TP - Traitement d’images

Séance2 : Prise en main d’un UNet

Metuarea Herearii - LARIS, Université d’Angers

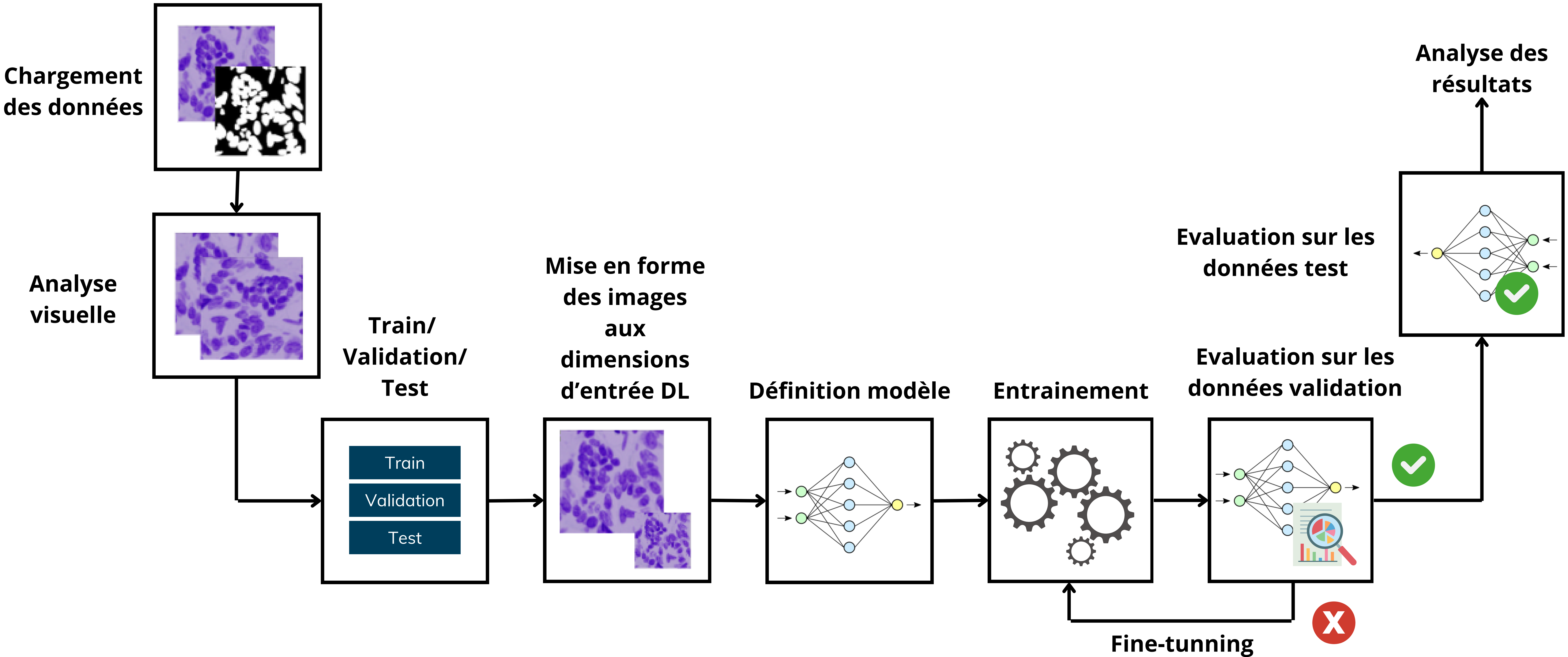
Master 2 Data Science, Université d’Angers

30 Janvier 2025

Plan de la séance

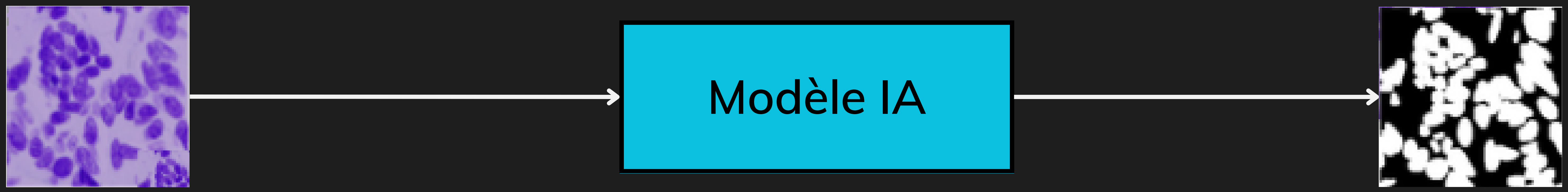
Guide pour concevoir un modèle DL de traitement d'images
TP

Guide pour concevoir un modèle de traitement d'images



Objectif de ce TP : Concevoir un modèle de segmentation d’images

Entrée **Traitement** **Sortie**



Prérequis : Importation des librairies et définition de fonctions utiles

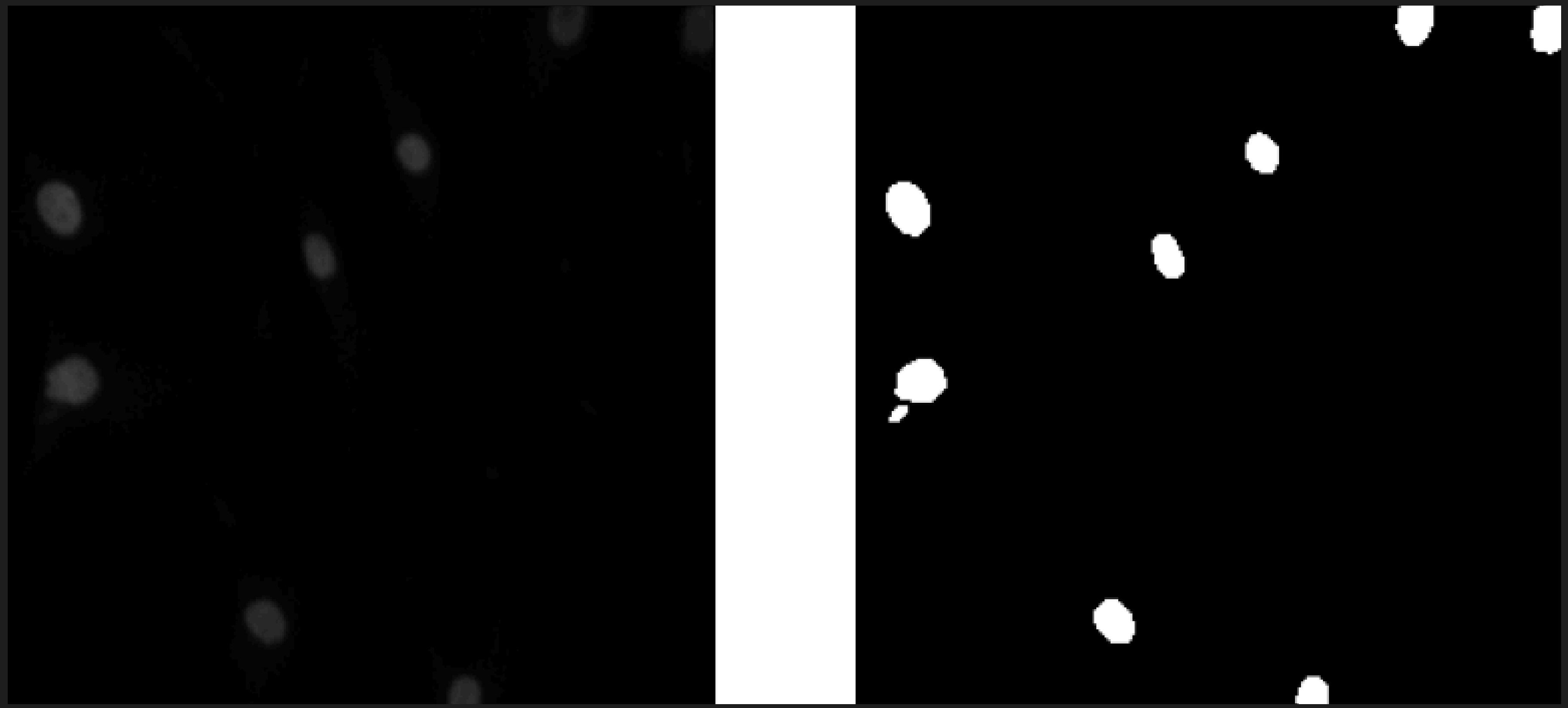
```python
import os
import sys
import random
import warnings

import numpy as np
import pandas as pd
import imageio
import matplotlib.pyplot as plt

from tqdm import tqdm
from itertools import chain
from skimage.io import imread, imshow, imread_collection, concatenate_images
from skimage.transform import resize
from skimage.morphology import label

from tensorflow.keras.models import Model, load_model
from tensorflow.keras.layers import Input, Conv2D, Conv2DTranspose, MaxPooling2D, concatenate
from tensorflow.keras.layers import BatchNormalization, Activation, Dense, Dropout
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
from tensorflow.keras import backend as K
from tensorflow.keras.optimizers import Adam
import cv2
import tensorflow as tf
import matplotlib.patches as mpatches
```

```python
def show_image(ix,x_train,y_train):
  fig = plt.figure(figsize=(10, 10))
  plt.subplot(121)
  plt.imshow(x_train[ix,:,:])
  plt.subplot(122)
  plt.imshow(y_train[ix, :, :, 0],cmap='gray')
  plt.axis('off')
  plt.show()

def get_data(path, train=True):
    """
    Loads and preprocesses image data.

    Args:
        path (str): Path to the directory containing the image data.
        train (bool, optional): Flag indicating if the data is for training.
                                Defaults to True.

    Returns:
        tuple or ndarray: If train is True, returns a tuple containing the
                          image data (X) and corresponding masks (Y).
                          If train is False, returns only the image data (X).
    """
    # Get the list of image IDs
    ids = next(os.walk(path))[1]

    # Initialize arrays to store image data and masks
    X = np.zeros((len(ids), IMG_HEIGHT, IMG_WIDTH, IMG_CHANNELS), dtype=np.uint8)

    # Initialize masks array only if train is True
    if train:
        Y = np.zeros((len(ids), IMG_HEIGHT, IMG_WIDTH, 1), dtype=bool)

    print('Getting and resizing images ... ')
    sys.stdout.flush()

    # Iterate through each image ID
    for n, id_ in tqdm(enumerate(ids), total=len(ids)):
        path_new = path + id_

        # Read and resize the image
        img = imread(path_new + '/images/' + id_ + '.png')[:,:,:IMG_CHANNELS]
        img = resize(img, (IMG_HEIGHT, IMG_WIDTH), mode='constant', preserve_range=True)
```

