МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ "ЛЬВІВСЬКА ПОЛІТЕХНІКА"

НАВЧАННЯ З УЧИТЕЛЕМ У КОМП'ЮТЕРНОМУ ЗОРІ: ВІД КЛАСИФІКАЦІЇ ЗОБРАЖЕНЬ ДО СЕМАНТИЧНОЇ СЕГМЕНТАЦІЇ

МЕТОДИЧНІ ВКАЗІВКИ

до виконання лабораторної роботи № 1
з дисципліни «Штучний інтелект в ігрових застосунках»
для студентів бакалаврського рівня вищої освіти спеціальності 121
"Інженерія програмного забезпечення"

Львів -- 2026

Навчання з учителем у комп'ютерному зорі: від класифікації зображень до семантичної сегментації: методичні вказівки до виконання лабораторної роботи №1 з дисципліни "Штучний інтелект в ігрових застосунках" для студентів першого (бакалаврського) рівня вищої освіти спеціальності 121 "Інженерія програмного забезпечення" . Укл.: О.Є. Бауск. -- Львів: Видавництво Національного університету "Львівська політехніка", 2026. -- 18 с.

Укладач: Бауск О.Є., к.т.н., асистент кафедри ПЗ

Відповідальний за випуск: Федасюк Д.В., доктор техн. наук, професор

Рецензенти: Федасюк Д.В., доктор техн. наук, професор

Задорожний І.М., асистент кафедри ПЗ

Тема роботи: Навчання з учителем у комп'ютерному зорі — класифікація зображень, візуалізація рішень моделі за допомогою Grad-CAM, та семантична сегментація з використанням PyTorch і torchvision.

Мета роботи: Ознайомитись з основними задачами комп'ютерного зору в контексті навчання з учителем; навчитися готувати набори даних та застосовувати аугментацію; створити й натренувати модель класифікації зображень із використанням передавального навчання; дослідити, на які частини зображення звертає увагу модель, за допомогою техніки Grad-CAM; ознайомитися із задачею семантичної сегментації та використати попередньо натреновану модель для сегментації ігрових сцен; побудувати комбінований конвеєр, що поєднує класифікацію та сегментацію.

Теоретичні відомості

Навчання з учителем (Supervised Learning)

Навчання з учителем — це парадигма машинного навчання, в якій модель тренується на наборі даних, що містить пари «вхідні дані — правильна відповідь» (так звані мітки, або *labels*). Модель навчається знаходити відображення з простору вхідних даних у простір відповідей, мінімізуючи функцію втрат, яка вимірює різницю між передбаченням моделі та правильною відповіддю.

Формально, маючи набір даних ${(x_i, y_i)}*{i=1}^{N}$, де x_i — вхідне зображення, а y_i — відповідна мітка, ми шукаємо параметри θ моделі $f*\theta$, що мінімізують:

$$\mathcal{L}(\theta) = \frac{1}{N} \sum_{i=1}^{N} \ell(f_\theta(x_i), y_i)$$

де ℓ — функція втрат (наприклад, крос-ентропія для класифікації).

У контексті ігрових застосунків навчання з учителем використовується для:

- Розпізнавання об'єктів на ігрових сценах (вороги, предмети, перешкоди)
- Класифікації типів ігрових середовищ для адаптивного завантаження ресурсів
- Сегментації ігрових сцен для генерації мап прохідності
- Автоматичного тегування ігрових ресурсів у великих бібліотеках контенту

Задачі комп'ютерного зору

Комп'ютерний зір охоплює декілька ключових задач, кожна з яких працює з різним рівнем деталізації:

Класифікація зображень — присвоєння одного загального класу (мітки) цілому зображенню. Наприклад: «це зображення містить денну ігрову сцену» або «це зображення містить нічну ігрову сцену». Модель приймає зображення і повертає ймовірності для кожного з можливих класів.

Виявлення об'єктів (Object Detection) — знаходження кількох об'єктів на зображенні та визначення їх розташування за допомогою обмежувальних рамок (bounding boxes). Кожен об'єкт класифікується окремо. Наприклад: на одному кадрі ігрової сцени можна виявити три вороги та два бонусних предмети.

