 30s 135ms/step - loss: 0.0642 - val_l
oss: 0.0643
Epoch 47/80
224/224 [==============================] - 30s 135ms/step - loss: 0.0641 - val_l
oss: 0.0642
Epoch 48/80
224/224 [==============================] - 30s 135ms/step - loss: 0.0641 - val_l
oss: 0.0642
Epoch 49/80
224/224 [==============================] - 30s 135ms/step - loss: 0.0641 - val_l
oss: 0.0644
Epoch 50/80
224/224 [==============================] - 30s 135ms/step - loss: 0.0640 - val_l
oss: 0.0642
Epoch 51/80
224/224 [==============================] - 30s 134ms/step - loss: 0.0640 - val_l
oss: 0.0641
Epoch 52/80
224/224 [==============================] - 31s 138ms/step - loss: 0.0640 - val_l
oss: 0.0641
Epoch 53/80
224/224 [==============================] - 30s 135ms/step - loss: 0.0640 - val_l
oss: 0.0641
Epoch 54/80
224/224 [==============================] - 30s 134ms/step - loss: 0.0639 - val_l
oss: 0.0640
Epoch 55/80
224/224 [==============================] - 30s 136ms/step - loss: 0.0639 - val_l
oss: 0.0640
Epoch 56/80
224/224 [==============================] - 30s 135ms/step - loss: 0.0639 - val_l
oss: 0.0640
Epoch 57/80
224/224 [==============================] - 30s 134ms/step - loss: 0.0638 - val_l
oss: 0.0639
Epoch 58/80
224/224 [==============================] - 30s 134ms/step - loss: 0.0638 - val_l
oss: 0.0639
Epoch 59/80
224/224 [==============================] - 30s 134ms/step - loss: 0.0638 - val_l
oss: 0.0639
Epoch 60/80
224/224 [==============================] - 30s 134ms/step - loss: 0.0638 - val_l
oss: 0.0638
```

```
Epoch 61/80
224/224 [==============================] - 30s 134ms/step - loss: 0.0638 - val_l
oss: 0.0638
Epoch 62/80
224/224 [==============================] - 30s 134ms/step - loss: 0.0637 - val_l
oss: 0.0638
Epoch 63/80
224/224 [==============================] - 30s 134ms/step - loss: 0.0637 - val_l
oss: 0.0638
Epoch 64/80
224/224 [==============================] - 30s 134ms/step - loss: 0.0637 - val_l
oss: 0.0638
Epoch 65/80
224/224 [==============================] - 30s 135ms/step - loss: 0.0636 - val_l
oss: 0.0639
Epoch 66/80
224/224 [==============================] - 30s 135ms/step - loss: 0.0636 - val_l
oss: 0.0637
Epoch 67/80
224/224 [==============================] - 30s 134ms/step - loss: 0.0636 - val_l
oss: 0.0638
Epoch 68/80
224/224 [==============================] - 30s 134ms/step - loss: 0.0636 - val_l
oss: 0.0637
Epoch 69/80
224/224 [==============================] - 30s 135ms/step - loss: 0.0636 - val_l
oss: 0.0637
Epoch 70/80
224/224 [==============================] - 30s 134ms/step - loss: 0.0636 - val_l
oss: 0.0636
Epoch 71/80
224/224 [==============================] - 30s 134ms/step - loss: 0.0635 - val_l
oss: 0.0636
Epoch 72/80
224/224 [==============================] - 30s 134ms/step - loss: 0.0635 - val_l
oss: 0.0636
Epoch 73/80
224/224 [==============================] - 30s 134ms/step - loss: 0.0635 - val_l
oss: 0.0636
Epoch 74/80
224/224 [==============================] - 30s 134ms/step - loss: 0.0635 - val_l
oss: 0.0636
Epoch 75/80
224/224 [==============================] - 30s 134ms/step - loss: 0.0635 - val_l
oss: 0.0635
Epoch 76/80
224/224 [==============================] - 30s 134ms/step - loss: 0.0634 - val_l
oss: 0.0635
Epoch 77/80
224/224 [==============================] - 30s 134ms/step - loss: 0.0634 - val_l
oss: 0.0635
Epoch 78/80
224/224 [==============================] - 30s 135ms/step - loss: 0.0634 - val_l
oss: 0.0635
Epoch 79/80
224/224 [==============================] - 30s 135ms/step - loss: 0.0634 - val_l
oss: 0.0635
Epoch 80/80
224/224 [==============================] - 30s 134ms/step - loss: 0.0634 - val_l
oss: 0.0635
```

Let's compare the convergence speed and the precision of our new CNN model.

In [ ]:
```
plt.plot(models[32]["logs"].history['loss'], label='Testing - One dense layer')
plt.plot(models[32]["logs"].history['val_loss'], label='Validation - One dense l

