Лабораторна робота №4

РОЗРОБКА ТА ДОСЛІДЖЕННЯ ЗАСОБІВ СЕМАНТИЧНОЇ СЕГМЕНТАЦІЇ ЗОБРАЖЕНЬ

Мета роботи: оволодіти практичними навичками розробки, навчання та оцінювання нейронних мереж для задачі семантичної сегментації растрових зображень.

ТЕОРЕТИЧНІ ВІДОМОСТІ

Семантична сегментація - це один із ключових напрямів комп’ютерного зору, який передбачає автоматичне розпізнавання змісту зображення на рівні пікселів. На відміну від задачі класифікації, де кожному зображенню відповідає один клас, або детекції об’єктів, де виділяються окремі області з об'єктами, семантична сегментація призначена для класифікації кожного пікселя зображення відповідно до його належності до певного класу (фон, людина, автомобіль, будівля тощо).

Результатом реалізації процедури семантичної сегментації є маска зображення, яка відповідає вхідному зображенню за розміром і в якій кожному пікселю призначено індекс класу. Семантична сегментація вимагає точної локалізації об'єктів та їх класифікації, що потребує збереження просторової структури вхідного зображення.

Класичні підходи не давали достатньо точних результатів через варіативність форми, розміру, кольору об'єктів і складність сцени. Сучасні підходи базуються на глибоких згорткових нейронних мережах (CNN), які здатні автоматично виділяти інформативні ознаки зображень і вивчати залежності між пікселями.

Однією з найефективніших архітектур для семантичної сегментації стала U-Net, що поєднує глибину ознак із точністю локалізації. U-Net була спочатку запропонована для біомедичної сегментації, але згодом продемонструвала ефективність і в інших галузях, таких як автономне водіння, аналіз супутникових зображень, агроаналітика тощо.

U-Net - це симетрична згорткова нейронна мережа, що складається з двох частин, енкодеру та декодера.

Енкодер (encoder) використовується для вилучення високорівневих ознак. Складається з послідовних згорткових шарів і операцій пониження

розмірності (пулінг).

Декодер (decoder) здійснює поступове відновлення просторової роздільної здатності через транспоновані згортки (up-sampling).

Ключова особливість U-Net - наявність "skip-зв'язків" між симетричними шарами encoder і decoder, які дозволяють передавати детальні просторові ознаки, втрачені під час згорток. Завдяки цьому мережа може точно локалізовувати межі об’єктів на зображенні.

Структура мережі U-Net показана на рис.1.