I- Importation et visualisation des données

Train

Test

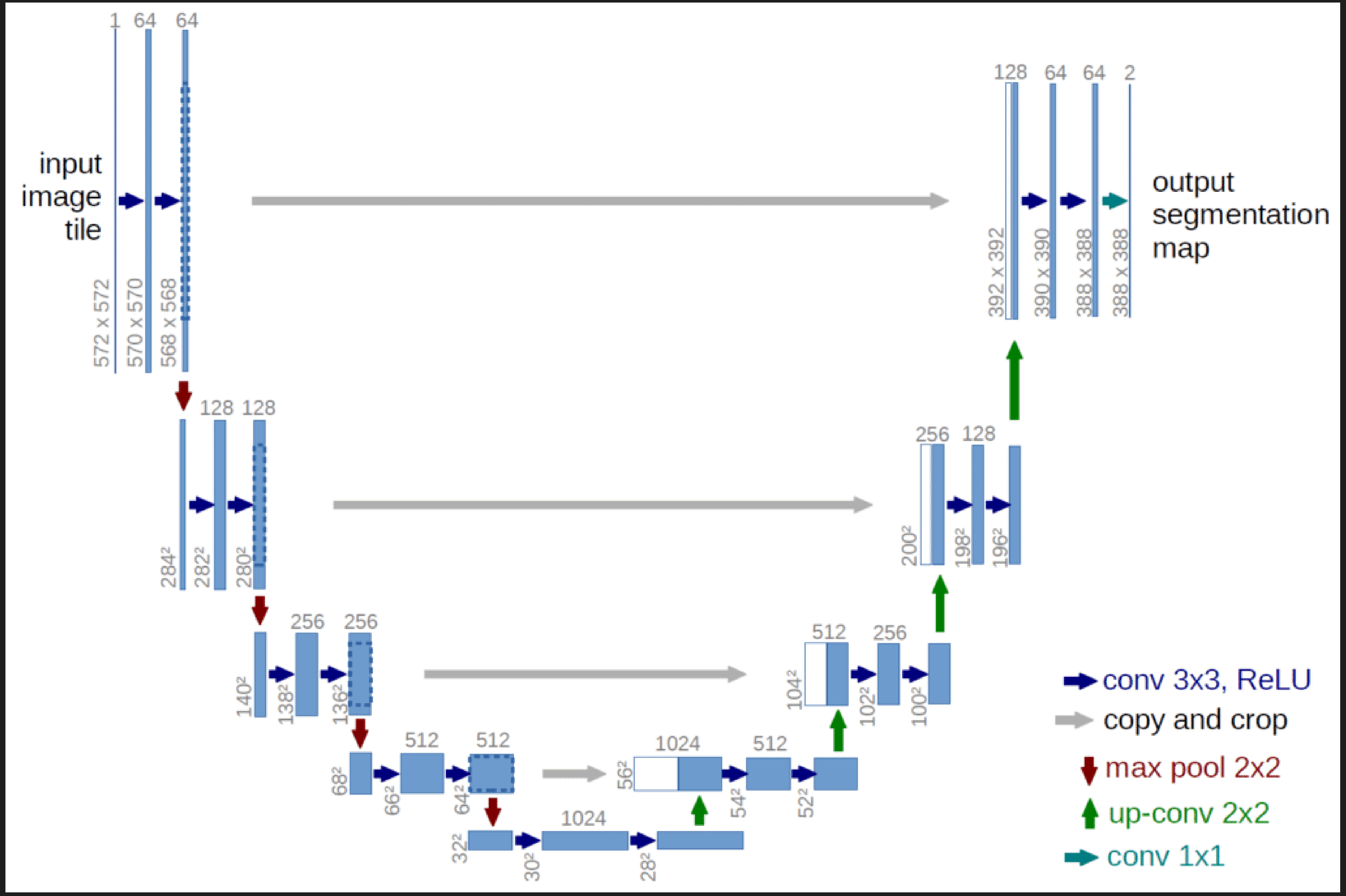


V- Définition du modèle

```
#Each block of u-net architecture consist of two Convolution layers
# These two layers are written in a function to make our code clean
def conv2d_block(input_tensor, n_filters,kernel_size=3):
    x = Conv2D(filters=n_filters,kernel_size=(kernel_size, kernel_size),activation='relu', padding="same")(input_tensor)
    x = Conv2D(filters=n_filters,kernel_size=(kernel_size, kernel_size),activation='relu', padding="same")(x)
    return x

# The u-net architecture consists of contracting and expansive paths which
# shrink and expands the inout image respectivly.
# Output image have the same size of input image
def get_unet(input_img, n_filters):
    # contracting path
    c1 = conv2d_block(input_img, n_filters=n_filters*4, kernel_size=3) #The first block of U-net
    p1 = MaxPooling2D((2, 2)) (c1)

    c2 = conv2d_block(p1, n_filters=n_filters*8, kernel_size=3)
    p2 = MaxPooling2D((2, 2)) (c2)

    c3 = conv2d_block(p2, n_filters=n_filters*16, kernel_size=3)
    p3 = MaxPooling2D((2, 2)) (c3)

    c4 = conv2d_block(p3, n_filters=n_filters*32, kernel_size=3)
    p4 = MaxPooling2D(pool_size=(2, 2)) (c4)

    c5 = conv2d_block(p4, n_filters=n_filters*64, kernel_size=3) # last layer on encoding path

    # expansive path
    u6 = Conv2DTranspose(n_filters*32, (3, 3), strides=(2, 2), padding='same') (c5) #upsampling included
    u6 = concatenate([u6, c4])
    c6 = conv2d_block(u6, n_filters=n_filters*32, kernel_size=3)

    u7 = Conv2DTranspose(n_filters*16, (3, 3), strides=(2, 2), padding='same') (c6)
    u7 = concatenate([u7, c3])
    c7 = conv2d_block(u7, n_filters=n_filters*16, kernel_size=3)

    u8 = Conv2DTranspose(n_filters*8, (3, 3), strides=(2, 2), padding='same') (c7)
    u8 = concatenate([u8, c2])
    c8 = conv2d_block(u8, n_filters=n_filters*8, kernel_size=3)

    u9 = Conv2DTranspose(n_filters*4, (3, 3), strides=(2, 2), padding='same') (c8)
    u9 = concatenate([u9, c1], axis=3)
    c9 = conv2d_block(u9, n_filters=n_filters*4, kernel_size=3)

    outputs = Conv2D(1, (1, 1), activation='sigmoid') (c9)
    model = Model(inputs=[input_img], outputs=[outputs])
    return model
```

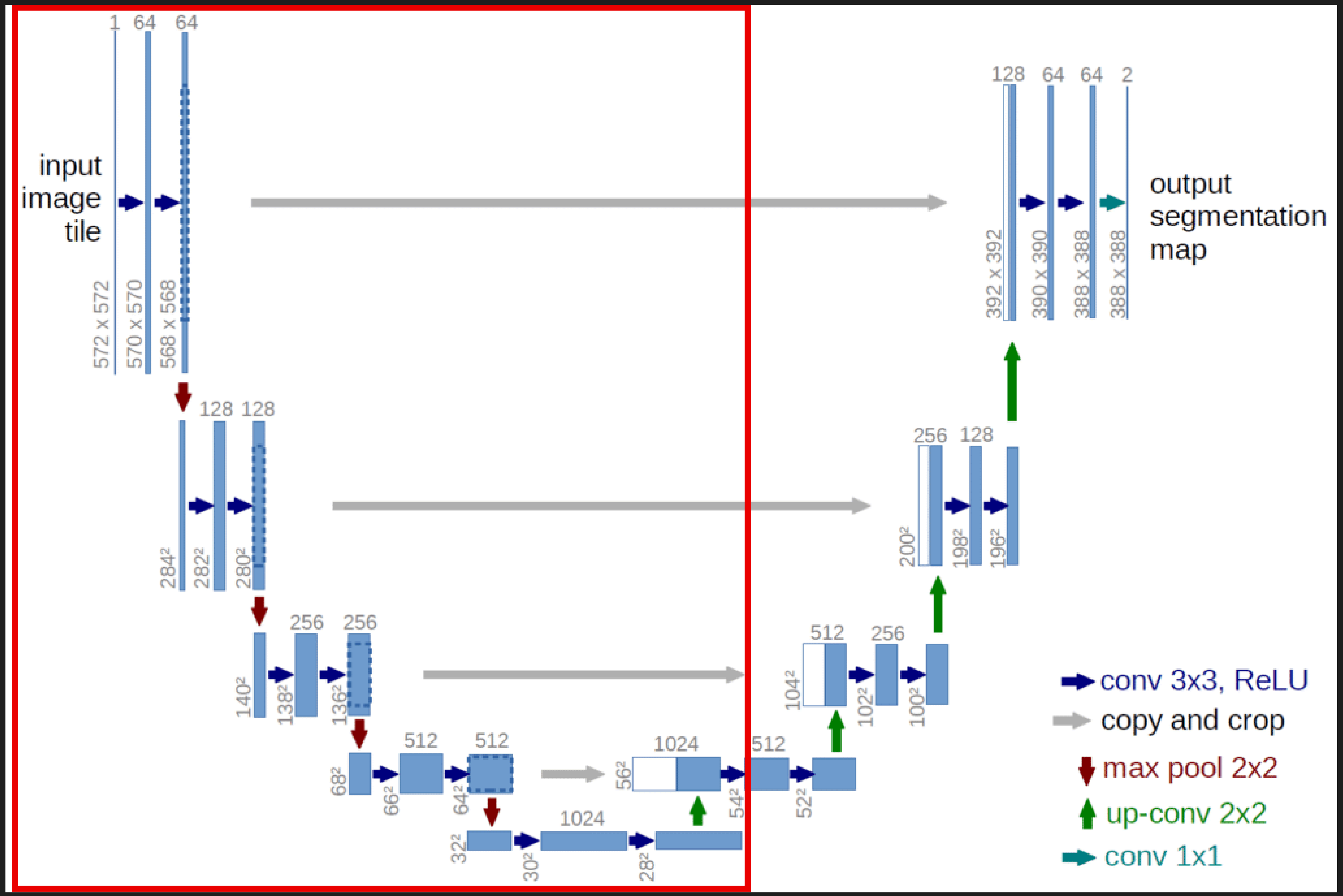
V- Définition du modèle

```
#Each block of u-net architecture consist of two Convolution layers
# These two layers are written in a function to make our code clean
def conv2d_block(input_tensor, n_filters,kernel_size=3):
    x = Conv2D(filters=n_filters,kernel_size=(kernel_size, kernel_size),activation='relu', padding="same")(input_tensor)
    x = Conv2D(filters=n_filters,kernel_size=(kernel_size, kernel_size),activation='relu', padding="same")(x)
    return x

# The u-net architecture consists of contracting and expansive paths which
# shrink and expands the inout image respectivly.
# Output image have the same size of input image
def get_unet(input_img, n_filters):
    # contracting path
    c1 = conv2d_block(input_img, n_filters=n_filters*4, kernel_size=3) #The first block of U-net
    p1 = MaxPooling2D((2, 2)) (c1)

    c2 = conv2d_block(p1, n_filters=n_filters*8, kernel_size=3)
    p2 = MaxPooling2D((2, 2)) (c2)

    c3 = conv2d_block(p2, n_filters=n_filters*16, kernel_size=3)
    p3 = MaxPooling2D((2, 2)) (c3)

    c4 = conv2d_block(p3, n_filters=n_filters*32, kernel_size=3)
    p4 = MaxPooling2D(pool_size=(2, 2)) (c4)

    c5 = conv2d_block(p4, n_filters=n_filters*64, kernel_size=3) # last layer on encoding path

    # expansive path
    u6 = Conv2DTranspose(n_filters*32, (3, 3), strides=(2, 2), padding='same') (c5) #upsampling included
    u6 = concatenate([u6, c4])
    c6 = conv2d_block(u6, n_filters=n_filters*32, kernel_size=3)

    u7 = Conv2DTranspose(n_filters*16, (3, 3), strides=(2, 2), padding='same') (c6)
    u7 = concatenate([u7, c3])
    c7 = conv2d_block(u7, n_filters=n_filters*16, kernel_size=3)

    u8 = Conv2DTranspose(n_filters*8, (3, 3), strides=(2, 2), padding='same') (c7)
    u8 = concatenate([u8, c2])
    c8 = conv2d_block(u8, n_filters=n_filters*8, kernel_size=3)

    u9 = Conv2DTranspose(n_filters*4, (3, 3), strides=(2, 2), padding='same') (c8)
    u9 = concatenate([u9, c1], axis=3)
    c9 = conv2d_block(u9, n_filters=n_filters*4, kernel_size=3)

    outputs = Conv2D(1, (1, 1), activation='sigmoid') (c9)
    model = Model(inputs=[input_img], outputs=[outputs])
    return model
```