plt.plot(logs3.history['loss'], label='Testing - Three dense layers')
plt.plot(logs3.history['val_loss'], label='Validation - Three dense layers')

plt.plot(logsConv.history['loss'], label='Testing - Two conv. layers')
plt.plot(logsConv.history['val_loss'], label='Validation - Two conv. layers')

plt.title('Binary cross entropy loss function - z=32')
plt.ylabel('Loss Error')
plt.xlabel('Epoch number')
plt.legend(loc="upper right")

plt.grid()
```



As expected, the use of convolutional layers allow to reach better result. Also, the number of weight is low in comparison with the other two models (28353) which alows a better converging speed.

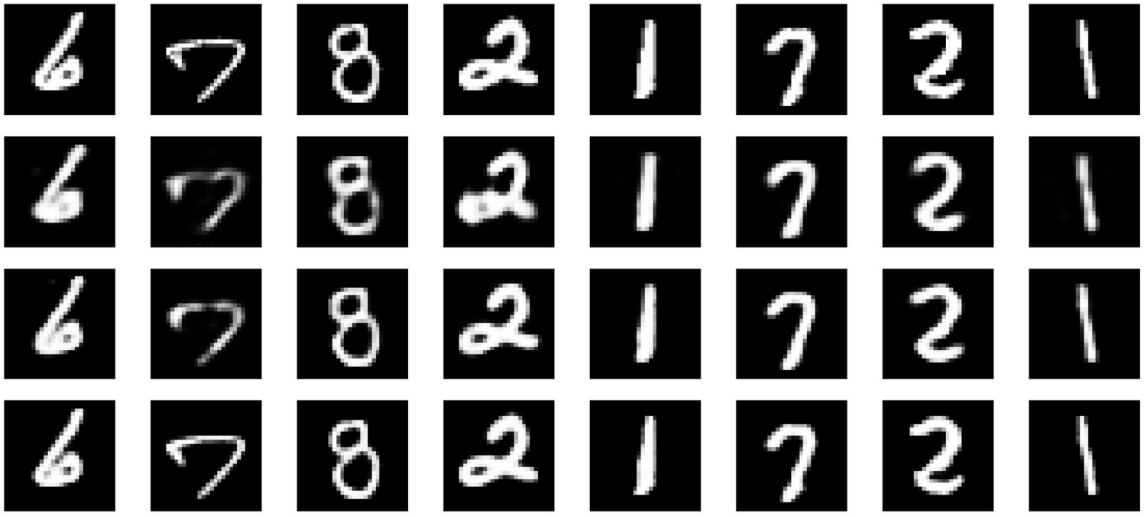
By the way, the remark about robustness that we made about the one-dense-layer model is still true for the other ones. As the testing and validation process reach the same precision, we know that there is no overfitting.

Here follows the comparison of the output images of the three models :

- Input image
- Output image - One dense layer
- Output image - Three dense layers
- Output image - Two convolutional layers

In [ ]:
```
decoded_imgs = models[32]["autoencoder"].predict(copy.deepcopy(X_test))
decoded_imgs3 = autoencoder3.predict(copy.deepcopy(X_test))
decoded_imgsConv = autoencoderConv.predict(copy.deepcopy(X_testIm))

plt.figure(figsize=(18, 8))
for i in range(8):
  # show reference images (input)
  ax = plt.subplot(4, 8, i + 1)
  plt.imshow(copy.deepcopy(X_test[i]).reshape(28, 28))
  ax.get_xaxis().set_visible(False)
  ax.get_yaxis().set_visible(False)

  # show decoded images (output) - One layer
  ax = plt.subplot(4, 8, i + 1 + 8)
  plt.imshow(copy.deepcopy(decoded_imgs[i]).reshape(28, 28))
  ax.get_xaxis().set_visible(False)
  ax.get_yaxis().set_visible(False)

  # show decoded images (output) - Three layers
  ax = plt.subplot(4, 8, i + 1 + 2*8)
  plt.imshow(copy.deepcopy(decoded_imgs3[i]).reshape(28, 28))
  ax.get_xaxis().set_visible(False)
  ax.get_yaxis().set_visible(False)