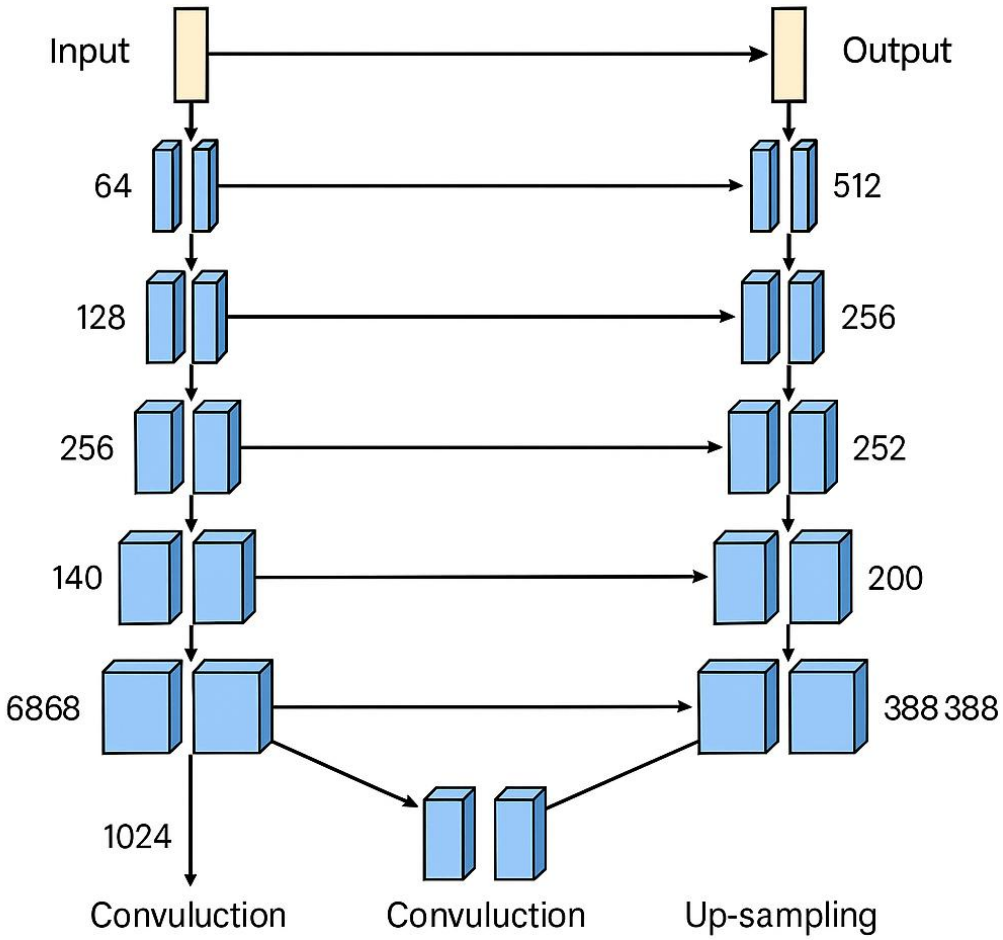

Рис. 1 Структура мережі

Оскільки сегментація - це задача багатокласової класифікації пікселів, то в U-Net використовується крос-ентропійна функція втрат:

$$\mathcal{L}_{\mathrm{C}} = -\sum_{i=1}^{N}\sum_{c=1}^{C} y_{i,c}\log\left(\widehat{y_{i,c}}\right)$$

N – кількість пікселів у зображенні,

C – кількість класів,

$y_{i,c}$ – реальна (one-hot) мітка класу для пікселя

$\widehat{y_{i,c}}$ – ймовірність приналежності пікселя iii до класу ccc, передбачена мережею.

Для задач з незбалансованими класами часто додаються вагові коефіцієнти або застосовуються альтернативні функції втрат (наприклад, Dice Loss, Focal Loss).

Для тренування моделей семантичної сегментації використовуються спеціалізовані датасети, що містять зображення разом з розміченими піксельними масками. Серед найпопулярніших:

- Pascal VOC - http://host.robots.ox.ac.uk/pascal/VOC/
- Cityscapes - https://www.cityscapes-dataset.com/
- COCO (Common Objects in Context) - https://cocodataset.org/

Короткий опис вказаних датасетів наведено в табл. 1

Таблиця 1

Характеристики датасетів

Характеристика	**Pascal VOC**	**Cityscapes**	**COCO**
Тематика	Повсякденні об'єкти	Міське середовище, дорожній рух	Складні сцени з багатьма об'єктами
Кількість класів	20 об’єктів + фон	~30 класів (основні: 19)	80+ класів
Тип зображень	Звичайні фото, різні сцени	Вулиці міст, зняті з автомобіля	Побутові та зовнішні сцени з багатьма об'єктами
Якість розмітки	Середня, растрові маски	Висока, піксельна точність	Змішана: полігони та маски
Розмір датасету	~10 тис. зображень	~5 тис. високоякісних зображень	>100 тис. зображень
Формат розмітки	PNG-маски	PNG-маски, JSON	JSON (COCO format) + PNG

Характеристика	Pascal VOC	Cityscapes	COCO
Переваги	Простий, легкий у використанні	Висока якість, реалістичні сцени	Найбільший і найрізноманітніший датасет
Недоліки	Обмежена кількість класів	Вузька тематика (місто)	Складна структура, великі обсяги

В багатьох випадках дані потребують попередньої обробки: зміни розміру, нормалізації, аугментації (обертання, дзеркалення, зсув), що підвищує стійкість моделі до варіацій вхідного сигналу.

В додатку 1 наведено приклад скрипта для завантаження та попередньої обробки Pascal VOC 2012 з використанням PyTorch та бібліотеки torchvision. Він охоплює: завантаження датасету, перетворення зображень та масок до тензорів, ресайзинг, аугментацію (опційно), створення DataLoader. Зазначимо, вимоги для скрипта: torch, torchvision, numpy, Pillow. За необхідності встановлення: pip install torch torchvision numpy.

Програмний код для реалізації U-Net, адаптованої до параметрів зображень із датасету Pascal VOC 2012, написаний на мові програмування Python наведено в додатку 2.