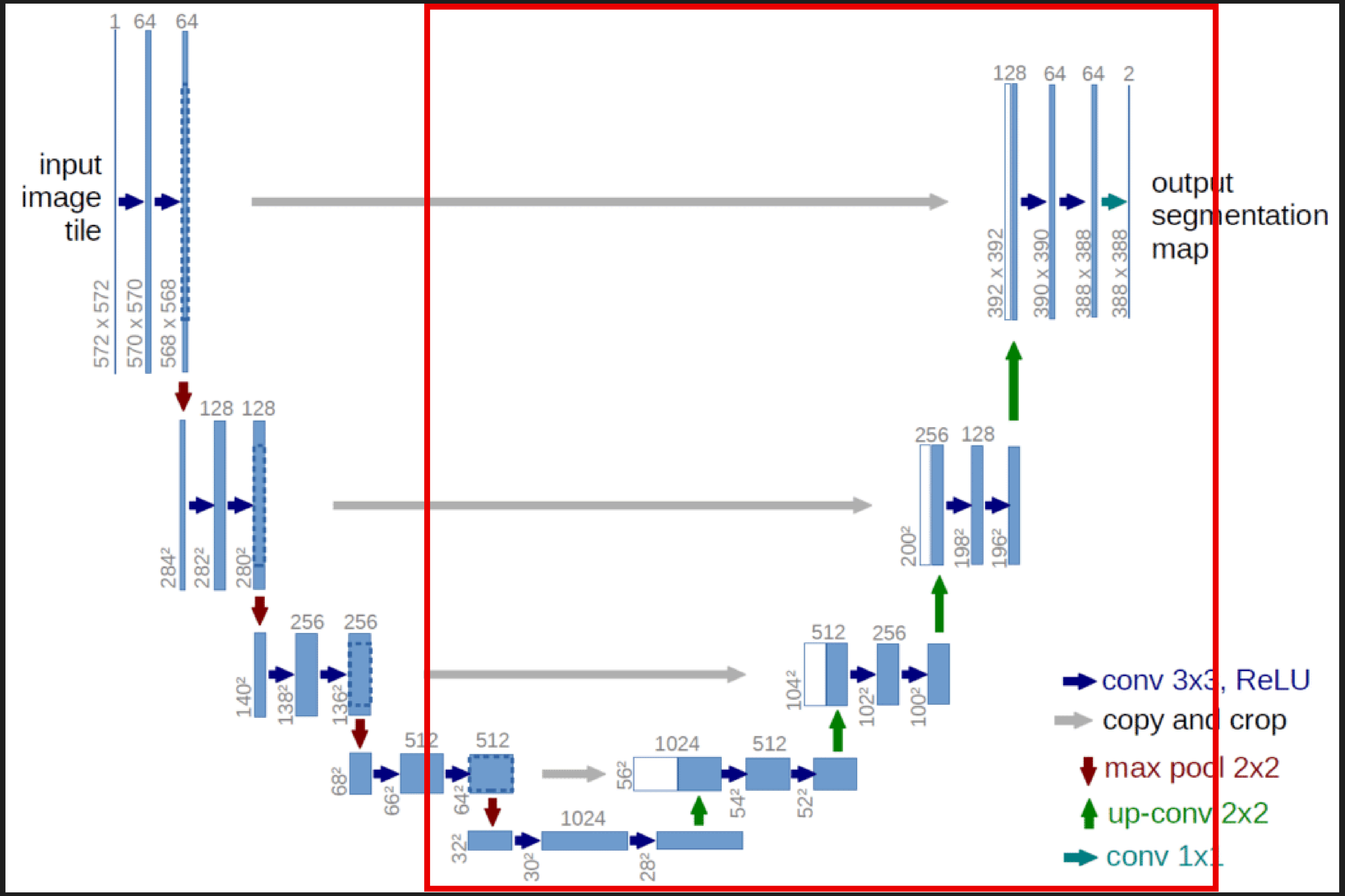
V- Définition du modèle

```
#Each block of u-net architecture consist of two Convolution layers
# These two layers are written in a function to make our code clean
def conv2d_block(input_tensor, n_filters,kernel_size=3):
    x = Conv2D(filters=n_filters,kernel_size=(kernel_size, kernel_size),activation='relu', padding="same")(input_tensor)
    x = Conv2D(filters=n_filters,kernel_size=(kernel_size, kernel_size),activation='relu', padding="same")(x)
    return x

# The u-net architecture consists of contracting and expansive paths which
# shrink and expands the inout image respectivly.
# Output image have the same size of input image
def get_unet(input_img, n_filters):
    # contracting path
    c1 = conv2d_block(input_img, n_filters=n_filters*4, kernel_size=3) #The first block of U-net
    p1 = MaxPooling2D((2, 2)) (c1)

    c2 = conv2d_block(p1, n_filters=n_filters*8, kernel_size=3)
    p2 = MaxPooling2D((2, 2)) (c2)

    c3 = conv2d_block(p2, n_filters=n_filters*16, kernel_size=3)
    p3 = MaxPooling2D((2, 2)) (c3)

    c4 = conv2d_block(p3, n_filters=n_filters*32, kernel_size=3)
    p4 = MaxPooling2D(pool_size=(2, 2)) (c4)

    c5 = conv2d_block(p4, n_filters=n_filters*64, kernel_size=3) # last layer on encoding path

    # expansive path
    u6 = Conv2DTranspose(n_filters*32, (3, 3), strides=(2, 2), padding='same') (c5) #upsampling included
    u6 = concatenate([u6, c4])
    c6 = conv2d_block(u6, n_filters=n_filters*32, kernel_size=3)

    u7 = Conv2DTranspose(n_filters*16, (3, 3), strides=(2, 2), padding='same') (c6)
    u7 = concatenate([u7, c3])
    c7 = conv2d_block(u7, n_filters=n_filters*16, kernel_size=3)

    u8 = Conv2DTranspose(n_filters*8, (3, 3), strides=(2, 2), padding='same') (c7)
    u8 = concatenate([u8, c2])
    c8 = conv2d_block(u8, n_filters=n_filters*8, kernel_size=3)

    u9 = Conv2DTranspose(n_filters*4, (3, 3), strides=(2, 2), padding='same') (c8)
    u9 = concatenate([u9, c1], axis=3)
    c9 = conv2d_block(u9, n_filters=n_filters*4, kernel_size=3)

    outputs = Conv2D(1, (1, 1), activation='sigmoid') (c9)
    model = Model(inputs=[input_img], outputs=[outputs])
    return model
```