  # show decoded images (output) - Two conv layers
  ax = plt.subplot(4, 8, i + 1 + 3*8)
  plt.imshow(copy.deepcopy(decoded_imgsConv[i]).reshape(28, 28))
  ax.get_xaxis().set_visible(False)
  ax.get_yaxis().set_visible(False)

plt.show()
```

```
438/438 [==============================] - 0s 549us/step
438/438 [==============================] - 0s 639us/step
438/438 [==============================] - 2s 4ms/step
```



6- Using auto-encoder for image denoising

When we compress an image, we try to keep only the relevant information. If some noise is adding in the input image, it should be 'ignored' by the compression process, that is, the noise should not be considered in the latent vector. Thus, the output image should not contain noise.

We can try this hypothesis. Notice that this time, the model does not try to fit the output image with the input image. Indeed, the input image are noisy. The model tries to fit the clean version of the input image.

In [ ]:
```
# We add noise to our dataset
ratioNoise = 0.5
X_trainIm_noisy = X_trainIm + ratioNoise * np.random.normal(loc=0.0, scale=1.0,
X_testIm_noisy = X_testIm + ratioNoise * np.random.normal(loc=0.0, scale=1.0, si

X_trainIm_noisy = np.clip(X_trainIm_noisy, 0, 1)
X_testIm_noisy = np.clip(X_testIm_noisy, 0, 1)

# We train our model, using the same previous CNN auto-encoder
logsConvNoise = autoencoderConv.fit(X_trainIm_noisy, X_trainIm, epochs=epochs, b
                validation_data=(X_testIm_noisy, X_testIm))
```