Семантична сегментація — класифікація кожного окремого пікселя зображення як належного до певного класу. Результатом є маска того самого розміру, що й вхідне зображення, де кожен піксель має

мітку класу. Наприклад: кожен піксель ігрового кадру класифікується як «небо», «земля», «будівля», «персонаж» або «об'єкт».

Сегментація екземплярів (Instance Segmentation) — поєднання семантичної сегментації та виявлення об'єктів: кожен піксель отримує мітку класу, а також ідентифікатор конкретного екземпляра об'єкта.

У даній лабораторній роботі ми зосередимося на двох задачах: **класифікації** та **семантичній сегментації**, і навчимося поєднувати їх у єдиний конвеєр обробки зображень.

Згорткові нейронні мережі (Convolutional Neural Networks)

Згорткові нейронні мережі (CNN) — це клас нейронних мереж, спеціально розроблених для обробки даних зі структурою сітки, таких як зображення. Ключові компоненти CNN:

Згортковий шар (Convolutional Layer) — застосовує набір навчених фільтрів (ядер) до вхідного зображення. Кожен фільтр виявляє певну ознаку (наприклад, краї, кути, текстури). Ранні шари виявляють прості ознаки, а глибші шари — складніші (частини об'єктів, цілі об'єкти).

Шар підвибірки (Pooling Layer) — зменшує просторові розміри карт ознак, зберігаючи найважливішу інформацію. Max pooling вибирає максимальне значення в кожному вікні, зменшуючи обчислювальне навантаження та забезпечуючи інваріантність до малих зсувів.

Повнозв'язний шар (Fully Connected Layer) — з'єднує кожний нейрон з усіма нейронами попереднього шару. Зазвичай використовується в кінці мережі для перетворення вивчених ознак у фінальне передбачення.

Архітектура ResNet

ResNet (Residual Network) — це архітектура CNN, запропонована Каймінгом Хе та співавторами у 2015 році. Головна інновація — **залишкові з'єднання (skip connections)**, які дозволяють градієнтам протікати безпосередньо через шари, минаючи одну або кілька проміжних операцій.

Формально, замість того щоб навчати відображення $H(x)$, блок ResNet навчає залишкове відображення $F(x) = H(x) - x$, а вихід обчислюється як $H(x) = F(x) + x$. Це значно полегшує оптимізацію глибоких мереж.

Існують різні варіанти ResNet:

- **ResNet-18** — 18 шарів, ~11.7M параметрів, швидка й компактна
- **ResNet-34** — 34 шари, ~21.8M параметрів
- **ResNet-50** — 50 шарів, ~25.6M параметрів, використовує bottleneck блоки
- **ResNet-101, ResNet-152** — глибші варіанти для складніших задач

Для даної лабораторної роботи ми використаємо **ResNet-18** як базову архітектуру — вона достатньо потужна для класифікації зображень, але при цьому компактна і швидко тренується навіть на обмежених обчислювальних ресурсах.

Передавальне навчання (Transfer Learning)

Передавальне навчання — це підхід, при якому модель, натренована на великому загальному наборі даних (наприклад, ImageNet з 1.2 мільйонами зображень і 1000 класами), використовується як відправна точка для нової задачі.

Процес передавального навчання:

1. **Завантаження попередньо натренованої моделі** — використовуємо ваги, навчені на ImageNet.
2. **Заморожування (freezing)** базових шарів — фіксуємо ваги ранніх згорткових шарів, які вже навчилися виявляти універсальні ознаки (краї, текстури, форми).
3. **Заміна класифікаційної голови** — останній повнозв'язний шар замінюється на новий, що відповідає кількості класів нашої задачі.
4. **Тонке налаштування (fine-tuning)** — тренуємо нову голову, а за потреби й деякі верхні шари базової мережі на нашому наборі даних.

Передавальне навчання надзвичайно корисне, коли у нас обмежений набір даних (сотні або тисячі зображень замість мільйонів), що є типовою ситуацією в ігровій розробці.

