Nelson Lefebvre
NUID: 03137910

Assignment 1: Convolutional Autoencoder

Code :

```
import matplotlib.pyplot as plt
import numpy as np
from keras.datasets import cifar10
from keras.models import Model
from keras.layers import Input, Conv2D, MaxPooling2D, UpSampling2D
from tensorflow.keras.optimizers import Adam

(x_train, _), (x_test, _) = cifar10.load_data()

print("train data shape :", x_train.shape)
print("test data shape :", x_test.shape)

#define the input shape
input_img = Input(shape=(32,32,3))
# Normalize the data
x_train = x_train / 255
x_test = x_test / 255


x = Conv2D(32, (3, 3), activation='relu', padding='same') (input_img)
x = MaxPooling2D((2, 2)) (x)

x = Conv2D(16, (3, 3), activation='relu', padding='same') (x)
x = MaxPooling2D((2, 2)) (x)

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

x = Conv2D(8, (3, 3), activation='relu', padding='same') (x)
x = UpSampling2D((2, 2)) (x)

x = Conv2D(16, (3, 3), activation='relu', padding='same') (x)
x = UpSampling2D((2, 2)) (x)

x = Conv2D(32, (3, 3), activation='relu', padding='same') (x)
```

```
decoded = Conv2D(3, (3, 3), padding='same') (x)

model = (Model(input_img,decoded))
model.compile(optimizer = 'adam', loss ='mae', metrics=['accuracy'] )
model.summary()

print(x_train.shape)
print(x_test.shape)


history = model.fit(x_train,x_train,
            epochs = 100,
            batch_size = 50,
            validation_data = (x_test, x_test))

predict = model.predict(x_test)

def display(img1, img2, count = 6):
      n = count
      plt.figure()

      for i in range(n):
      ax = plt.subplot(2, n, i+1)
      plt.imshow(img1[300*i])
      ax = plt.subplot(2, n, i+1+n)
      plt.imshow(img2[300*i])
      plt.show()

display(x_test, predict)
```

Output :

train data (50000, 32, 32, 3)
test data (10000, 32, 32, 3)
Model: "model"

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[(None, 32, 32, 3)]	0
conv2d (Conv2D)	(None, 32, 32, 32)	896
max_pooling2d (MaxPooling2D)	(None, 16, 16, 32)	0
conv2d_1 (Conv2D)	(None, 16, 16, 16)	4624

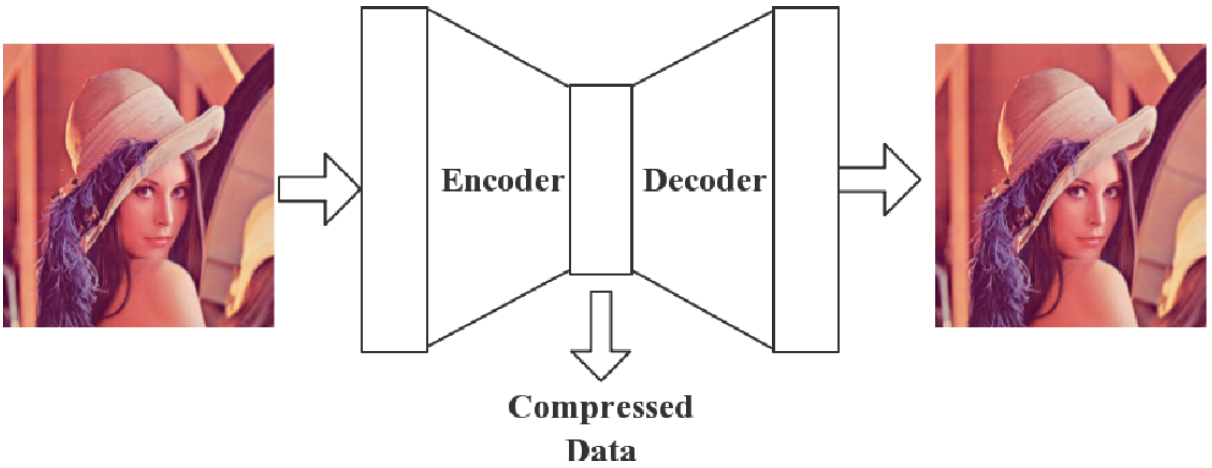
```
max_pooling2d_1 (MaxPooling  (None, 8, 8, 16)         0
2D)

conv2d_2 (Conv2D)           (None, 8, 8, 8)           1160

conv2d_3 (Conv2D)           (None, 8, 8, 8)           584

up_sampling2d (UpSampling2D  (None, 16, 16, 8)        0
)

conv2d_4 (Conv2D)           (None, 16, 16, 16)        1168

up_sampling2d_1 (UpSampling  (None, 32, 32, 16)       0
2D)

conv2d_5 (Conv2D)           (None, 32, 32, 32)        4640

conv2d_6 (Conv2D)           (None, 32, 32, 3)         867

=================================================================
Total params: 13,939
Trainable params: 13,939
Non-trainable params: 0
_________________________________________________________________
(50000, 32, 32, 3)
(10000, 32, 32, 3)
Epoch 1/100
1000/1000 [==============================] - 85s 84ms/step - loss: 0.0805 -
accuracy: 0.6277 - val_loss: 0.0607 - val_accuracy: 0.7274
Epoch 2/100
1000/1000 [==============================] - 82s 82ms/step - loss: 0.0584 -
accuracy: 0.7360 - val_loss: 0.0552 - val_accuracy: 0.7603
Epoch 3/100
1000/1000 [==============================] - 84s 84ms/step - loss: 0.0545 -
accuracy: 0.7533 - val_loss: 0.0568 - val_accuracy: 0.7596
Epoch 4/100
1000/1000 [==============================] - 87s 87ms/step - loss: 0.0519 -
accuracy: 0.7627 - val_loss: 0.0504 - val_accuracy: 0.7675
Epoch 5/100
1000/1000 [==============================] - 81s 81ms/step - loss: 0.0500 -
accuracy: 0.7723 - val_loss: 0.0485 - val_accuracy: 0.7721
…

…
1000/1000 [==============================] - 81s 81ms/step - loss: 0.0370 -
accuracy: 0.8182 - val_loss: 0.0364 - val_accuracy: 0.8220
Epoch 96/100
```