```
Epoch 1/80
224/224 [==============================] - 32s 142ms/step - loss: 0.1553 - val_l
oss: 0.1236
Epoch 2/80
224/224 [==============================] - 31s 140ms/step - loss: 0.1205 - val_l
oss: 0.1178
Epoch 3/80
224/224 [==============================] - 33s 145ms/step - loss: 0.1166 - val_l
oss: 0.1152
Epoch 4/80
224/224 [==============================] - 33s 150ms/step - loss: 0.1143 - val_l
oss: 0.1136
Epoch 5/80
224/224 [==============================] - 34s 150ms/step - loss: 0.1128 - val_l
oss: 0.1123
Epoch 6/80
224/224 [==============================] - 33s 149ms/step - loss: 0.1115 - val_l
oss: 0.1109
Epoch 7/80
224/224 [==============================] - 33s 148ms/step - loss: 0.1104 - val_l
oss: 0.1098
Epoch 8/80
224/224 [==============================] - 33s 149ms/step - loss: 0.1094 - val_l
oss: 0.1088
Epoch 9/80
224/224 [==============================] - 33s 147ms/step - loss: 0.1085 - val_l
oss: 0.1081
Epoch 10/80
224/224 [==============================] - 33s 148ms/step - loss: 0.1078 - val_l
oss: 0.1073
Epoch 11/80
224/224 [==============================] - 33s 149ms/step - loss: 0.1072 - val_l
oss: 0.1067
Epoch 12/80
224/224 [==============================] - 33s 148ms/step - loss: 0.1066 - val_l
oss: 0.1062
Epoch 13/80
224/224 [==============================] - 33s 148ms/step - loss: 0.1061 - val_l
oss: 0.1057
Epoch 14/80
224/224 [==============================] - 33s 149ms/step - loss: 0.1056 - val_l
oss: 0.1058
Epoch 15/80
224/224 [==============================] - 33s 147ms/step - loss: 0.1051 - val_l
oss: 0.1048
Epoch 16/80
224/224 [==============================] - 33s 147ms/step - loss: 0.1047 - val_l
oss: 0.1044
Epoch 17/80
224/224 [==============================] - 33s 149ms/step - loss: 0.1044 - val_l
oss: 0.1044
Epoch 18/80
224/224 [==============================] - 33s 149ms/step - loss: 0.1040 - val_l
oss: 0.1038
Epoch 19/80
224/224 [==============================] - 33s 149ms/step - loss: 0.1036 - val_l
oss: 0.1035
Epoch 20/80
224/224 [==============================] - 33s 148ms/step - loss: 0.1034 - val_l
oss: 0.1035
```

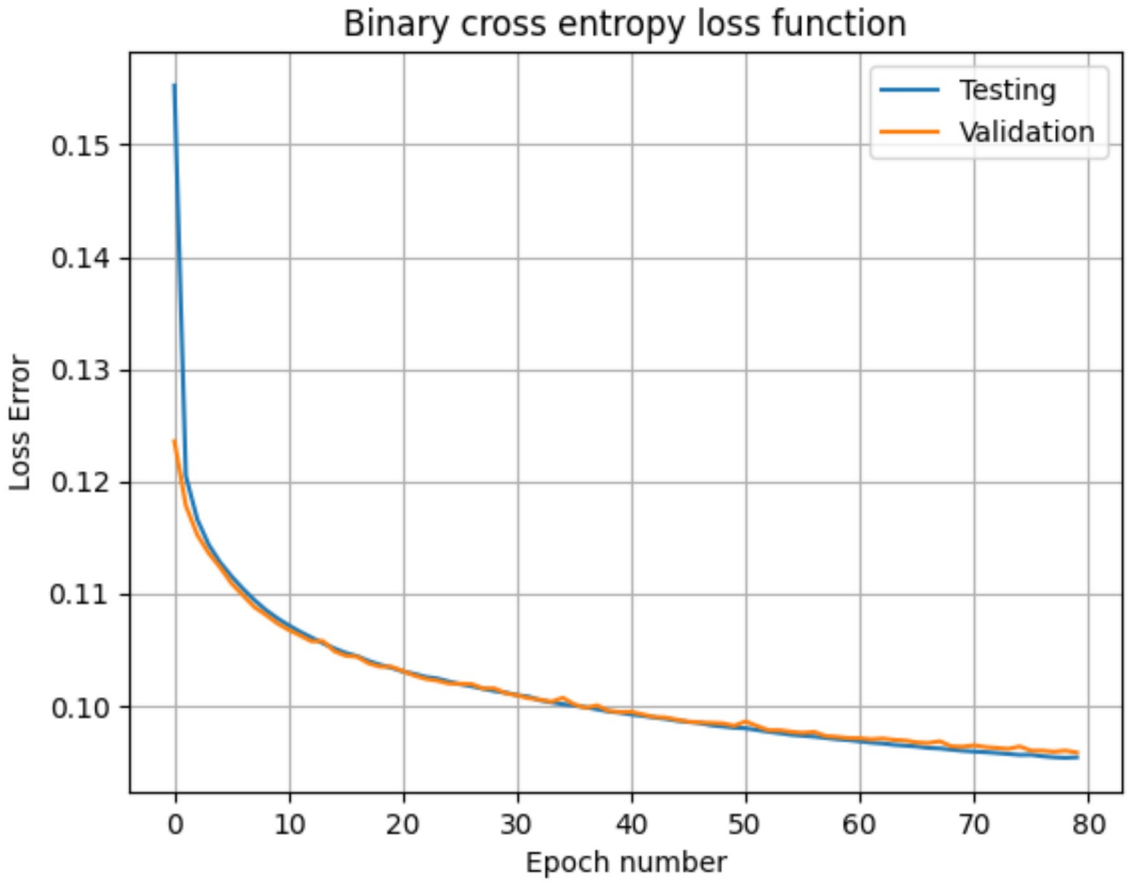
```
Epoch 21/80
224/224 [==============================] - 33s 148ms/step - loss: 0.1030 - val_l
oss: 0.1031
Epoch 22/80
224/224 [==============================] - 34s 150ms/step - loss: 0.1028 - val_l
oss: 0.1027
Epoch 23/80
224/224 [==============================] - 33s 147ms/step - loss: 0.1026 - val_l
oss: 0.1024
Epoch 24/80
224/224 [==============================] - 33s 148ms/step - loss: 0.1024 - val_l
oss: 0.1022
Epoch 25/80
224/224 [==============================] - 33s 149ms/step - loss: 0.1021 - val_l
oss: 0.1020
Epoch 26/80
224/224 [==============================] - 33s 148ms/step - loss: 0.1019 - val_l
oss: 0.1019
Epoch 27/80
224/224 [==============================] - 33s 147ms/step - loss: 0.1017 - val_l
oss: 0.1019
Epoch 28/80
224/224 [==============================] - 33s 147ms/step - loss: 0.1015 - val_l
oss: 0.1015
Epoch 29/80
224/224 [==============================] - 33s 148ms/step - loss: 0.1013 - val_l
oss: 0.1015
Epoch 30/80
224/224 [==============================] - 33s 148ms/step - loss: 0.1011 - val_l
oss: 0.1011
Epoch 31/80
224/224 [==============================] - 33s 149ms/step - loss: 0.1009 - val_l
oss: 0.1009
Epoch 32/80
224/224 [==============================] - 34s 152ms/step - loss: 0.1008 - val_l
oss: 0.1007
Epoch 33/80
224/224 [==============================] - 34s 150ms/step - loss: 0.1005 - val_l
oss: 0.1005
Epoch 34/80
224/224 [==============================] - 34s 151ms/step - loss: 0.1003 - val_l
oss: 0.1003
Epoch 35/80
224/224 [==============================] - 34s 152ms/step - loss: 0.1001 - val_l
oss: 0.1007
Epoch 36/80
224/224 [==============================] - 34s 150ms/step - loss: 0.1000 - val_l
oss: 0.1001
Epoch 37/80
224/224 [==============================] - 34s 151ms/step - loss: 0.0998 - val_l
oss: 0.0998
Epoch 38/80
224/224 [==============================] - 34s 150ms/step - loss: 0.0997 - val_l
oss: 0.1000
Epoch 39/80
224/224 [==============================] - 34s 151ms/step - loss: 0.0995 - val_l
oss: 0.0995
Epoch 40/80
224/224 [==============================] - 34s 151ms/step - loss: 0.0994 - val_l
oss: 0.0994
```