Аугментація даних (Data Augmentation)

Аугментація — це техніка збільшення ефективного обсягу навчального набору шляхом застосування випадкових трансформацій до зображень під час тренування:

- **Випадкове обрізання (RandomResizedCrop)** — вирізає випадкову частину зображення та масштабує до потрібного розміру
- **Горизонтальне відзеркалення (RandomHorizontalFlip)** — дзеркально відображає зображення з ймовірністю 50%
- **Зміна кольору (ColorJitter)** — випадково змінює яскравість, контраст, насиченість
- **Випадкове повертання (RandomRotation)** — повертає зображення на випадковий кут
- **Нормалізація (Normalize)** — приводить значення пікселів до стандартного діапазону

Аугментація допомагає моделі узагальнювати краще, зменшуючи перенавчання (overfitting) — ситуацію, коли модель «запам'ятовує» тренувальні дані замість того, щоб вивчати загальні закономірності.

Grad-CAM: візуалізація рішень моделі

Grad-CAM (Gradient-weighted Class Activation Mapping) — це техніка пояснення рішень згорткових нейронних мереж, запропонована Selvaraju та співавторами у 2017 році.

Принцип роботи:

1. Виконується прямий прохід (forward pass) — зображення проходить через мережу, отримуємо передбачення.
2. Обираємо цільовий клас і обчислюємо градієнти скору цього класу відносно карт ознак останнього згорткового шару.
3. Для кожної карти ознак (каналу) обчислюємо середнє значення градієнта — це вага важливості цього каналу.
4. Обчислюємо зважену суму карт ознак і застосовуємо ReLU (залишаємо лише позитивні значення).

5. Масштабуємо отриману теплову карту до розміру вхідного зображення.

Результат — теплова карта (heatmap), що показує, які ділянки зображення найбільше вплинули на передбачення конкретного класу.

Чому це важливо для навчального процесу:

Grad-CAM є «мостом» між класифікацією та сегментацією. Коли студент бачить, що теплова карта класифікатора виділяє розмиту область навколо об'єкта, природно виникає питання: «А чи можна отримати точну, піксельну маску замість розмитої плями?» — і саме це робить семантична сегментація.

Семантична сегментація

Семантична сегментація — це задача класифікації кожного пікселя зображення. На відміну від класифікації, яка дає одну мітку на все зображення, сегментація створює *маску* розміром $H \times W$, де кожен елемент містить ідентифікатор класу.

Архітектура DeepLabV3:

DeepLabV3 — це одна з найпопулярніших архітектур для семантичної сегментації. Її ключові особливості:

1. **Backbone (базова мережа)** — зазвичай ResNet-50 або ResNet-101, що витягує ієрархічні ознаки з зображення.
2. **Atrous (Dilated) Convolution** — згортки з «розширеними» ядрами, що дозволяють збільшити рецептивне поле без збільшення кількості параметрів та без зменшення розмірності карт ознак.
3. **ASPP (Atrous Spatial Pyramid Pooling)** — модуль, що паралельно застосовує atrous convolution з різними коефіцієнтами розширення, захоплюючи контекст на різних масштабах.
4. **Декодер** — відновлює просторову роздільність і виробляє фінальну маску сегментації.

У бібліотеці torchvision доступні попередньо натреновані моделі DeepLabV3, навчені на наборі даних COCO (Common Objects in Context), який містить 21 клас: фон, аероплан, велосипед, птах, човен, пляшка, автобус, автомобіль, кіт, стілець, корова, обідній стіл, собака, кінь, мотоцикл, людина, горщична рослина, вівця, диван, потяг, монітор.

Відмінності архітектур для класифікації та сегментації

Характеристика	Класифікація	Семантична сегментація
Вхід	Зображення (H×W×3)	Зображення (H×W×3)
Вихід	Вектор ймовірностей (C)	Маска (H×W), кожен піксель — клас
Втрата просторової інформації	Так (pooling + FC шари)	Ні (декодер відновлює)
Типові архітектури	ResNet, EfficientNet, ViT	DeepLabV3, U-Net, SegFormer
Кількість параметрів (типово)	11–25M	40–60M
Обчислювальна вартість	Помірна	Висока

Обидва типи моделей використовують **однаковий backbone** (наприклад, ResNet), але відрізняються тим, що відбувається після нього: класифікатор стискає інформацію у вектор, а сегментатор відновлює просторову розмірність.