Хід виконання роботи

1. Ознайомлення з теоретичними відомості
2. Підготовка даних
2.1. Вибір навчального датасету (Pascal VOC, Cityscapes, COCO)
2.2. Попередня обробка зображень (за необхідності)
3. Визначення архітектурних параметрів та розробка програмного забезпечення для реалізації нейромережевої моделі типу U-Net.
4. Проведення навчання нейромережевої моделі.
5. Оцінка та візуалізація результатів навчання.
6. Проведення експериментальних досліджень для верифікації отриманих рішень.
7. Оформлення звіту в якому слід відобразити: П.І.Б. студента, номер групи, назву та мету лабораторної роботи; детальний опис етапів виконання; презентацію отриманих результатів; висновки щодо ефективності моделей та рекомендації для подальших досліджень.

Питання для самоперевірки

1. Що таке семантична сегментація зображень і в чому її основна мета?
2. Які особливості має архітектура U-Net? Чим вона відрізняється від інших CNN?
3. Для чого потрібен шар MaxPooling у згорткових нейронних мережах?
4. Яке призначення ConvTranspose2d у мережі U-Net?
5. Як реалізовано з’єднання (skip connections) у цій спрощеній моделі?
6. Яка функція втрат використовується у вашій реалізації та чому?
7. Які вхідні та вихідні дані обробляє модель U-Net у цьому скрипті?
8. Які переваги використання синтетичних даних для тестування моделі?
9. Як виглядає процес тренування моделі у вашій реалізації (основні етапи)?
10. Які метрики якості можна використати для оцінки результатів сегментації, окрім втрат?
11. Як можна адаптувати дану модель для роботи з реальними датасетами, такими як Pascal VOC?
12. Які обмеження має ця спрощена реалізація порівняно з повноцінним U-Net?

Список літератури

1. Tereikovskyi I., Hu Z., Chernyshev D., Tereikovska L., Korystin O., Tereikovskyi O. The Method of Semantic Image Segmentation Using Neural Networks. International Journal of Image, Graphics and Signal Processing (IJIGSP), Vol. 14, No. 6, pp. 1–14, 2022. DOI: 10.5815/ijigsp.2022.06.01

2. Bagitova K., Tereikovskyi I., Babayev I., Tereikovska L., Tereikovskyi O. Model for Processing Images of Online Social Networks Used to Recognize Political Extremism. Journal of Mathematics, Mechanics and Computer Science, Vol. 119, No. 3, pp. 91–103, 2023. DOI: 10.26577/JMMCS2023v119i3a8

3. Ronneberger O., Fischer P., Brox T. U-Net: Convolutional Networks for Biomedical Image Segmentation. In: Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015, Springer, LNCS 9351, pp. 234–241. DOI: 10.1007/978-3-319-24574-4_28

4. Goodfellow I., Bengio Y., Courville A. Deep Learning. MIT Press, 2016. http://www.deeplearningbook.org

Додаток 1

Приклад скрипта для завантаження та попередньої обробки Pascal VOC 2012

```
import torch
from torchvision import transforms
from torchvision.datasets import VOCSegmentation
from torch.utils.data import DataLoader
from torchvision.transforms import functional as F
import random

# Кастомне перетворення для зображень і масок
class VOCTransform:
    def __init__(self, resize=(256, 256), augment=False):
        self.resize = resize
        self.augment = augment

    def __call__(self, image, mask):
        # Зміна розміру
        image = F.resize(image, self.resize)
        mask = F.resize(mask, self.resize, interpolation=transforms.InterpolationMode.NEAREST)

        # Аугментація (дзеркальне відображення)
        if self.augment and random.random() > 0.5:
            image = F.hflip(image)
            mask = F.hflip(mask)

        # Перетворення в тензори
        image = F.to_tensor(image)
        mask = torch.as_tensor(np.array(mask), dtype=torch.long)

        return image, mask

# Завантаження датасету
def get_voc_dataloader(root_dir="./data", batch_size=8, train=True, augment=False):
    transform = VOCTransform(resize=(256, 256), augment=augment)
```

```
    dataset = VOCSegmentation(
        root=root_dir,
        year="2012",
        image_set="train" if train else "val",
        download=True,
        transforms=transform
    )

    dataloader = DataLoader(dataset, batch_size=batch_size, shuffle=train)
    return dataloader

# Використання
train_loader = get_voc_dataloader(train=True, augment=True)
val_loader = get_voc_dataloader(train=False)
```