```
Epoch 41/80
224/224 [==============================] - 34s 151ms/step - loss: 0.0992 - val_l
oss: 0.0994
Epoch 42/80
224/224 [==============================] - 34s 151ms/step - loss: 0.0991 - val_l
oss: 0.0992
Epoch 43/80
224/224 [==============================] - 34s 150ms/step - loss: 0.0989 - val_l
oss: 0.0990
Epoch 44/80
224/224 [==============================] - 33s 149ms/step - loss: 0.0988 - val_l
oss: 0.0989
Epoch 45/80
224/224 [==============================] - 33s 149ms/step - loss: 0.0986 - val_l
oss: 0.0987
Epoch 46/80
224/224 [==============================] - 33s 149ms/step - loss: 0.0985 - val_l
oss: 0.0986
Epoch 47/80
224/224 [==============================] - 34s 150ms/step - loss: 0.0984 - val_l
oss: 0.0985
Epoch 48/80
224/224 [==============================] - 33s 149ms/step - loss: 0.0982 - val_l
oss: 0.0985
Epoch 49/80
224/224 [==============================] - 33s 149ms/step - loss: 0.0981 - val_l
oss: 0.0984
Epoch 50/80
224/224 [==============================] - 34s 153ms/step - loss: 0.0980 - val_l
oss: 0.0982
Epoch 51/80
224/224 [==============================] - 35s 156ms/step - loss: 0.0980 - val_l
oss: 0.0986
Epoch 52/80
224/224 [==============================] - 34s 152ms/step - loss: 0.0978 - val_l
oss: 0.0982
Epoch 53/80
224/224 [==============================] - 34s 152ms/step - loss: 0.0977 - val_l
oss: 0.0978
Epoch 54/80
224/224 [==============================] - 35s 155ms/step - loss: 0.0975 - val_l
oss: 0.0978
Epoch 55/80
224/224 [==============================] - 33s 149ms/step - loss: 0.0974 - val_l
oss: 0.0977
Epoch 56/80
224/224 [==============================] - 33s 148ms/step - loss: 0.0973 - val_l
oss: 0.0976
Epoch 57/80
224/224 [==============================] - 33s 149ms/step - loss: 0.0972 - val_l
oss: 0.0976
Epoch 58/80
224/224 [==============================] - 34s 152ms/step - loss: 0.0971 - val_l
oss: 0.0973
Epoch 59/80
224/224 [==============================] - 33s 146ms/step - loss: 0.0970 - val_l
oss: 0.0972
Epoch 60/80
224/224 [==============================] - 32s 144ms/step - loss: 0.0969 - val_l
oss: 0.0971
```