Метрики оцінки якості

Для класифікації:

- **Accuracy (точність)** — частка правильно класифікованих зображень
- **Precision** — частка справді позитивних серед усіх передбачених позитивних
- **Recall** — частка знайдених позитивних серед усіх справжніх позитивних
- **F1-score** — гармонійне середнє precision та recall

Для сегментації:

- **Pixel Accuracy** — частка правильно класифікованих пікселів
- **Mean IoU (Intersection over Union)** — середнє відношення площі перетину передбаченої маски та правильної маски до площі їх об'єднання. Це основна метрика для оцінки якості сегментації.

Інструменти та середовище розробки

PyTorch — гнучка бібліотека глибокого навчання, що надає низькорівневий контроль над процесом тренування. На відміну від високорівневих бібліотек (fastai, Keras), PyTorch вимагає від розробника явно описувати цикл тренування, що забезпечує глибше розуміння процесу.

torchvision — компаньйон-бібліотека до PyTorch, що містить:

- Попередньо натреновані моделі (ResNet, DeepLabV3, EfficientNet тощо)
- Утиліти для завантаження та трансформації зображень
- Стандартні набори даних

Google Colab — хмарне середовище, що надає безкоштовний доступ до GPU (NVIDIA T4). Ідеальне для тренування моделей без необхідності мати власне потужне апаратне забезпечення.

Kaggle Notebooks — альтернативне хмарне середовище з безкоштовним доступом до GPU (30 годин на тиждень на T4).

VS Code з розширенням Google Colab — офіційне розширення **Google Colab** для VS Code дозволяє підключатися до середовища виконання Colab безпосередньо з локального редактора. Це поєднує потужне середовище розробки VS Code (автодоповнення, навігація по коду, Git-інтеграція, розширення) з безкоштовним доступом до GPU/TPU від Google через Colab. Для роботи необхідно встановити розширення «Google Colab» з VS Code Marketplace (залежність — розширення Jupyter), відкрити файл `.ipynb`, обрати ядро «Colab» через меню вибору ядра (Select Kernel), автентифікуватись через Google-акаунт та обрати потрібне середовище виконання (включаючи GPU T4). Детальніше: https://developers.googleblog.com/google-colab-is-coming-to-vs-code/. Це рекомендований спосіб роботи для студентів, які бажають працювати у звичному локальному середовищі замість веб-інтерфейсу, зберігаючи при цьому доступ до хмарного GPU.

Висновок

У даній лабораторній роботі ми послідовно пройдемо шлях від класифікації зображень через візуалізацію рішень моделі (Grad-CAM) до семантичної сегментації. Цей шлях демонструє, як різні задачі комп'ютерного зору пов'язані між собою і як вони можуть бути об'єднані у конвеєр обробки, що є типовим підходом у сучасній ігровій розробці — від автоматичного тегування ігрових ресурсів до генерації мап прохідності з ігрових сцен.

Хід роботи

Етап 1. Налаштування середовища розробки

1.1. Налаштуйте VS Code з розширенням Google Colab для доступу до хмарного GPU.

Офіційне розширення **Google Colab** для VS Code дозволяє виконувати код на хмарному GPU (NVIDIA T4) від Google, працюючи при цьому у звичному локальному середовищі VS Code з усіма його перевагами: автодоповнення, навігація по коду, Git-інтеграція, розширення тощо.

Попередні вимоги:

- Встановлений VS Code (версія 1.98.0 або новіша): https://code.visualstudio.com/
- Google-акаунт (для автентифікації в Colab)

Налаштування:

1. Відкрийте VS Code та перейдіть до панелі розширень (Extensions): `Ctrl+Shift+X` (Windows/Linux) або `Cmd+Shift+X` (macOS).
2. Знайдіть та встановіть розширення **Jupyter** (видавець: Microsoft). Це розширення забезпечує підтримку файлів `.ipynb` у VS Code і є обов'язковою залежністю для роботи з ноутбуками.
3. Знайдіть та встановіть розширення **Google Colab** (видавець: Google). Якщо розширення Jupyter ще не встановлено, воно буде запропоноване автоматично.
4. Створіть новий файл ноутбука `.ipynb` у VS Code (File → New File → Jupyter Notebook) або відкрийте існуючий.
5. Натисніть **Select Kernel** у верхньому правому куті ноутбука (або запустіть будь-яку ячейку коду).
6. У списку ядер оберіть **Colab**.
7. Автентифікуйтесь через ваш Google-акаунт у вікні браузера, що відкриється.
8. Оберіть середовище виконання з GPU — **T4**.