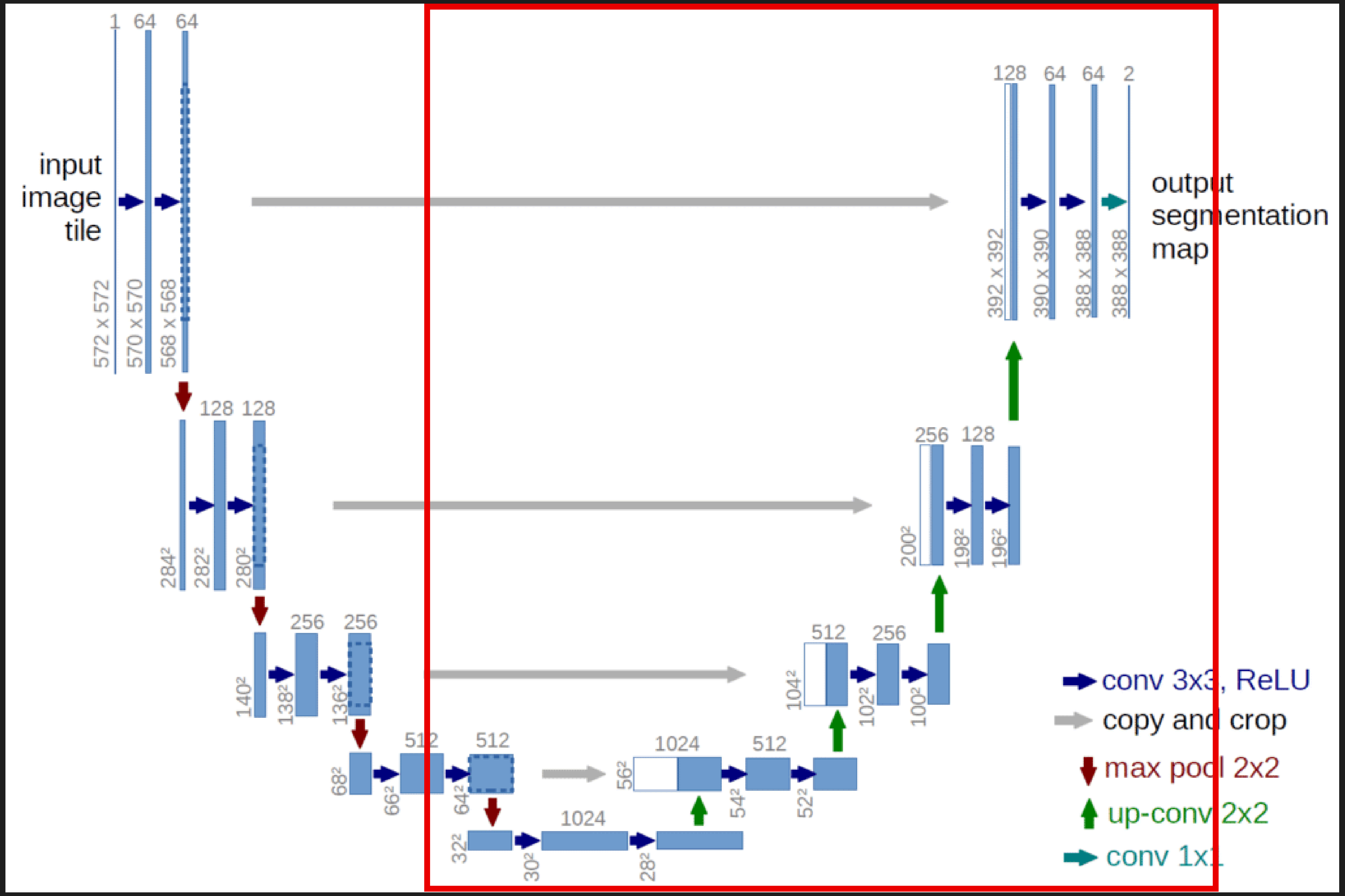
V- Définition du modèle

```python
#Each block of u-net architecture consist of two Convolution layers
# These two layers are written in a function to make our code clean
def conv2d_block(input_tensor, n_filters,kernel_size=3):
    x = Conv2D(filters=n_filters,kernel_size=(kernel_size, kernel_size),activation='relu', padding="same")(input_tensor)
    x = Conv2D(filters=n_filters,kernel_size=(kernel_size, kernel_size),activation='relu', padding="same")(x)
    return x

# The u-net architecture consists of contracting and expansive paths which
# shrink and expands the inout image respectivly.
# Output image have the same size of input image
def get_unet(input_img, n_filters):
    # contracting path
    c1 = conv2d_block(input_img, n_filters=n_filters*4, kernel_size=3) #The first block of U-net
    p1 = MaxPooling2D((2, 2)) (c1)

    c2 = conv2d_block(p1, n_filters=n_filters*8, kernel_size=3)
    p2 = MaxPooling2D((2, 2)) (c2)

    c3 = conv2d_block(p2, n_filters=n_filters*16, kernel_size=3)
    p3 = MaxPooling2D((2, 2)) (c3)

    c4 = conv2d_block(p3, n_filters=n_filters*32, kernel_size=3)
    p4 = MaxPooling2D(pool_size=(2, 2)) (c4)

    c5 = conv2d_block(p4, n_filters=n_filters*64, kernel_size=3) # last layer on encoding path

    # expansive path
    u6 = Conv2DTranspose(n_filters*32, (3, 3), strides=(2, 2), padding='same') (c5) #upsampling included
    u6 = concatenate([u6, c4])
    c6 = conv2d_block(u6, n_filters=n_filters*32, kernel_size=3)

    u7 = Conv2DTranspose(n_filters*16, (3, 3), strides=(2, 2), padding='same') (c6)
    u7 = concatenate([u7, c3])
    c7 = conv2d_block(u7, n_filters=n_filters*16, kernel_size=3)

    u8 = Conv2DTranspose(n_filters*8, (3, 3), strides=(2, 2), padding='same') (c7)
    u8 = concatenate([u8, c2])
    c8 = conv2d_block(u8, n_filters=n_filters*8, kernel_size=3)

    u9 = Conv2DTranspose(n_filters*4, (3, 3), strides=(2, 2), padding='same') (c8)
    u9 = concatenate([u9, c1], axis=3)
    c9 = conv2d_block(u9, n_filters=n_filters*4, kernel_size=3)

    outputs = Conv2D(1, (1, 1), activation='sigmoid') (c9)
    model = Model(inputs=[input_img], outputs=[outputs])
    return model
```

