NATIONAL
YANG MING CHIAO TUNG
UNIVERSITY

RECONNAISSANCE IN NATURAL ENVIRONMENTS II

PROJECT

Image Data Generation

CLEMENCEAU Maxime
PAULY Alexandre

Team N°3

PROJECT PRESENTATION

This presentation describes the work conducted as part of the project carried out at NYCU in Hsinchu, Taiwan, during the spring semester of 2024, initiated by the AI CUP. The objective of this competition is to use generative AI techniques to create precise delineation lines of waterways and roads from images captured by drones.

DATASET

- Images acquired by drone
- Road and river images
- Training data : 4320 (images and labels) - 2160 of each
- Test data : 720 (images) - 360 of each
- Images dimension : 428 x 240
- Different qualities and points of view

428 pixels

240 pixels

LEARNING METHODS

V-Net :

- Input : Images 428 x 240
- Downsampling : Convolution 2D with 16, 32 and 64 filters using ReLu + BatchNormalization + MaxPooling2D
- Bottleneck : Convolution 2D with 128 filters using ReLu + BatchNormalization + MaxPooling2D
- Upsampling : Convolution 2D with 64, 32 and 16 filters using ReLu + BatchNormalization + MaxPooling2D
- Output : Convolution 2D with 1 filter using Sigmoid

U-Net :

- Input : Images 428 x 240
- Downsampling : Convolution 2D with 64 and 128 filters using ReLu + MaxPooling 2D
- Bottleneck : Convolution 2D with 256 filters using ReLu
- Upsampling : Transposed convolution 2D with 128 and 64 filters using ReLu
- Output : Convolution 2D with 1 filter using Sigmoid

U-Net deeper :

- The same than U-Net, but with more layers (from 64 to 1024 filters)

POST-TREATMENT METHODS

- Opening and closing by erosion and dilatation
- Extract the largest connected component
- Refine edges using the Laplacian filter
- HoughLinesP from open-cv
- Adding a middle line

Fig : Opening and closing

Fig : HoughLinesP

Fig : Laplacian filter

POST-TREATMENT METHODS

Original Image

Post Model Image

Post Treatment Image

EVALUATION

$$Recall = \frac{TP}{TP + FN}$$

$$Precision = \frac{TP}{TP + FP}$$

$$F1 = \frac{2 \times Precision \times Recall}{Precision + Recall}$$

RESULT AND ANALYSIS

	U-Net 15 epochs / 5 batch size	V-Net 10 epochs / 5 batch size
Precision	**0.367**	0.143
Recall	0.533	**0.638**
F-Score	**0.435**	0.233

Fig : Results for both Model after Post Treatment

RESULT AND ANALYSIS

	Raw Predictions without Post-Treatment	Opening / Closing and Largest Component	Adding Laplacian Filter
Precision	0.218	0.277	**0.367**
Recall	**0.753**	0.635	0.533
F-Score	0.339	0.386	**0.435**

Fig : Results without different Post Treatment on U-Net

RESULT AND ANALYSIS

	U-Net without middle line	U-Net with middle line
Precision	**0.367**	0.331
Recall	0.533	**0.559**
F-Score	**0.435**	0.416

Fig : Results for U-Net Model with / without the Middle Line

IMPROVEMENTS

- Experiment with different neural network architectures (like ResNet-based U-Nets, SegNet, etc.)
- Try different loss function and optimizer
- Implementing more sophisticated data augmentation strategies, such as random rotations, flips, or elastic deformations
- Advanced preprocessing methods like contrast enhancement or adaptive thresholding
- Adding a middle line

DISCLAIMER

Our efforts are currently constrained by the performance limitations of our computational resources.

Thanks for your attention !

REFERENCES

1. IMAGE-TO-IMAGE TRANSLATION WITH CONDITIONAL ADVERSARIAL NETWORKS. (S. D.). IEEE XPLORE. HTTPS://IEEEXPLORE.IEEE.ORG/DOCUMENT/8100115

2. PAPERS WITH CODE - ATTENTION U-NET: LEARNING WHERE TOLOOK FOR THE PANCREAS. (S. D.). THE LATEST IN MACHINE LEARNING | PAPERS WITH CODE, HTTPS://PAPERSWITHCODE.COM/PAPER/ATTENTION-U-NET-LEARNING-WHERE-TO-LOOK-FOR

