Atelier 3: Segmentation d'Images avec U-Net

Objectif du TP

- Comprendre le principe de la segmentation d'images
- Découvrir l'architecture U-Net
- Appliquer U-Net sur un dataset pour segmenter des objets
- Visualiser les résultats

1) Définitions

- ### Segmentation d'images

Tâche de vision par ordinateur qui consiste à **attribuer une classe à chaque pixel d'une image**.

Type de tâche	But
Classification	Une classe pour toute l'image
Détection	Boîte + classe
Segmentation	Classe pour chaque pixel

→ Exemple : isoler un organe, une route, des arbres…

- ### Segmentation binaire

Chaque pixel est :

- 1 : appartient à l'objet
- 0 : n'appartient pas à l'objet

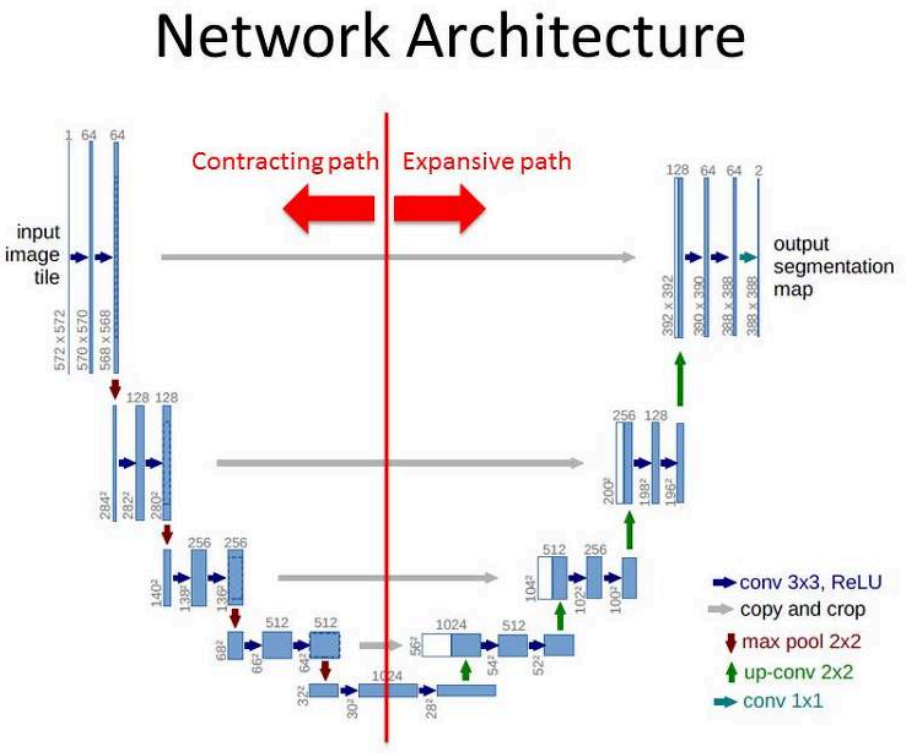
- ### Masque (mask)

Image indiquant la classe de chaque pixel.

- ### Architecture U-Net

Un réseau CNN encoder–decoder conçu pour la segmentation d'images biomédicales.
Introduit en 2015 par Ronneberger et al.

2) U-Net : Architecture (Vue d'ensemble)



U-Net est composé de trois parties :

1) Encoder (contracting path)

- Suite de convolutions + max-pooling
- Extrait les caractéristiques
- Réduit la dimension spatiale

2) Bottleneck

- Niveau le plus comprimé

3) Decoder (expanding path)

- Upsampling

- Reconstruction de l'image segmentée

Les **skip connections** relient encoder → decoder
→ Cela permet de récupérer les détails perdus.

Schéma :

```
Input → Encoder → Bottleneck → Decoder → Output Mask
```

3) Prérequis techniques

Environnement

- Python 3.8+
- TensorFlow ou PyTorch
- GPU recommandé

Installer :

```
pip install tensorflow opencv-python matplotlib
```

4) Préparation du dataset

Organisation :

```
dataset/
  ├── images/
  └── masks/
```

- Une image a un masque correspondant
- Les masques sont binaires (0 = background, 1 = objet)

Exemples de datasets :

- ISBI
- DRIVE
- Kaggle datasets (cellules, poumons, routes…)

5) Code — U-Net (TensorFlow)

Importations

```
import tensorflow as tf
from tensorflow.keras import layers, models
```