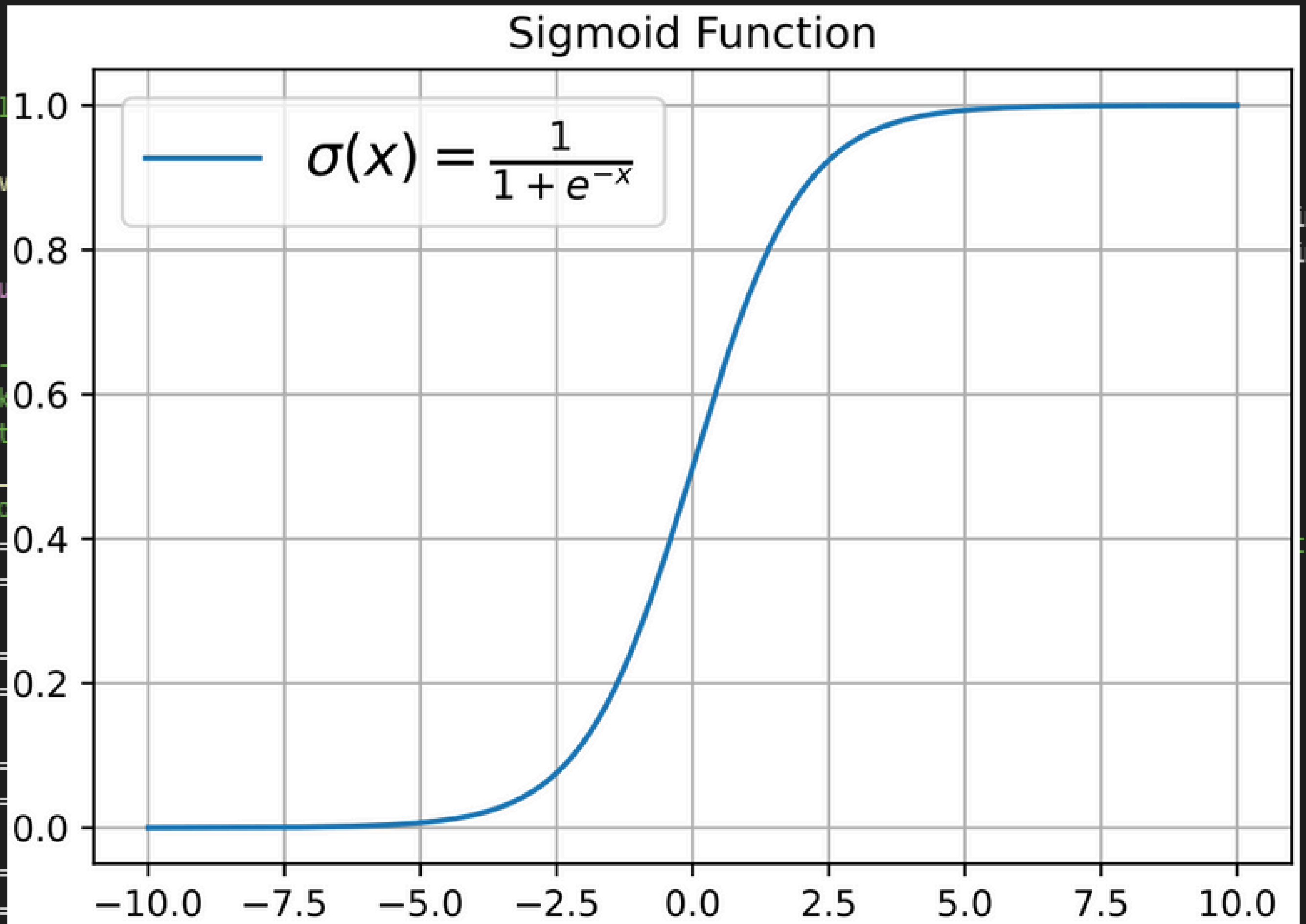


V- Définition du modèle

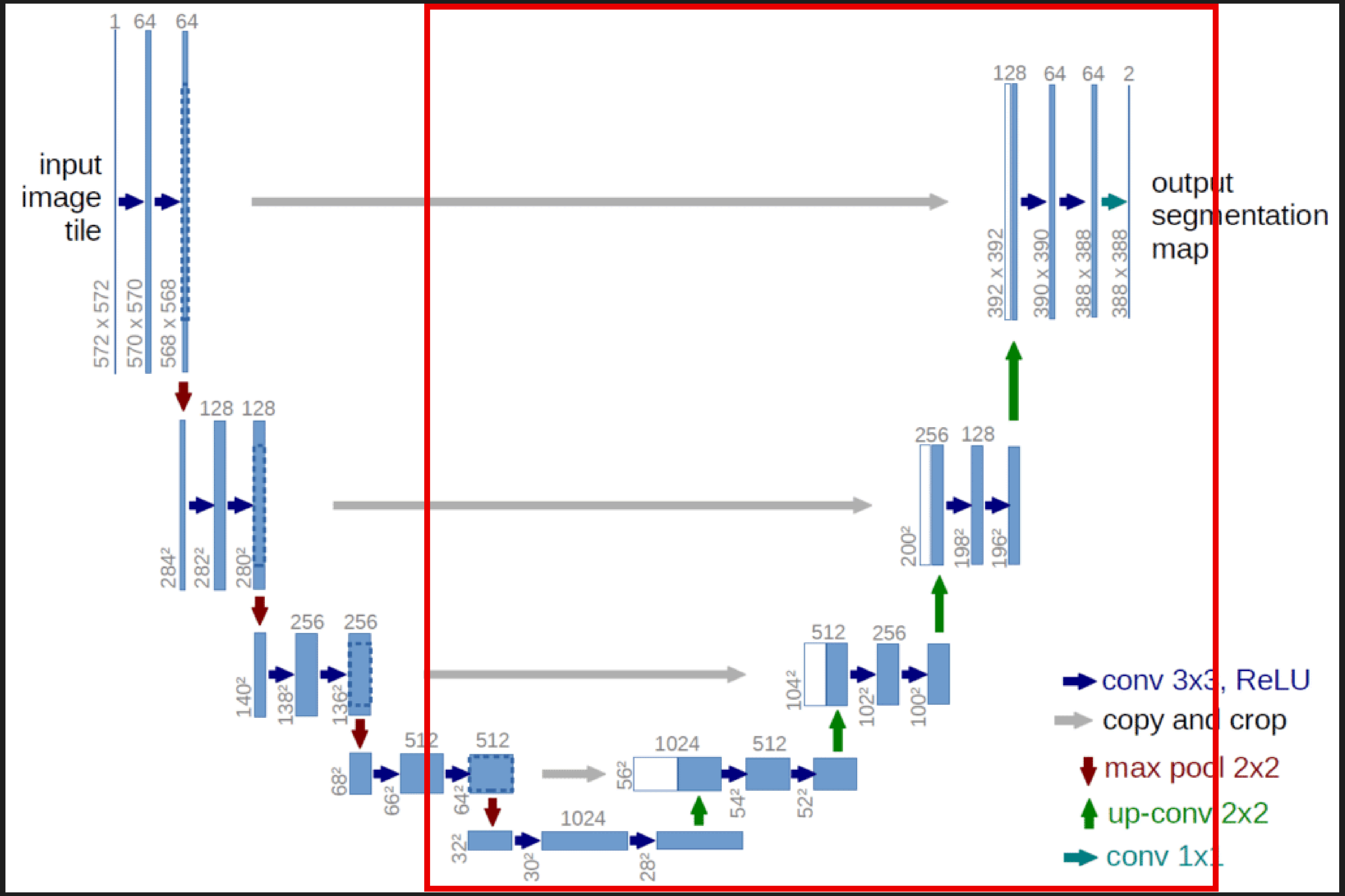
```python
#Each block of u-net architecture consist of two Convolution layers
# These two layers are written in a function to make our code clean
def conv2d_block(input_tensor, n_filters,kernel_size=3):
    x = Conv2D(filters=n_filters,kernel_size=(kernel_size, kernel_size),activation='relu', padding="same")(input_tensor)
    x = Conv2D(filters=n_filters,kernel_size=(kernel_size, kernel_size),activation='relu', padding="same")(x)
    return x

# The u-net architecture consists of contracting and expansive paths which
# shrink and expands the inout image respectivly.
# Output image have the same size of input image
def get_unet(input_img, n_filters):
    # contracting path
    c1 = conv2d_block(input_img, n_filters=n_filters*4, kernel_size=3) #The first block of U-net
    p1 = MaxPooling2D((2, 2)) (c1)

    c2 = conv2d_block(p1, n_filters=n_filters*8, kernel_size=3)
    p2 = MaxPooling2D((2, 2)) (c2)

    c3 = conv2d_block(p2, n_filters=n_filters*16, kernel_size=3)
    p3 = MaxPooling2D((2, 2)) (c3)

    c4 = conv2d_block(p3, n_filters=n_filters*32, kernel_size=3)
    p4 = MaxPooling2D(pool_size=(2, 2)) (c4)

    c5 = conv2d_block(p4, n_filters=n_filters*64, kernel_size=3) # last layer on encoding path

    # expansive path
    u6 = Conv2DTranspose(n_filters*32, (3, 3), strides=(2, 2), padding='same') (c5) #upsampling included
    u6 = concatenate([u6, c4])
    c6 = conv2d_block(u6, n_filters=n_filters*32, kernel_size=3)

    u7 = Conv2DTranspose(n_filters*16, (3, 3), strides=(2, 2), padding='same') (c6)
    u7 = concatenate([u7, c3])
    c7 = conv2d_block(u7, n_filters=n_filters*16, kernel_size=3)

    u8 = Conv2DTranspose(n_filters*8, (3, 3), strides=(2, 2), padding='same') (c7)
    u8 = concatenate([u8, c2])
    c8 = conv2d_block(u8, n_filters=n_filters*8, kernel_size=3)

    u9 = Conv2DTranspose(n_filters*4, (3, 3), strides=(2, 2), padding='same') (c8)
    u9 = concatenate([u9, c1], axis=3)
    c9 = conv2d_block(u9, n_filters=n_filters*4, kernel_size=3)

    outputs = Conv2D(1, (1, 1), activation='sigmoid') (c9)
    model = Model(inputs=[input_img], outputs=[outputs])
    return model
```



V- Définition du modèle

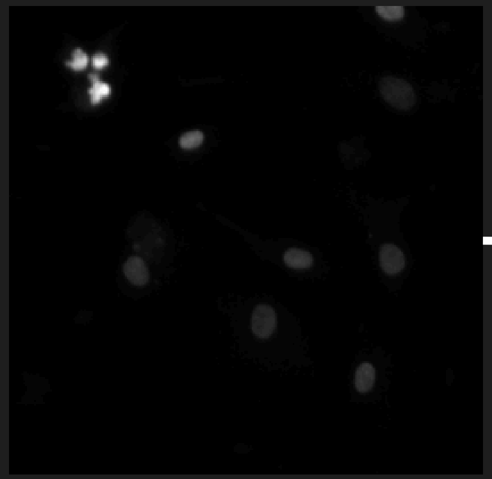



```
#Each bl
# These
def conv                                                                        ing="same")(input_tensor)
    x =                                                                         ing="same")(x)
    x =
    retu

# The u-
# shrink
# Output
def get_
    # co
    c1 =
    p1 =

    c2 =
    p2 =

    c3 =
    p3 =

    c4 =
    p4 =

    c5 = conv2d_block(p4, n_filters=n_filters*64, kernel_size=3) # last layer on encoding path

    # expansive path
    u6 = Conv2DTranspose(n_filters*32, (3, 3), strides=(2, 2), padding='same') (c5) #upsampling included
    u6 = concatenate([u6, c4])
    c6 = conv2d_block(u6, n_filters=n_filters*32, kernel_size=3)

    u7 = Conv2DTranspose(n_filters*16, (3, 3), strides=(2, 2), padding='same') (c6)
    u7 = concatenate([u7, c3])
    c7 = conv2d_block(u7, n_filters=n_filters*16, kernel_size=3)

    u8 = Conv2DTranspose(n_filters*8, (3, 3), strides=(2, 2), padding='same') (c7)
    u8 = concatenate([u8, c2])
    c8 = conv2d_block(u8, n_filters=n_filters*8, kernel_size=3)

    u9 = Conv2DTranspose(n_filters*4, (3, 3), strides=(2, 2), padding='same') (c8)
    u9 = concatenate([u9, c1], axis=3)
    c9 = conv2d_block(u9, n_filters=n_filters*4, kernel_size=3)

    outputs = Conv2D(1, (1, 1), activation='sigmoid') (c9)
    model = Model(inputs=[input_img], outputs=[outputs])
    return model
```