Після цього весь код у вашому ноутбуці буде виконуватись на хмарному GPU від Google.

Детальніше про розширення: https://developers.googleblog.com/google-colab-is-coming-to-vs-code/

УВАГА! Безкоштовний рівень Google Colab надає обмежений час GPU. Не залишайте середовище виконання підключеним без потреби — відключайтеся після завершення обчислень, щоб зберегти ліміт.

1.2. Перевірте наявність GPU та встановіть необхідні бібліотеки.

Виконайте в першій ячейці ноутбука:

```
import torch
print(f"PyTorch version: {torch.__version__}")
print(f"CUDA available: {torch.cuda.is_available()}")
if torch.cuda.is_available():
    print(f"GPU: {torch.cuda.get_device_name(0)}")
```

Встановіть додаткові бібліотеки:

```
!pip install -q matplotlib numpy Pillow
```

Імпортуйте основні модулі:

```
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, Dataset, random_split
from torchvision import transforms, models, datasets
from torchvision.models import ResNet18_Weights
from PIL import Image
import matplotlib.pyplot as plt
import numpy as np
import os
from pathlib import Path
```

Етап 2. Підготовка набору даних

На цьому етапі ви завантажите відкритий набір даних для бінарної класифікації та підготуєте його для тренування. Оберіть один з двох наборів даних та пару класів відповідно до вашого **індивідуального варіанта** (див. розділ «Індивідуальні варіанти завдання»).

Доступні набори даних:

- **Oxford-IIIT Pet** — 37 порід котів та собак, ~200 зображень на породу, високе розрішення (~300px+). Класи «кіт» і «собака» присутні в COCO, тому семантична сегментація на етапі 5 даватиме змістовні результати.
- **Caltech-101** — 101 категорія об'єктів (тварини, транспорт, предмети тощо), ~40–800 зображень на категорію, високе розрішення (~300px+). Багато категорій збігаються з класами COCO.

2.1. Завантажте набір даних за допомогою torchvision.

Варіант А: Oxford-IIIT Pet

```
from torchvision.datasets import OxfordIIITPet

# Завантаження набору даних (автоматичне скачування ~800 МБ)
full_pet_dataset = OxfordIIITPet(root="data", split="trainval", download=True)

# Список усіх порід
all_classes = full_pet_dataset.classes
print(f"Кількість класів: {len(all_classes)}")
print(f"Класи: {all_classes}")
```

Варіант Б: Caltech-101

```
from torchvision.datasets import Caltech101

# Завантаження набору даних (автоматичне скачування ~130 МБ)
full_caltech_dataset = Caltech101(root="data", download=True)

# Список усіх категорій
all_classes = full_caltech_dataset.categories
print(f"Кількість класів: {len(all_classes)}")
print(f"Класи: {all_classes}")
```

2.2. Відфільтруйте набір даних для двох класів вашого варіанта.

```
# Замініть назви класів відповідно до вашого варіанта!
CLASS_A = "siamese"   # Наприклад для Oxford-IIIT Pet
CLASS_B = "persian"   # Наприклад для Oxford-IIIT Pet

# --- Для Oxford-IIIT Pet ---
from torchvision.datasets import OxfordIIITPet

raw_dataset = OxfordIIITPet(root="data", split="trainval", download=True)
class_names = raw_dataset.classes
```

```
idx_a = class_names.index(CLASS_A)
idx_b = class_names.index(CLASS_B)

# Фільтруємо зображення лише для двох обраних класів
filtered_indices = [
    i for i, (_, label) in enumerate(raw_dataset._images_and_targets())
    if label in (idx_a, idx_b)
]
# Альтернативний спосіб фільтрації:
filtered_images = []
filtered_labels = []
for i in range(len(raw_dataset)):
    img, label = raw_dataset[i]
    if label == idx_a:
        filtered_images.append((img, 0))
    elif label == idx_b:
        filtered_images.append((img, 1))

print(f"Відфільтровано: {len(filtered_images)} зображень")
print(f"  Клас 0 ({CLASS_A}): {sum(1 for _, l in filtered_images if l == 0)}")
print(f"  Клас 1 ({CLASS_B}): {sum(1 for _, l in filtered_images if l == 1)}")

# --- Для Caltech-101 ---
# from torchvision.datasets import Caltech101
# raw_dataset = Caltech101(root="data", download=True)
# Аналогічна фільтрація за raw_dataset.categories
```