```
Epoch 61/80
224/224 [==============================] - 33s 149ms/step - loss: 0.0968 - val_l
oss: 0.0971
Epoch 62/80
224/224 [==============================] - 33s 146ms/step - loss: 0.0967 - val_l
oss: 0.0970
Epoch 63/80
224/224 [==============================] - 33s 146ms/step - loss: 0.0966 - val_l
oss: 0.0971
Epoch 64/80
224/224 [==============================] - 33s 146ms/step - loss: 0.0965 - val_l
oss: 0.0969
Epoch 65/80
224/224 [==============================] - 33s 146ms/step - loss: 0.0964 - val_l
oss: 0.0969
Epoch 66/80
224/224 [==============================] - 33s 148ms/step - loss: 0.0963 - val_l
oss: 0.0967
Epoch 67/80
224/224 [==============================] - 33s 146ms/step - loss: 0.0962 - val_l
oss: 0.0967
Epoch 68/80
224/224 [==============================] - 33s 146ms/step - loss: 0.0962 - val_l
oss: 0.0968
Epoch 69/80
224/224 [==============================] - 33s 146ms/step - loss: 0.0961 - val_l
oss: 0.0964
Epoch 70/80
224/224 [==============================] - 33s 146ms/step - loss: 0.0960 - val_l
oss: 0.0963
Epoch 71/80
224/224 [==============================] - 33s 146ms/step - loss: 0.0959 - val_l
oss: 0.0965
Epoch 72/80
224/224 [==============================] - 33s 146ms/step - loss: 0.0958 - val_l
oss: 0.0963
Epoch 73/80
224/224 [==============================] - 33s 147ms/step - loss: 0.0958 - val_l
oss: 0.0962
Epoch 74/80
224/224 [==============================] - 33s 146ms/step - loss: 0.0957 - val_l
oss: 0.0961
Epoch 75/80
224/224 [==============================] - 33s 146ms/step - loss: 0.0956 - val_l
oss: 0.0964
Epoch 76/80
224/224 [==============================] - 33s 145ms/step - loss: 0.0956 - val_l
oss: 0.0960
Epoch 77/80
224/224 [==============================] - 33s 145ms/step - loss: 0.0955 - val_l
oss: 0.0960
Epoch 78/80
224/224 [==============================] - 33s 148ms/step - loss: 0.0954 - val_l
oss: 0.0959
Epoch 79/80
224/224 [==============================] - 34s 151ms/step - loss: 0.0953 - val_l
oss: 0.0960
Epoch 80/80
224/224 [==============================] - 33s 147ms/step - loss: 0.0954 - val_l
oss: 0.0958
```

Let's plot the evolution of the precision during the training.

In [ ]:
```
plt.plot(logsConvNoise.history['loss'], label='Testing')
plt.plot(logsConvNoise.history['val_loss'], label='Validation')
plt.title('Binary cross entropy loss function')
plt.ylabel('Loss Error')
plt.xlabel('Epoch number')
plt.legend(loc="upper right")

plt.grid()
plt.show()
```



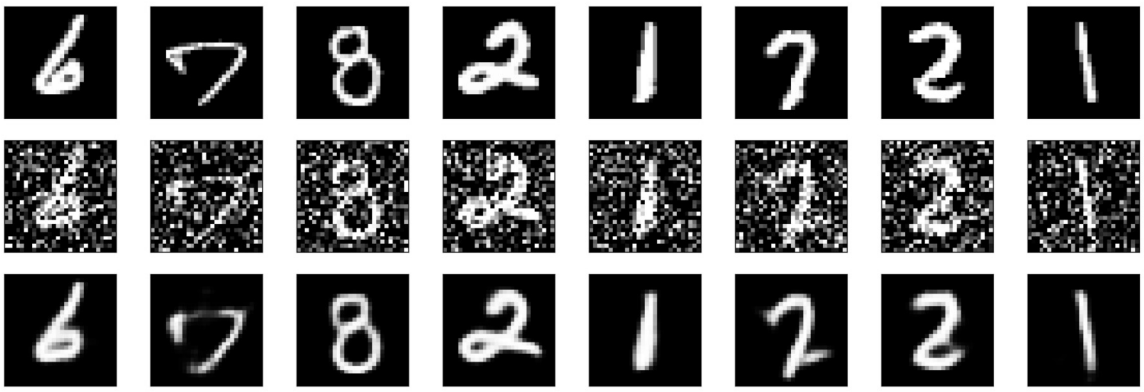
We see that the converging time is longer than for the previous task. This is due to the higher complexity of the task.

Also, the obtained precision is very good. Even with such noisy image, our model can reach a performance similar to the ones of the previous part.

We can now display the original image, its noisy version and the output image, after the denoising process.

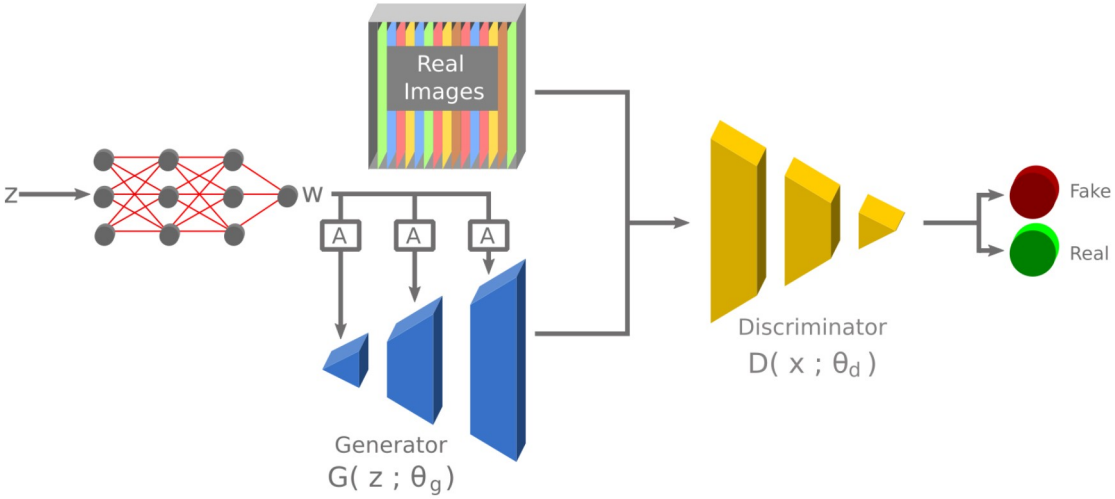
In [ ]:
```
decoded_imgsConvNoise = autoencoderConv.predict(copy.deepcopy(X_testIm_noisy))