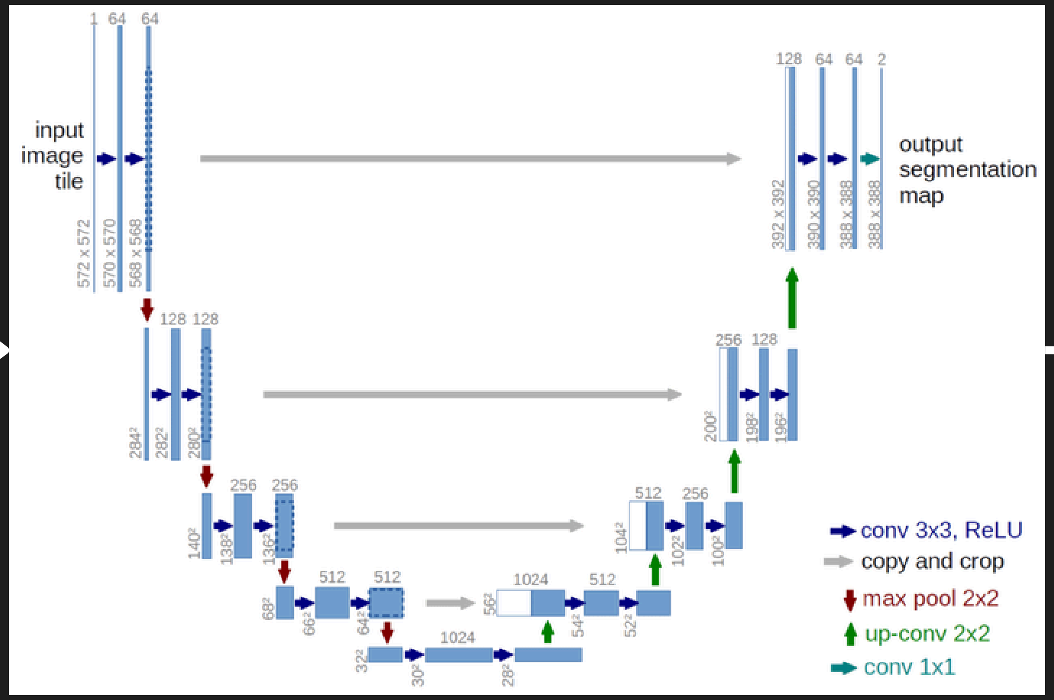


Résumé

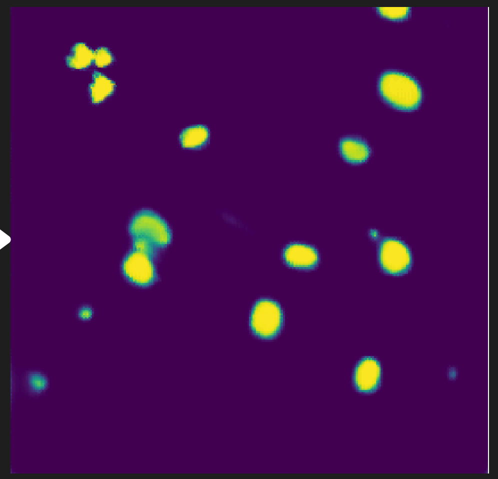
Input



UNet

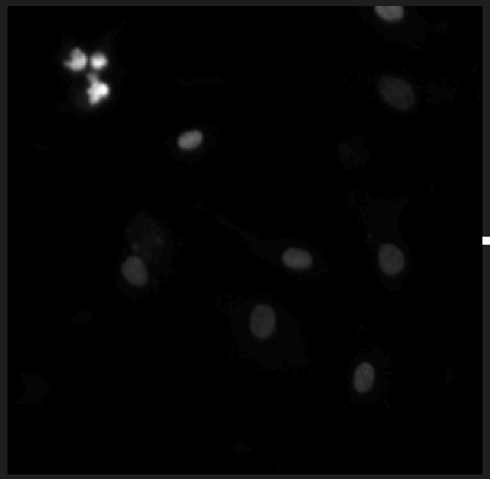


Output

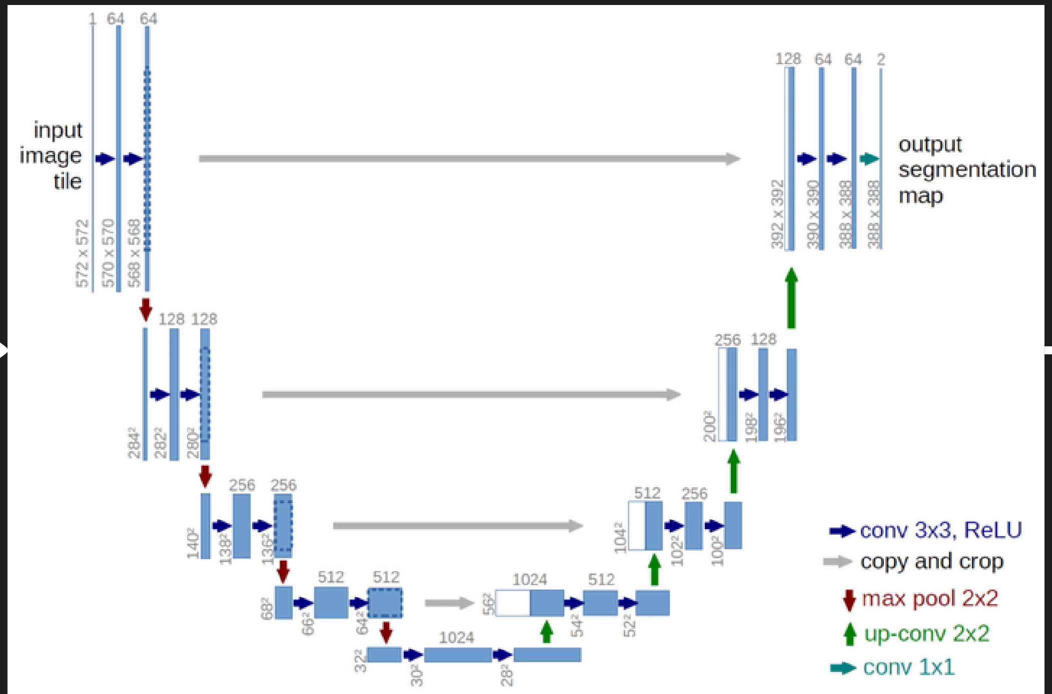


Résumé

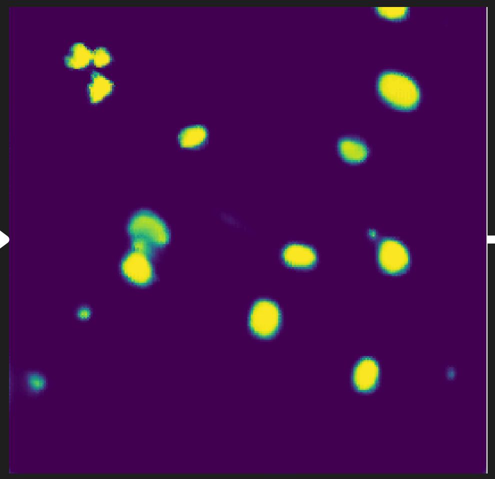
Input



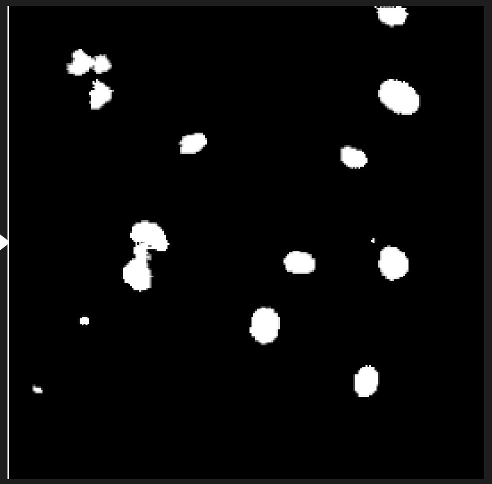
UNet



Output



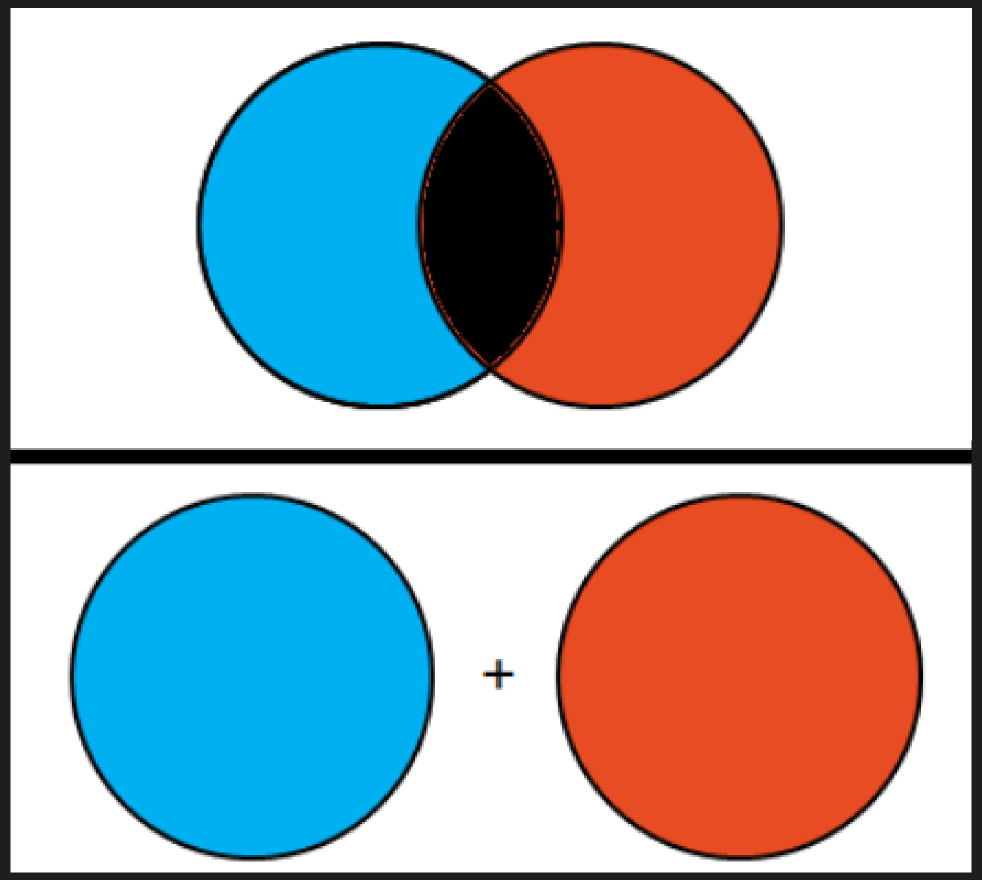
Post-traitement



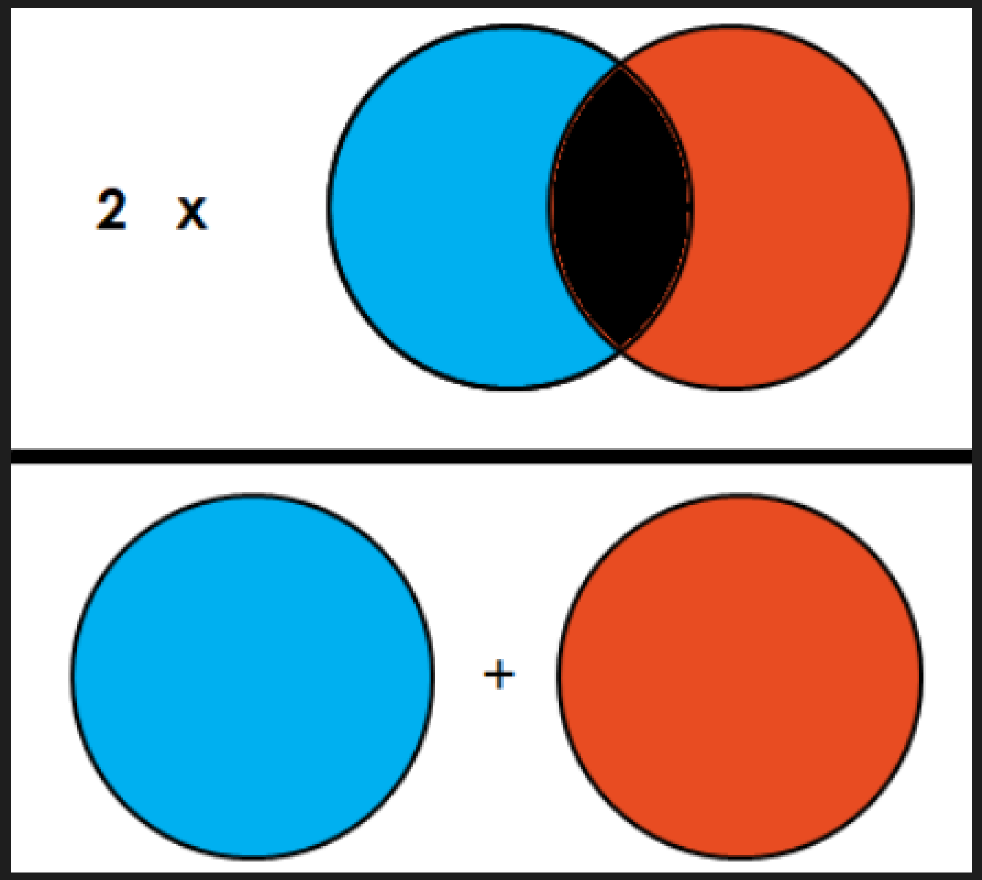
**Seuillage sur
la carte de caractéristique
pour obtenir un masque binaire**

Métrique

IoU score



Dice score



Fonction de perte

Binary cross entropy

$$\text{BCE} = -\frac{1}{N}\sum_{i=1}^{N}\left[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)\right]$$

N is the number of samples,

y_i is the true binary label (0 or 1),

$\hat{y}_i$ is the predicted probability for the positive class (i.e., the model's output),

$\log$ is the natural logarithm.