2.3. Створіть клас Dataset-обгортку для відфільтрованих даних та визначте трансформації.

```
class BinaryClassDataset(Dataset):
    """Обгортка для бінарної класифікації з двох класів."""

    def __init__(self, image_label_pairs, transform=None):
        self.data = image_label_pairs  # список (PIL.Image, label)
        self.classes = [CLASS_A, CLASS_B]
        self.transform = transform

    def __len__(self):
        return len(self.data)

    def __getitem__(self, idx):
        image, label = self.data[idx]
        if not isinstance(image, Image.Image):
            image = image.convert("RGB")
        else:
            image = image.convert("RGB")
        if self.transform:
            image = self.transform(image)
        return image, label

# Нормалізація ImageNet
IMAGENET_MEAN = [0.485, 0.456, 0.406]
IMAGENET_STD = [0.229, 0.224, 0.225]

# Трансформації для тренувального набору (з аугментацією)
train_transform = transforms.Compose([
```

```
    transforms.RandomResizedCrop(224),
    transforms.RandomHorizontalFlip(),
    transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2),
    transforms.RandomRotation(15),
    transforms.ToTensor(),
    transforms.Normalize(mean=IMAGENET_MEAN, std=IMAGENET_STD)
])

# Трансформації для валідаційного набору (без аугментації)
val_transform = transforms.Compose([
    transforms.Resize(256),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize(mean=IMAGENET_MEAN, std=IMAGENET_STD)
])
```

2.4. Розділіть набір даних на тренувальну та валідаційну вибірки.

```
import random

# Перемішуємо для рівномірного розподілу
random.seed(42)
shuffled_data = filtered_images.copy()
random.shuffle(shuffled_data)

# Розділення: 80% тренування, 20% валідація
split_idx = int(0.8 * len(shuffled_data))
train_data = shuffled_data[:split_idx]
val_data = shuffled_data[split_idx:]

train_dataset = BinaryClassDataset(train_data, transform=train_transform)
val_dataset = BinaryClassDataset(val_data, transform=val_transform)

BATCH_SIZE = 32
train_loader = DataLoader(train_dataset, batch_size=BATCH_SIZE, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=BATCH_SIZE, shuffle=False)

print(f"Тренувальна вибірка: {len(train_dataset)} зображень")
print(f"Валідаційна вибірка: {len(val_dataset)} зображень")
```

2.5. Візуалізуйте декілька зображень з набору даних.

```
def denormalize(tensor, mean=IMAGENET_MEAN, std=IMAGENET_STD):
    """Скасовує нормалізацію для візуалізації."""
    img = tensor.numpy().transpose((1, 2, 0))
    img = np.array(std) * img + np.array(mean)
    return np.clip(img, 0, 1)

def show_batch(dataloader, classes, n=8):
    """Показує перші n зображень з батча."""
    images, labels = next(iter(dataloader))
    n = min(n, len(images))
    fig, axes = plt.subplots(1, n, figsize=(2.5 * n, 3))
    if n == 1:
```

```
        axes = [axes]
    for i in range(n):
        img = denormalize(images[i])
        axes[i].imshow(img)
        axes[i].set_title(classes[labels[i]], fontsize=10)
        axes[i].axis("off")
    plt.tight_layout()
    plt.show()

show_batch(train_loader, train_dataset.classes)
```

Етап 3. Класифікація зображень із передавальним навчанням

На цьому етапі ви створите модель класифікації зображень на базі попередньо натренованої ResNet-18.