Construction U-Net

```
from tensorflow.keras import layers, models

def unet(input_shape=(256,256,3), n_classes=1):
    inputs = layers.Input(input_shape)

    # Encoder
    c1 = layers.Conv2D(64, 3, activation='relu', padding='same')(inputs)
    c1 = layers.Conv2D(64, 3, activation='relu', padding='same')(c1)
    p1 = layers.MaxPooling2D((2, 2))(c1)

    c2 = layers.Conv2D(128, 3, activation='relu', padding='same')(p1)
    c2 = layers.Conv2D(128, 3, activation='relu', padding='same')(c2)
    p2 = layers.MaxPooling2D((2, 2))(c2)

    c3 = layers.Conv2D(256, 3, activation='relu', padding='same')(p2)
    c3 = layers.Conv2D(256, 3, activation='relu', padding='same')(c3)
    p3 = layers.MaxPooling2D((2, 2))(c3)

    c4 = layers.Conv2D(512, 3, activation='relu', padding='same')(p3)
    c4 = layers.Conv2D(512, 3, activation='relu', padding='same')(c4)
    p4 = layers.MaxPooling2D((2, 2))(c4)
```

```
# Bottleneck
c5 = layers.Conv2D(1024, 3, activation='relu', padding='same')(p4)
c5 = layers.Conv2D(1024, 3, activation='relu', padding='same')(c5)

# Decoder
u6 = layers.Conv2DTranspose(512, 2, strides=2, padding='same')(c5)
u6 = layers.concatenate([u6, c4])
c6 = layers.Conv2D(512, 3, activation='relu', padding='same')(u6)
c6 = layers.Conv2D(512, 3, activation='relu', padding='same')(c6)

u7 = layers.Conv2DTranspose(256, 2, strides=2, padding='same')(c6)
u7 = layers.concatenate([u7, c3])
c7 = layers.Conv2D(256, 3, activation='relu', padding='same')(u7)
c7 = layers.Conv2D(256, 3, activation='relu', padding='same')(c7)

u8 = layers.Conv2DTranspose(128, 2, strides=2, padding='same')(c7)
u8 = layers.concatenate([u8, c2])
c8 = layers.Conv2D(128, 3, activation='relu', padding='same')(u8)
c8 = layers.Conv2D(128, 3, activation='relu', padding='same')(c8)

u9 = layers.Conv2DTranspose(64, 2, strides=2, padding='same')(c8)
u9 = layers.concatenate([u9, c1])
c9 = layers.Conv2D(64, 3, activation='relu', padding='same')(u9)
c9 = layers.Conv2D(64, 3, activation='relu', padding='same')(c9)

    # Output
    if n_classes == 1:
        outputs = layers.Conv2D(1, 1, activation='sigmoid')(c9)
    else:
        outputs = layers.Conv2D(n_classes, 1, activation='softmax')(c9)

    return models.Model(inputs, outputs)

# Exemple minimal
model = unet(input_shape=(128,128,3), n_classes=1)
model.summary()
```

6) Compilation du modèle

```
model.compile(
    optimizer="adam",
    loss="binary_crossentropy",
    metrics=["accuracy"]
)
```

7) Chargement des données

À adapter selon structure…

```
import numpy as np
import cv2
import glob

def load_data(path_img, path_mask):
    imgs = []
    masks = []

    for img_path in glob.glob(path_img + "/*.png"):
        img = cv2.imread(img_path)
        img = cv2.resize(img, (128,128)) / 255.0
        imgs.append(img)

        mname = img_path.replace("images", "masks")
        mask = cv2.imread(mname, 0)
        mask = cv2.resize(mask, (128,128)) / 255.0
        masks.append(mask)

    return np.array(imgs), np.array(masks)

X, y = load_data("dataset/images", "dataset/masks")
y = y.reshape(-1, 128, 128, 1)
```

8) Entraînement

```
history = model.fit(
    X, y,
    batch_size=8,
    epochs=20,
    validation_split=0.1
)
```

9) Visualisation des prédictions

```
pred = model.predict(X[:5])

import matplotlib.pyplot as plt

for i in range(5):
    plt.figure(figsize=(10,4))
    plt.subplot(1,3,1)
    plt.title("Image")
    plt.imshow(X[i])

    plt.subplot(1,3,2)
    plt.title("Mask")
    plt.imshow(y[i].squeeze(), cmap="gray")

    plt.subplot(1,3,3)
    plt.title("Prediction")
    plt.imshow(pred[i].squeeze(), cmap="gray")
    plt.show()
```

10) Métriques utiles

Dice coefficient

$$Dice = \frac{2|A \cap B|}{|A| + |B|}$$

IoU

Rapport intersection / union.

11) Améliorations possibles

- Data augmentation
- U-Net++
- Attention U-Net
- ResUNet
- Multi-classes
- Utilisation de weights pré-entraînés
- Post-processing (CRF, etc.)

12) Exercice proposé

Objectif

- Entraîner un U-Net sur un dataset simple (ex. cellules)
- Visualiser masque + prédiction
- Calculer IoU ou Dice Score

Bonus

- Data augmentation
- Multi-classes
- Comparaison avec un modèle pré-entraîné