Weighted binary cross entropy

$$\text{WBCE} = -\frac{1}{N}\sum_{i=1}^{N}\left[w_+ y_i \log(\hat{y}_i) + w_-(1 - y_i)\log(1 - \hat{y}_i)\right]$$

N is the number of samples,

y_i is the true label (0 or 1),

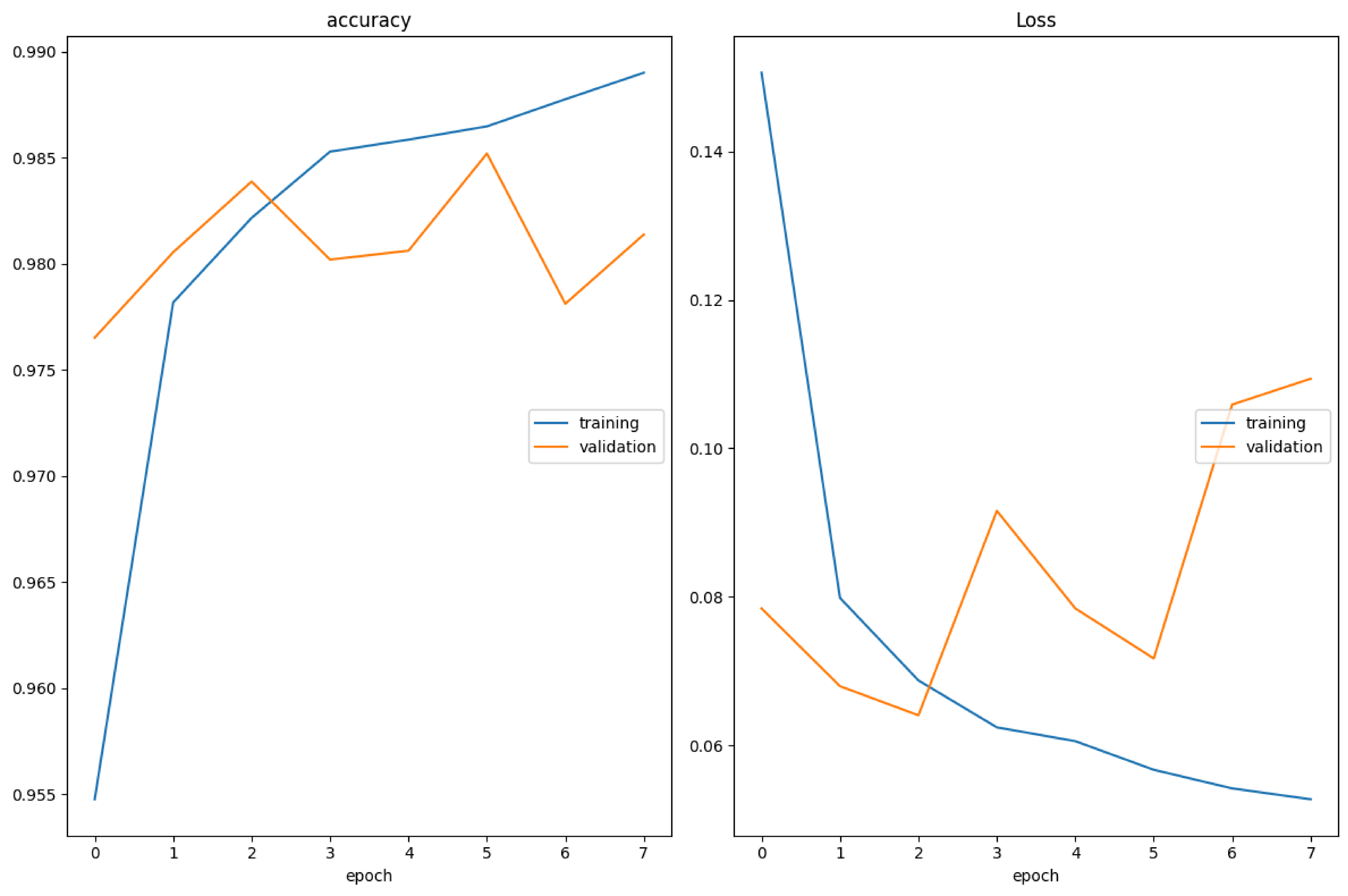
$\hat{y}_i$ is the predicted probability of the positive class,

w_+ is the weight for the positive class (label 1),

w_- is the weight for the negative class (label 0),

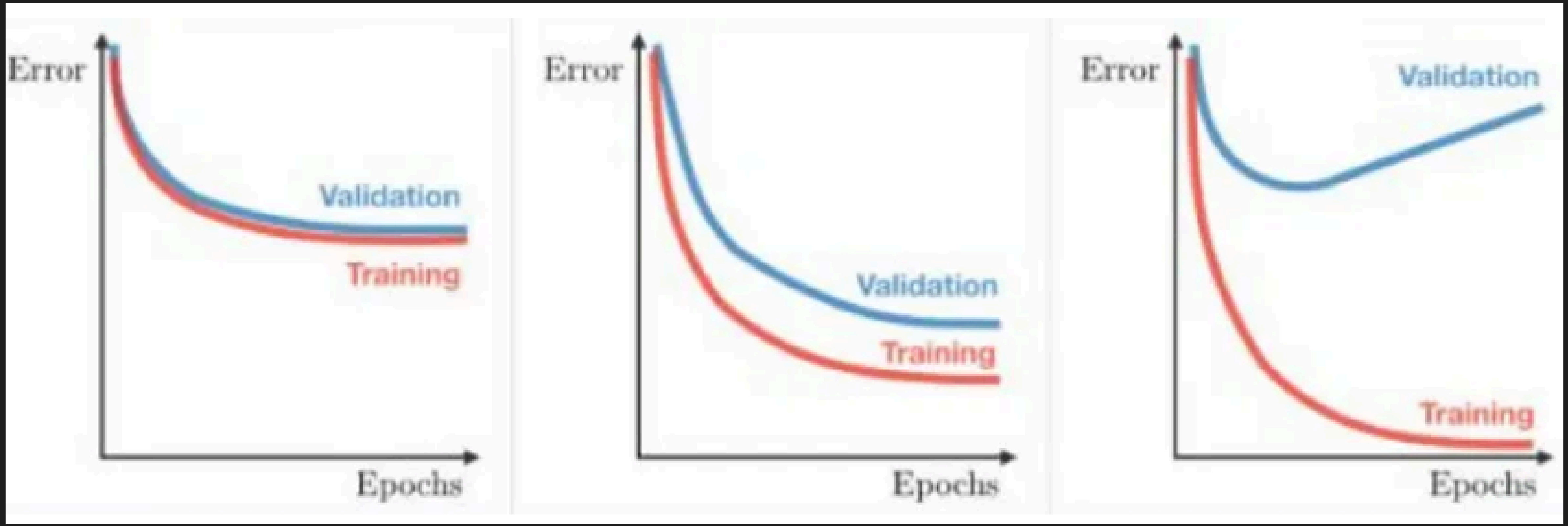
$\log$ is the natural logarithm.

Entrainement du modèle et évaluation sur les données de validation



Entrainement du modèle et évaluation sur les données de validation

Analyse des courbes d’entrainement

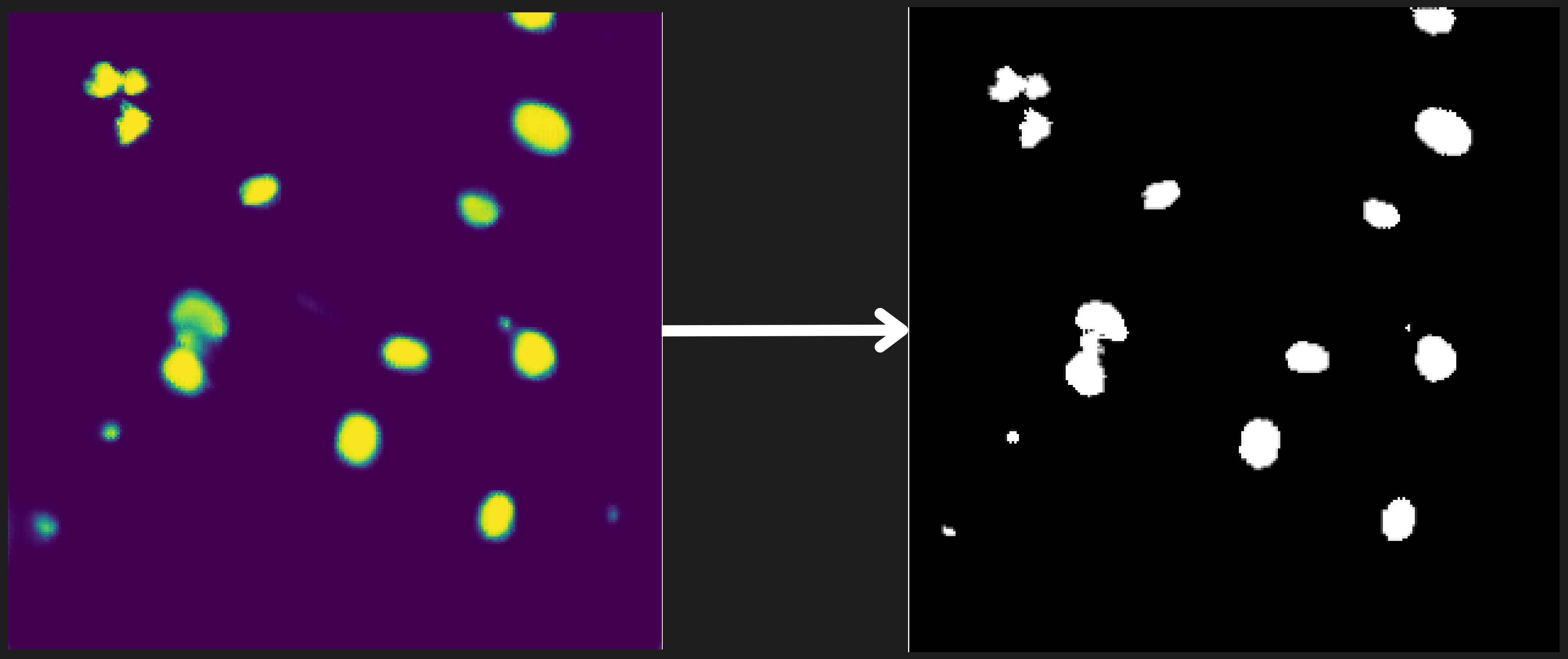


Underfitting **Good fitting** **Overfitting**

Si overfitting ou underfitting:
1- Appliquer un fine tunning
2- Réinitialiser le modèle
3- Lancer un nouveau entrainement