3.1. Завантажте попередньо натреновану модель та адаптуйте її.

```
# Завантаження ResNet-18 з вагами, натренованими на ImageNet
model = models.resnet18(weights=ResNet18_Weights.IMAGENET1K_V1)

# Заморожуємо всі шари базової мережі
for param in model.parameters():
    param.requires_grad = False

# Замінюємо останній повнозв'язний шар
num_features = model.fc.in_features
model.fc = nn.Sequential(
    nn.Dropout(0.3),
    nn.Linear(num_features, 2)  # 2 класи для бінарної класифікації
)

# Переносимо модель на GPU
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = model.to(device)
print(f"Модель працює на: {device}")
```

3.2. Визначте функцію втрат, оптимізатор та функцію тренування.

```
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.fc.parameters(), lr=0.001)
scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=5, gamma=0.1)

def train_one_epoch(model, dataloader, criterion, optimizer, device):
    """Тренує модель протягом однієї епохи."""
    model.train()
    running_loss = 0.0
    correct = 0
    total = 0

    for images, labels in dataloader:
        images, labels = images.to(device), labels.to(device)

        optimizer.zero_grad()
        outputs = model(images)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        running_loss += loss.item() * images.size(0)
        _, predicted = outputs.max(1)
        total += labels.size(0)
        correct += predicted.eq(labels).sum().item()
```

```
    epoch_loss = running_loss / total
    epoch_acc = correct / total
    return epoch_loss, epoch_acc

def evaluate(model, dataloader, criterion, device):
    """Оцінює модель на валідаційному наборі."""
    model.eval()
    running_loss = 0.0
    correct = 0
    total = 0

    with torch.no_grad():
        for images, labels in dataloader:
            images, labels = images.to(device), labels.to(device)
            outputs = model(images)
            loss = criterion(outputs, labels)

            running_loss += loss.item() * images.size(0)
            _, predicted = outputs.max(1)
            total += labels.size(0)
            correct += predicted.eq(labels).sum().item()

    epoch_loss = running_loss / total
    epoch_acc = correct / total
    return epoch_loss, epoch_acc
```

3.3. Натренуйте модель.

```
NUM_EPOCHS = 10

train_losses, val_losses = [], []
train_accs, val_accs = [], []

for epoch in range(NUM_EPOCHS):
    train_loss, train_acc = train_one_epoch(
        model, train_loader, criterion, optimizer, device
    )
    val_loss, val_acc = evaluate(model, val_loader, criterion, device)
    scheduler.step()

    train_losses.append(train_loss)
    val_losses.append(val_loss)
    train_accs.append(train_acc)
    val_accs.append(val_acc)

    print(f"Епоха {epoch+1}/{NUM_EPOCHS} | "
          f"Train Loss: {train_loss:.4f} | Train Acc: {train_acc:.4f} | "
          f"Val Loss: {val_loss:.4f} | Val Acc: {val_acc:.4f}")
```

3.4. Побудуйте графіки навчання.

```
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(14, 5))

ax1.plot(train_losses, label="Train Loss")
```

```
ax1.plot(val_losses, label="Val Loss")
ax1.set_xlabel("Епоха")
ax1.set_ylabel("Втрати (Loss)")
ax1.set_title("Крива втрат")
ax1.legend()
ax1.grid(True)

ax2.plot(train_accs, label="Train Accuracy")
ax2.plot(val_accs, label="Val Accuracy")
ax2.set_xlabel("Епоха")
ax2.set_ylabel("Точність (Accuracy)")
ax2.set_title("Крива точності")
ax2.legend()
ax2.grid(True)

plt.tight_layout()
plt.show()
```

3.5. Протестуйте модель на кількох зображеннях.

```
def predict_image(model, image_path, transform, classes, device):
    """Класифікує одне зображення та виводить результат."""
    image = Image.open(image_path).convert("RGB")
    input_tensor = transform(image).unsqueeze(0).to(device)

    model.eval()
    with torch.no_grad():
        output = model(input_tensor)
        probabilities = torch.softmax(output, dim=1)
        predicted_class = probabilities.argmax(dim=1).item()
        confidence = probabilities[0, predicted_class].item()

    plt.imshow(image)
    plt.title(f"Передбачення: {classes[predicted_class]} ({confidence:.2%})")
    plt.axis("off")
    plt.show()

    return classes[predicted_class], confidence

# Протестуйте на зображеннях з валідаційного набору
# Використайте зображення з val_dataset для тестування
```

Етап 4. Візуалізація рішень моделі за допомогою Grad-CAM

На цьому етапі ви реалізуєте Grad-CAM, щоб зрозуміти, на які частини зображення «дивиться» ваш класифікатор.