Додаток 2

Програмний код для реалізації U-Net

```
# Реалізація U-Net та навчання на Pascal VOC 2012
try:
    import torch
    import torch.nn as nn
    import torch.nn.functional as F
    from torchvision import transforms
    from torchvision.datasets import VOCSegmentation
    from torch.utils.data import DataLoader
    from torchvision.transforms import functional as TF
    import numpy as np
    import random
except ModuleNotFoundError as e:
    print("\n[Помилка] Не встановлено необхідний модуль:", e.name)
    print("Будь ласка, встановіть залежності за допомогою команди: pip install
torch torchvision numpy Pillow")
    exit(1)
```

```
# ---------- 1. Трансформації для зображень і масок ----------
class VOCTransform:
    def __init__(self, resize=(256, 256), augment=False):
        self.resize = resize
        self.augment = augment

    def __call__(self, image, mask):
        image = TF.resize(image, self.resize)
        mask = TF.resize(mask, self.resize, interpolation=transforms.InterpolationMode.NEAREST)

        if self.augment and random.random() > 0.5:
            image = TF.hflip(image)
            mask = TF.hflip(mask)

        image = TF.to_tensor(image)
        mask = torch.as_tensor(np.array(mask), dtype=torch.long)

        return image, mask

# ---------- 2. Завантаження датасету ----------
def get_voc_dataloader(root_dir="./data", batch_size=4, train=True, augment=False):
    transform = VOCTransform(resize=(256, 256), augment=augment)

    dataset = VOCSegmentation(
        root=root_dir,
        year="2012",
        image_set="train" if train else "val",
        download=True,
        transforms=transform
    )

    return DataLoader(dataset, batch_size=batch_size, shuffle=train)

# ---------- 3. Реалізація U-Net ----------
```

```
class UNet(nn.Module):
    def __init__(self, in_channels=3, out_channels=21):
        super(UNet, self).__init__()

        def conv_block(in_c, out_c):
            return nn.Sequential(
                nn.Conv2d(in_c, out_c, 3, padding=1),
                nn.ReLU(inplace=True),
                nn.Conv2d(out_c, out_c, 3, padding=1),
                nn.ReLU(inplace=True)
            )

        self.enc1 = conv_block(in_channels, 64)
        self.enc2 = conv_block(64, 128)
        self.enc3 = conv_block(128, 256)
        self.enc4 = conv_block(256, 512)

        self.pool = nn.MaxPool2d(2)

        self.bottleneck = conv_block(512, 1024)

        self.upconv4 = nn.ConvTranspose2d(1024, 512, 2, stride=2)
        self.dec4 = conv_block(1024, 512)
        self.upconv3 = nn.ConvTranspose2d(512, 256, 2, stride=2)
        self.dec3 = conv_block(512, 256)
        self.upconv2 = nn.ConvTranspose2d(256, 128, 2, stride=2)
        self.dec2 = conv_block(256, 128)
        self.upconv1 = nn.ConvTranspose2d(128, 64, 2, stride=2)
        self.dec1 = conv_block(128, 64)

        self.final = nn.Conv2d(64, out_channels, 1)

    def forward(self, x):
        e1 = self.enc1(x)
        e2 = self.enc2(self.pool(e1))
        e3 = self.enc3(self.pool(e2))
        e4 = self.enc4(self.pool(e3))
```

```
        b = self.bottleneck(self.pool(e4))

        d4 = self.upconv4(b)
        d4 = self.dec4(torch.cat([d4, e4], dim=1))
        d3 = self.upconv3(d4)
        d3 = self.dec3(torch.cat([d3, e3], dim=1))
        d2 = self.upconv2(d3)
        d2 = self.dec2(torch.cat([d2, e2], dim=1))
        d1 = self.upconv1(d2)
        d1 = self.dec1(torch.cat([d1, e1], dim=1))

        return self.final(d1)

# ---------- 4. Навчання ----------
def train_model():
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

    train_loader = get_voc_dataloader(train=True, augment=True)
    val_loader = get_voc_dataloader(train=False)

    model = UNet().to(device)
    optimizer = torch.optim.Adam(model.parameters(), lr=1e-4)
    criterion = nn.CrossEntropyLoss()

    num_epochs = 10
    for epoch in range(num_epochs):
        model.train()
        total_loss = 0
        for images, masks in train_loader:
            images, masks = images.to(device), masks.to(device)
            optimizer.zero_grad()
            outputs = model(images)
            loss = criterion(outputs, masks)
            loss.backward()
            optimizer.step()
            total_loss += loss.item()
```

```
        print(f"Epoch {epoch+1}/{num_epochs}, Loss: {total_loss/len(train_loader):.4f}")

    torch.save(model.state_dict(), "unet_voc2012.pth")

if __name__ == "__main__":
    train_model()
```