3. DEEPUNET: A DEEP FULLY CONVOLUTIONAL NETWORK FOR PIXEL-LEVEL SEA-LAND SEGMENTATION. (S. D.). IEEE XPLORE, HTTPS://IEEEXPLORE.IEEE.ORG/DOCUMENT/8370071

4. PYTHON OPENCV - CANNY() FUNCTION - GEEKSFORGEEKS. (S. D.). GEEKSFORGEEKS, HTTPS://WWW.GEEKSFORGEEKS.ORG/PYTHON-OPENCV-CANNY-FUNCTION/

5. OPENCV: HOUGH LINE TRANSFORM. (S. D.). OPENCV DOCUMENTATION INDEX, HTTPS://DOCS.OPENCV.ORG/3.4/D9/DB0/TUTORIAL_HOUGH_LINES.HTML

6. 3.4. METRICS AND SCORING: QUANTIFYING THE QUALITY OF PREDICTIONS. (S. D.). SCIKIT-LEARN, HTTPS://SCIKIT-LEARN.ORG/STABLE/MODULES/MODEL_EVALUATION.HTML

U-Net and U-Net deeper

```python
# U-Net model
def UNet(input_shape):
    inputs = layers.Input(shape=input_shape)

    # Downsample
    conv1 = layers.Conv2D(64, (3, 3), activation='relu', padding='same')(inputs)
    conv1 = layers.Conv2D(64, (3, 3), activation='relu', padding='same')(conv1)
    pool1 = layers.MaxPooling2D((2, 2))(conv1)

    conv2 = layers.Conv2D(128, (3, 3), activation='relu', padding='same')(pool1)
    conv2 = layers.Conv2D(128, (3, 3), activation='relu', padding='same')(conv2)
    pool2 = layers.MaxPooling2D((2, 2))(conv2)

    # Bottleneck
    conv3 = layers.Conv2D(256, (3, 3), activation='relu', padding='same')(pool2)
    conv3 = layers.Conv2D(256, (3, 3), activation='relu', padding='same')(conv3)

    # Upsample
    u1 = layers.Conv2DTranspose(128, (2, 2), strides=(2, 2), padding='same')(conv3)
    u1 = layers.concatenate([u1, conv2])
    conv4 = layers.Conv2D(128, (3, 3), activation='relu', padding='same')(u1)
    conv4 = layers.Conv2D(128, (3, 3), activation='relu', padding='same')(conv4)

    u2 = layers.Conv2DTranspose(64, (2, 2), strides=(2, 2), padding='same')(conv4)
    u2 = layers.concatenate([u2, conv1])
    conv5 = layers.Conv2D(64, (3, 3), activation='relu', padding='same')(u2)
    conv5 = layers.Conv2D(64, (3, 3), activation='relu', padding='same')(conv5)

    outputs = layers.Conv2D(1, (1, 1), activation='sigmoid')(conv5)

    model = models.Model(inputs=[inputs], outputs=[outputs])

    return model
```

```python
def UNet_deeper(input_shape):
    inputs = layers.Input(shape=input_shape)

    # Downsample
    conv1 = layers.Conv2D(64, (3, 3), activation='relu', padding='same')(inputs)
    conv1 = layers.Conv2D(64, (3, 3), activation='relu', padding='same')(conv1)
    pool1 = layers.MaxPooling2D((2, 2))(conv1)

    conv2 = layers.Conv2D(128, (3, 3), activation='relu', padding='same')(pool1)
    conv2 = layers.Conv2D(128, (3, 3), activation='relu', padding='same')(conv2)
    pool2 = layers.MaxPooling2D((2, 2))(conv2)

    conv3 = layers.Conv2D(256, (3, 3), activation='relu', padding='same')(pool2)
    conv3 = layers.Conv2D(256, (3, 3), activation='relu', padding='same')(conv3)
    pool3 = layers.MaxPooling2D((2, 2))(conv3)

    conv4 = layers.Conv2D(512, (3, 3), activation='relu', padding='same')(pool3)
    conv4 = layers.Conv2D(512, (3, 3), activation='relu', padding='same')(conv4)
    pool4 = layers.MaxPooling2D((2, 2))(conv4)

    # Bottleneck
    conv5 = layers.Conv2D(1024, (3, 3), activation='relu', padding='same')(pool4)
    conv5 = layers.Conv2D(1024, (3, 3), activation='relu', padding='same')(conv5)

    # Upsample
    u1 = layers.Conv2DTranspose(512, (2, 2), strides=(2, 2), padding='same')(conv5)
    u1 = layers.concatenate([u1, conv4])
    conv6 = layers.Conv2D(512, (3, 3), activation='relu', padding='same')(u1)
    conv6 = layers.Conv2D(512, (3, 3), activation='relu', padding='same')(conv6)

    u2 = layers.Conv2DTranspose(256, (2, 2), strides=(2, 2), padding='same')(conv6)
    u2 = layers.concatenate([u2, conv3])
    conv7 = layers.Conv2D(256, (3, 3), activation='relu', padding='same')(u2)
    conv7 = layers.Conv2D(256, (3, 3), activation='relu', padding='same')(conv7)

    u3 = layers.Conv2DTranspose(128, (2, 2), strides=(2, 2), padding='same')(conv7)
    u3 = layers.concatenate([u3, conv2])
    conv8 = layers.Conv2D(128, (3, 3), activation='relu', padding='same')(u3)
    conv8 = layers.Conv2D(128, (3, 3), activation='relu', padding='same')(conv8)

    u4 = layers.Conv2DTranspose(64, (2, 2), strides=(2, 2), padding='same')(conv8)
    u4 = layers.concatenate([u4, conv1])
    conv9 = layers.Conv2D(64, (3, 3), activation='relu', padding='same')(u4)
    conv9 = layers.Conv2D(64, (3, 3), activation='relu', padding='same')(conv9)

    outputs = layers.Conv2D(1, (1, 1), activation='sigmoid')(conv9)

    model = models.Model(inputs=[inputs], outputs=[outputs])

    return model
```