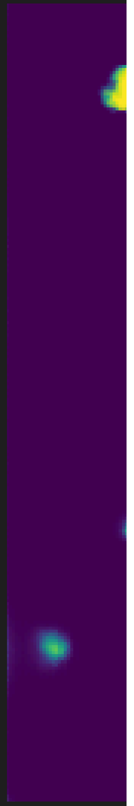
Fine tunning (ou réglage sans fin)
Ajustement des hyperparamètres du modèle ou de l’entrainement

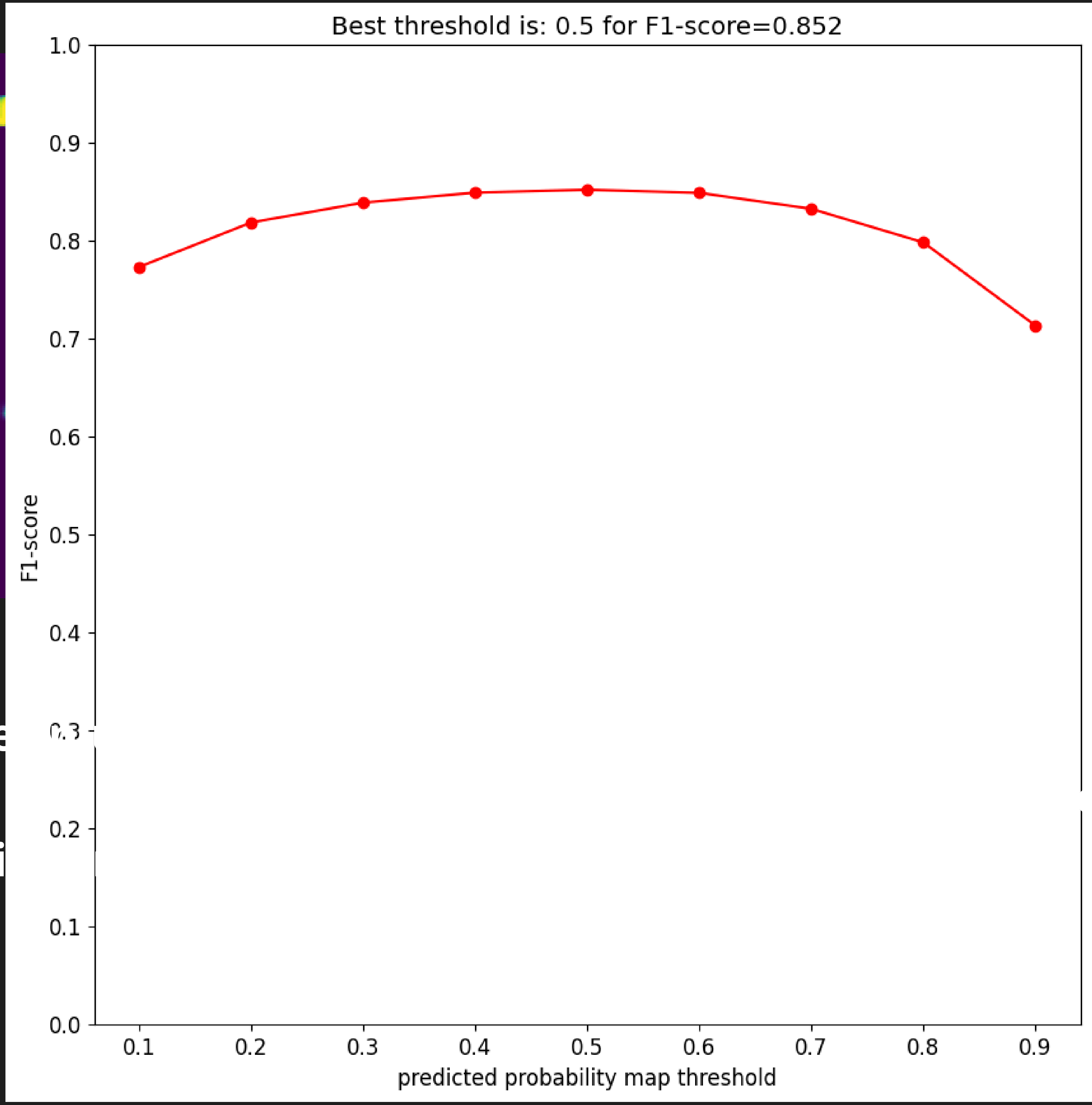
Recherche du seuil pour segmenter les cartes de caractéristiques

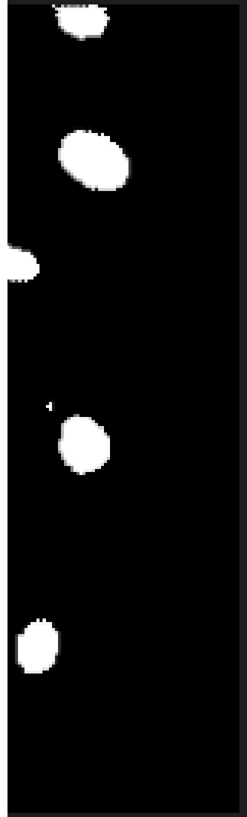


Fixer plusieurs seuils : 0, 0.1, 0.2, ... , 0.9, 1
Evaluer avec f1 score les segmentations et les vérités terrains
Meilleur seuil est associé au plus grand f1 score

Recherche du seuil pour segmenter les cartes de caractéristiques







Fixe

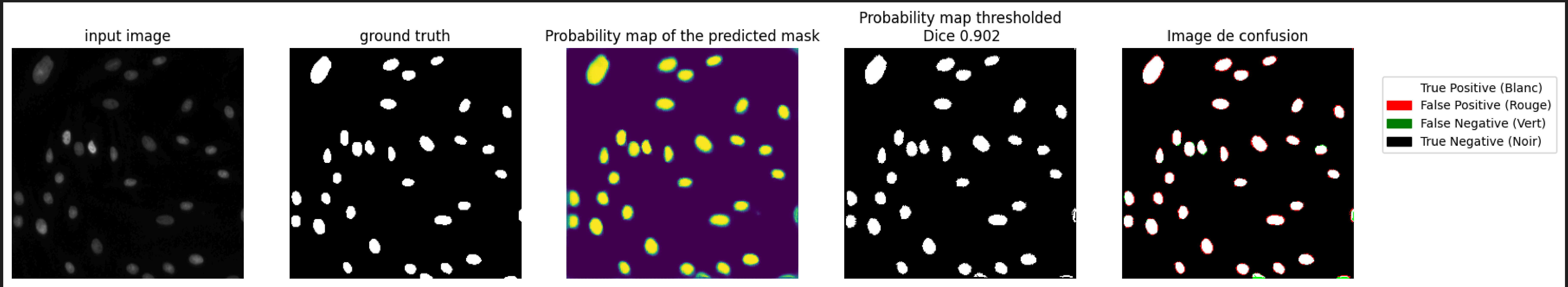
Eva … és terrains

Mei

Evaluation sur les données test

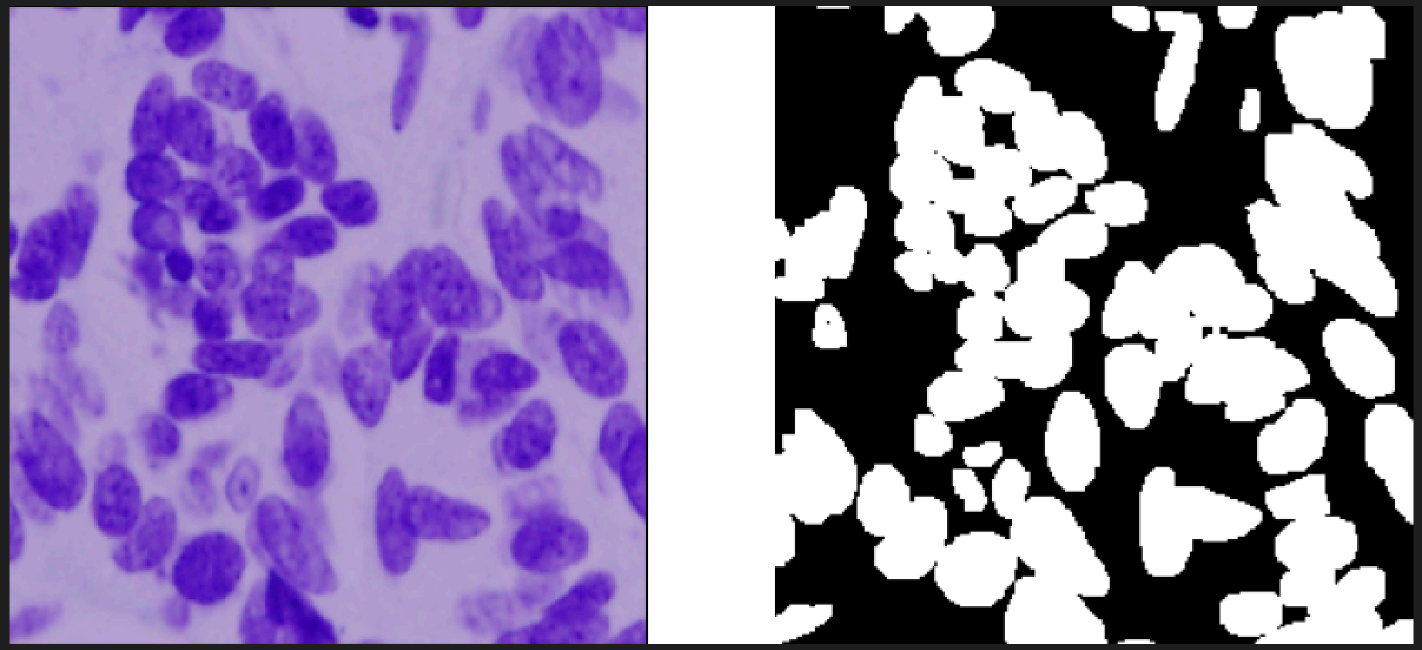
```
5/5 ━━━━━━━━━━━━━━━━━━━━ 29s 5s/step
Dice coefficient: 0.959
```

Analyse visuelle avec l'image confusion



Prochaines séances

Séance 2 -
i) Prise en main d'un code de segmentation d'images UNet
ii) Techniques classiques pour améliorer un modèle : focus sur la Data Augmentation et le transfert learning



Séance 3 - Prise en main de Napari
Séance 4 - Évaluation