1000/1000 [==============================] - 81s 81ms/step - loss: 0.0367 - accuracy: 0.8210 - val_loss: 0.0373 - val_accuracy: 0.8243
Epoch 97/100
1000/1000 [==============================] - 81s 81ms/step - loss: 0.0368 - accuracy: 0.8199 - val_loss: 0.0362 - val_accuracy: 0.8231
Epoch 98/100
1000/1000 [==============================] - 81s 81ms/step - loss: 0.0367 - accuracy: 0.8197 - val_loss: 0.0402 - val_accuracy: 0.7476
Epoch 99/100
1000/1000 [==============================] - 81s 81ms/step - loss: 0.0367 - accuracy: 0.8197 - val_loss: 0.0369 - val_accuracy: 0.8177
Epoch 100/100
1000/1000 [==============================] - 81s 81ms/step - loss: 0.0368 - accuracy: 0.8193 - val_loss: 0.0365 - val_accuracy: 0.8287
313/313 [==============================] - 4s 12ms/step

Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

Discussion of thinking and discovery:

1) I used the same schema as in the course for the construction of the convolutional autoencoder. Who is based on encoding the image in a compressed one with a neural network and then decoding it to generate the “sentence's” main pattern.



But I had to add a layer in addition to encoding (Conv + Pooling) and a layer in addition to decoding (Conv + UpSempling).
Then I chose to use the fit() function to fit (train/evaluate) to see the loss and the accuracy of the model. So in this example with 3 layers and 100 fitting I get a model accuracy of 82% and a loss of 0.036%.

2) The thing where I had the most “difficulty” was when displaying the images. But after some research I found quite quickly how to print them. For example, I need to add in the parameters of plt.subplot (Xaxis, Yaxis, shareX). ShareX is for sharing the image in the axis x, to make them “in multiple x lines”.

3) Bonus: I tried like in class, only 2 layers and only one test fitting and here is the result :

```
import matplotlib.pyplot as plt
import numpy as np
from keras.datasets import cifar10
from keras.models import Model
from keras.layers import Input, Conv2D, MaxPooling2D, UpSampling2D
from tensorflow.keras.optimizers import Adam

(x_train, _), (x_test, _) = cifar10.load_data()

print("train data shape :", x_train.shape)
print("test data shape :", x_test.shape)

#define the input shape
input_img = Input(shape=(32,32,3))
# Normalize the data
x_train = x_train / 255
x_test = x_test / 255


x = Conv2D(32, (3, 3), activation='relu', padding='same') (input_img)
x = MaxPooling2D((2, 2)) (x)

x = Conv2D(16, (3, 3), activation='relu', padding='same') (x)
encoded = MaxPooling2D((2, 2)) (x)

x = Conv2D(16, (3, 3), activation='relu', padding='same') (x)
x = UpSampling2D((2, 2)) (x)

x = Conv2D(32, (3, 3), activation='relu', padding='same') (x)

decoded = Conv2D(3, (3, 3), padding='same') (x)

model = (Model(input_img,decoded))
model.compile(optimizer = 'adam', loss ='mae', metrics=['accuracy'] )
model.summary()

print(x_train.shape)
print(x_test.shape)


history = model.fit(x_train,x_train,
           epochs = 1,
           batch_size = 50,
           validation_data = (x_test, x_test))

predict = model.predict(x_test)

def display(img1, img2, count = 6):
    n = count
    plt.figure()

    for i in range(n):
        ax = plt.subplot(2, n, i+1)
        plt.imshow(img1[300*i])
        ax = plt.subplot(2, n, i+1+n)
        plt.imshow(img2[300*i])
    plt.show()

display(x_test, predict)
```

```
train data shape : (50000, 32, 32, 3)
test data shape : (10000, 32, 32, 3)
Model: "model_10"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_11 (InputLayer)       [(None, 32, 32, 3)]       0

 conv2d_70 (Conv2D)          (None, 32, 32, 32)        896

 max_pooling2d_30 (MaxPoolin  (None, 16, 16, 32)       0
 g2D)

 conv2d_71 (Conv2D)          (None, 16, 16, 16)        4624

 conv2d_72 (Conv2D)          (None, 16, 16, 16)        2320

 up_sampling2d_20 (UpSamplin  (None, 32, 32, 16)       0
 g2D)

 conv2d_73 (Conv2D)          (None, 32, 32, 32)        4640

 conv2d_74 (Conv2D)          (None, 32, 32, 3)         867

=================================================================
Total params: 13,347
Trainable params: 13,347
Non-trainable params: 0
_________________________________________________________________
(50000, 32, 32, 3)
(10000, 32, 32, 3)
1000/1000 [==============================] - 83s 82ms/step - loss: 0.0436 - accuracy: 0.7511 - val_loss: 0.0278 - val_accu
racy: 0.7979
313/313 [==============================] - 4s 12ms/step
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
```