V-Net

```python
from tensorflow.keras.layers import Conv2D, BatchNormalization, Activation, ZeroPadding2D

def conv_block(inputs, num_filters, kernel_size=(3, 3), activation='relu', padding='same'):
    x = Conv2D(num_filters, kernel_size, padding=padding)(inputs)
    x = BatchNormalization()(x)
    x = Activation(activation)(x)
    x = Conv2D(num_filters, kernel_size, padding=padding)(x)
    x = BatchNormalization()(x)
    x = Activation(activation)(x)
    return x

def VNet(input_shape):
    inputs = layers.Input(shape=input_shape)

    # Encoder
    c1 = conv_block(inputs, 16)
    p1 = layers.MaxPooling2D((2, 2))(c1)
    c2 = conv_block(p1, 32)
    p2 = layers.MaxPooling2D((2, 2))(c2)
    c3 = conv_block(p2, 64)
    p3 = layers.MaxPooling2D((2, 2))(c3)
    c4 = conv_block(p3, 128)

    # Decoder
    u1 = layers.Conv2DTranspose(64, (2, 2), strides=(2, 2), padding='same')(c4)
    # Adjust dimensions if necessary
    if c3.shape[1] != u1.shape[1] or c3.shape[2] != u1.shape[2]:
        u1 = ZeroPadding2D(((0, c3.shape[1] - u1.shape[1]), (0, c3.shape[2] - u1.shape[2])))(u1)
    u1 = layers.concatenate([u1, c3])
    c5 = conv_block(u1, 64)

    u2 = layers.Conv2DTranspose(32, (2, 2), strides=(2, 2), padding='same')(c5)
    # Adjust dimensions if necessary
    if c2.shape[1] != u2.shape[1] or c2.shape[2] != u2.shape[2]:
        u2 = ZeroPadding2D(((0, c2.shape[1] - u2.shape[1]), (0, c2.shape[2] - u2.shape[2])))(u2)
    u2 = layers.concatenate([u2, c2])
    c6 = conv_block(u2, 32)

    u3 = layers.Conv2DTranspose(16, (2, 2), strides=(2, 2), padding='same')(c6)
    # Adjust dimensions if necessary
    if c1.shape[1] != u3.shape[1] or c1.shape[2] != u3.shape[2]:
        u3 = ZeroPadding2D(((0, c1.shape[1] - u3.shape[1]), (0, c1.shape[2] - u3.shape[2])))(u3)
    u3 = layers.concatenate([u3, c1])
    c7 = conv_block(u3, 16)

    outputs = layers.Conv2D(1, (1, 1), activation='sigmoid')(c7)

    model = models.Model(inputs=[inputs], outputs=[outputs])

    return model
```