plt.figure(figsize=(18, 6))
for i in range(8):
  # show reference images (input)
  ax = plt.subplot(3, 8, i + 1)
  plt.imshow(copy.deepcopy(X_test[i]).reshape(28, 28))
  ax.get_xaxis().set_visible(False)
  ax.get_yaxis().set_visible(False)

  # show decoded images (output) - One Layer
  ax = plt.subplot(3, 8, i + 1 + 8)
  plt.imshow(copy.deepcopy(X_testIm_noisy[i]).reshape(28, 28))
  ax.get_xaxis().set_visible(False)
  ax.get_yaxis().set_visible(False)

  # show decoded images (output) - Three Layers
  ax = plt.subplot(3, 8, i + 1 + 2*8)
  plt.imshow(copy.deepcopy(decoded_imgsConvNoise[i]).reshape(28, 28))
  ax.get_xaxis().set_visible(False)
  ax.get_yaxis().set_visible(False)

plt.show()
```

```
438/438 [==============================] - 2s 4ms/step
```



The results are not perfect but still impressive. The ability of an autoencoder to compress the input images is thus very powerful as a denoising process.

Latent space and style transfer



In StyleGAN2, each generator layer is fed a specific latent variable through linear projection (as shown in the image above). The first layers are responsible for the global appearance of the output image, while the last ones focus on finer details.

Let's import two images, one used as a content, and one used its style.

In [ ]:
```
# Function that download and preprocess an image.
def getImage(img_url,img_size):
    img = tf.io.decode_image(
        tf.io.read_file(tf.keras.utils.get_file(os.path.basename(img_url)[-128:]
        channels=3, dtype=tf.float32)[tf.newaxis, ...]

    # Obtaining square image through cropping
    new_shape = min(img.shape[1], img.shape[2])
    offset_y = max(img.shape[1] - img.shape[2], 0) // 2
    offset_x = max(img.shape[2] - img.shape[1], 0) // 2
    img = tf.image.crop_to_bounding_box(img, offset_y, offset_x, new_shape, new_
    img = tf.image.resize(img, (img_size, img_size), preserve_aspect_ratio=True)
    return img
```

In [ ]:
```
output_img_size = 400 # Can be modified
style_img_size = 256  # Corresponds to the size of the image used to train the m

content_img_url = 'https://upload.wikimedia.org/wikipedia/commons/d/d2/Emmanuel_
content_img = getImage(content_img_url,output_img_size)

style_img_url = 'https://upload.wikimedia.org/wikipedia/commons/thumb/e/ea/Van_G
style_img = getImage(style_img_url,style_img_size)

# Display the images
plt.figure()

plt.subplot(1,2,1)
plt.imshow(content_img[0])
plt.title("Content image")
plt.axis('off')

plt.subplot(1,2,2)
plt.imshow(style_img[0])
plt.title("Style image")
plt.axis('off')

plt.show()
```

We use a TF hub module to import a pretrained model and feed it with our images. The model will compute the latent vector of the two images.

Then, it will create a new latent vector whose the first half corresponds to the first half of the style image latent vector. The second half correspond to the second half of the content image latent vector.

Finally, the decoder part of the model will be feed with this new latent vector.

Basically, this latent vector have the finner detail of Macron but the style of the Starry Night.

In [ ]:
```
tf_hub = hub.load('https://tfhub.dev/google/magenta/arbitrary-image-stylization-

output = tf_hub(tf.constant(content_img), tf.constant(style_img))
stylized_img = output[0]

plt.figure()

plt.imshow(stylized_img[0])
plt.title("Content image")
plt.axis('off')