We can conclude that the more we add layers, the more precise is the accuracy model and the loss is less important. Nevertheless, one important thing is the time execution. For the 3 layers and the 100 test fitting function it takes 1,5 hours of executing the code. For the 2 layers and 1 test fitting I just spent around 1min20sec.

4) Bonus : I saw some people using different layers one the web like dense(), dropout(), flatten().. Here is an example of possible convolutional autoencoder :

Entrée [19]:

```
import matplotlib.pyplot as plt
import numpy as np
from keras.datasets import cifar10
from keras.models import Model
from keras.layers import Input, Conv2D, MaxPooling2D, UpSampling2D, Flatten
from tensorflow.keras.optimizers import Adam

(x_train, _), (x_test, _) = cifar10.load_data()

print("train data shape :", x_train.shape)
print("test data shape :", x_test.shape)

#define the input shape
input_img = Input(shape=(32,32,3))
# Normalize the data
x_train = x_train / 255
x_test = x_test / 255


x = Conv2D(32, (3, 3), activation='relu', padding='same') (input_img)
x = MaxPooling2D((2, 2)) (x)

x = Conv2D(16, (3, 3), activation='relu', padding='same') (x)
encoded = MaxPooling2D((2, 2)) (x)

x = Conv2D(16, (3, 3), activation='relu', padding='same') (x)
x = UpSampling2D((2, 2)) (x)

x = Conv2D(32, (3, 3), activation='relu', padding='same') (x)

x = Dense(units=128,activation='relu')(x)

x = Dropout(0.5,noise_shape=None,seed=None)(x)



decoded = Conv2D(3, (3, 3), padding='same') (x)

model = (Model(input_img,decoded))
model.compile(optimizer = 'adam', loss ='mae', metrics=['accuracy'] )
model.summary()

print(x_train.shape)
print(x_test.shape)


history = model.fit(x_train,x_train,
           epochs = 1,
           batch_size = 50,
           validation_data = (x_test, x_test))

predict = model.predict(x_test)

def display(img1, img2, count = 6):
    n = count
    plt.figure()

    for i in range(n):
        ax = plt.subplot(2, n, i+1)
        plt.imshow(img1[300*i])
        ax = plt.subplot(2, n, i+1+n)
        plt.imshow(img2[300*i])
    plt.show()

display(x_test, predict)
```

```
train data shape : (50000, 32, 32, 3)
test data shape : (10000, 32, 32, 3)
Model: "model_13"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_19 (InputLayer)       [(None, 32, 32, 3)]       0

 conv2d_108 (Conv2D)         (None, 32, 32, 32)        896

 max_pooling2d_46 (MaxPoolin  (None, 16, 16, 32)       0
 g2D)

 conv2d_109 (Conv2D)         (None, 16, 16, 16)        4624

 conv2d_110 (Conv2D)         (None, 16, 16, 16)        2320

 up_sampling2d_28 (UpSamplin  (None, 32, 32, 16)       0
 g2D)

 conv2d_111 (Conv2D)         (None, 32, 32, 32)        4640

 dense_4 (Dense)             (None, 32, 32, 128)       4224

 dropout_4 (Dropout)         (None, 32, 32, 128)       0

 conv2d_112 (Conv2D)         (None, 32, 32, 3)         3459

=================================================================
Total params: 20,163
Trainable params: 20,163
Non-trainable params: 0
_________________________________________________________________
(50000, 32, 32, 3)
(10000, 32, 32, 3)
1000/1000 [==============================] - 224s 224ms/step - loss: 0.0497 - accuracy: 0.7356 - val_loss: 0.0287 - val_ac
curacy: 0.8335
313/313 [==============================] - 7s 22ms/step
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
```