plt.show()
```

The resulting image show well the difference between the first and the late values of the latent vector associated to a picture.

With the used model, the global appearance of a picture is encoded in the first values of the latent vector of an image. As we are using the first values of Starry Night's latent vector, the resulting image has the same "style". Here, we really see that the texture of the style image is used on the content image.

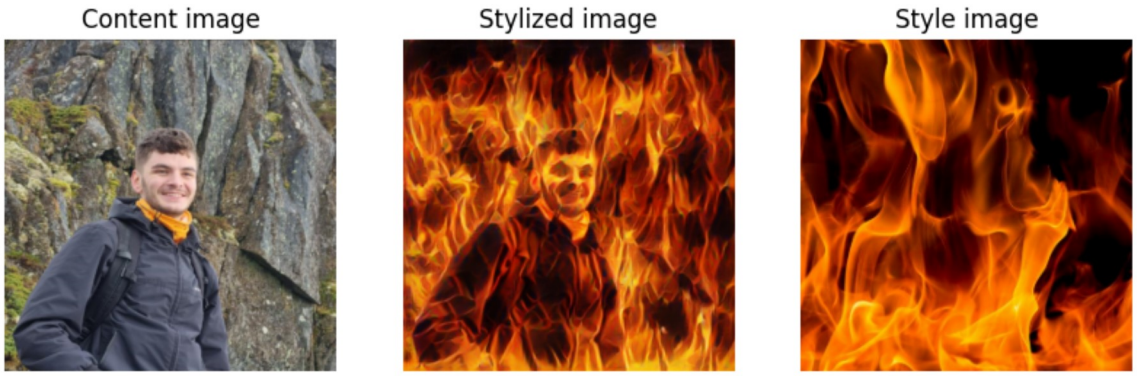
In the opposite, the last values of the latent vector code for finer details. Here the face and the shape of Macron is well present. However, the blurred background has disapeared.

In the following section, we show other examples.

We hope this project showed you the power of the auto-encoders and gave you an intuition about what are the latent vectors.

Happy new year!

In [ ]:
```
content_img = getImage('https://i.ibb.co/hsV9yQs/LOUIS-FR-NEAU.jpg',output_img_s
style_img = getImage('https://www.paintedpaperart.com/wp-content/uploads/2022/05
stylized_img = tf_hub(tf.constant(content_img), tf.constant(style_img))[0]
plt.figure(figsize=(10,4))
plt.subplot(1,3,1),plt.imshow(content_img[0]),plt.title("Content image"),plt.axi
plt.subplot(1,3,2),plt.imshow(stylized_img[0]),plt.title("Stylized image"),plt.a
plt.subplot(1,3,3),plt.imshow(style_img[0]);plt.title("Style image");plt.axis('o

content_img = getImage('https://i.ibb.co/GRHz6dG/h-h-Capture-d-cran-2022-12-30-0
style_img = getImage('https://rescue18.fr/wp-content/uploads/2020/04/flammes.jpg
stylized_img = tf_hub(tf.constant(content_img), tf.constant(style_img))[0]
plt.figure(figsize=(10,4))
plt.subplot(1,3,1),plt.imshow(content_img[0]),plt.title("Content image"),plt.axi
plt.subplot(1,3,2),plt.imshow(stylized_img[0]),plt.title("Stylized image"),plt.a
plt.subplot(1,3,3),plt.imshow(style_img[0]);plt.title("Style image");plt.axis('o

content_img = getImage('https://i.ibb.co/kGt9pzS/oui1664277434909.jpg',output_im
style_img = getImage('https://da32ev14kd4yl.cloudfront.net/versioned/milone-art-
stylized_img = tf_hub(tf.constant(content_img), tf.constant(style_img))[0]
plt.figure(figsize=(10,4))
plt.subplot(1,3,1),plt.imshow(content_img[0]),plt.title("Content image"),plt.axi
plt.subplot(1,3,2),plt.imshow(stylized_img[0]),plt.title("Stylized image"),plt.a
plt.subplot(1,3,3),plt.imshow(style_img[0]);plt.title("Style image");plt.axis('o

content_img = getImage('https://i1.rgstatic.net/ii/profile.image/272281872105501
style_img = getImage('https://upload.wikimedia.org/wikipedia/commons/thumb/e/ea/
stylized_img = tf_hub(tf.constant(content_img), tf.constant(style_img))[0]
plt.figure(figsize=(10,4))
plt.subplot(1,3,1),plt.imshow(content_img[0]),plt.title("Content image"),plt.axi
plt.subplot(1,3,2),plt.imshow(stylized_img[0]),plt.title("Stylized image"),plt.a
plt.subplot(1,3,3),plt.imshow(style_img[0]);plt.title("Style image");plt.axis('o
```




```
Downloading data from https://da32ev14kd4yl.cloudfront.net/versioned/milone-art-
academy/images/marine-coloriste-acrylique-tableau.JPG
9717983/9717983 [==============================] - 4s 0us/step
```

Content image

Stylized image

Style image

Content image

Stylized image

Style image