4.1. Реалізуйте Grad-CAM.

```
class GradCAM:
    """Реалізація Grad-CAM для візуалізації рішень CNN."""

    def __init__(self, model, target_layer):
        self.model = model
        self.target_layer = target_layer
        self.gradients = None
        self.activations = None

        # Реєструємо хуки для збору градієнтів та активацій
        target_layer.register_forward_hook(self._forward_hook)
        target_layer.register_full_backward_hook(self._backward_hook)

    def _forward_hook(self, module, input, output):
        self.activations = output.detach()

    def _backward_hook(self, module, grad_input, grad_output):
        self.gradients = grad_output[0].detach()

    def generate(self, input_tensor, target_class=None):
        """Генерує теплову карту Grad-CAM."""
        self.model.eval()
        output = self.model(input_tensor)

        if target_class is None:
            target_class = output.argmax(dim=1).item()

        self.model.zero_grad()
        output[0, target_class].backward()

        # Обчислюємо ваги: середнє градієнтів по просторових вимірах
        weights = self.gradients.mean(dim=[2, 3], keepdim=True)

        # Зважена сума активацій
        cam = (weights * self.activations).sum(dim=1, keepdim=True)
        cam = torch.relu(cam)

        # Нормалізація до [0, 1]
        cam = cam - cam.min()
        if cam.max() > 0:
            cam = cam / cam.max()

        return cam.squeeze().cpu().numpy()
```

4.2. Візуалізуйте теплові карти для зображень обох класів.

```
import torch.nn.functional as F

def visualize_gradcam(model, image_path, transform, classes, device):
    """Візуалізує оригінальне зображення та теплову карту Grad-CAM."""
    # Останній згортковий шар ResNet-18
    grad_cam = GradCAM(model, model.layer4[-1].conv2)

    image = Image.open(image_path).convert("RGB")
    input_tensor = transform(image).unsqueeze(0).to(device)

    # Передбачення
    model.eval()
    with torch.no_grad():
        output = model(input_tensor)
        probs = torch.softmax(output, dim=1)
        pred_class = probs.argmax(dim=1).item()

    # Генерація теплової карти (потрібен grad, тому без no_grad)
    input_tensor.requires_grad = True
    cam = grad_cam.generate(input_tensor)

    # Масштабуємо теплову карту до розміру зображення
    cam_resized = np.array(Image.fromarray(
        (cam * 255).astype(np.uint8)
    ).resize(image.size, Image.BILINEAR)) / 255.0

    # Візуалізація
    fig, axes = plt.subplots(1, 3, figsize=(18, 6))

    axes[0].imshow(image)
    axes[0].set_title("Оригінальне зображення")
    axes[0].axis("off")

    axes[1].imshow(cam_resized, cmap="jet")
    axes[1].set_title("Теплова карта Grad-CAM")
    axes[1].axis("off")

    axes[2].imshow(image)
    axes[2].imshow(cam_resized, cmap="jet", alpha=0.5)
    axes[2].set_title(
        f"Накладення | Клас: {classes[pred_class]} "
        f"({probs[0, pred_class]:.2%})"
    )
    axes[2].axis("off")

    plt.tight_layout()
    plt.show()

# Застосуйте до кількох зображень кожного класу з val_dataset
```

4.3. Проаналізуйте результати.

Дайте відповідь на наступні питання у звіті:

- На які частини зображення звертає увагу модель?

- Чи відрізняються ділянки уваги для різних класів?
- Чи завжди модель «дивиться» на правильний об'єкт? Чи бувають випадки, коли модель фокусується на фоні замість головного об'єкта?
- Яка якість (роздільність) теплової карти Grad-CAM? Чи можна за нею точно визначити межі об'єкта?

Етап 5. Семантична сегментація з попередньо натренованою моделлю

На цьому етапі ви використаєте попередньо натреновану модель DeepLabV3 для семантичної сегментації зображень.

5.1. Завантажте попередньо натреновану модель DeepLabV3.

```
from torchvision.models.segmentation import deeplabv3_resnet50, DeepLabV3_ResNet50_Weights

# Завантаження моделі з вагами, натренованими на COCO
seg_weights = DeepLabV3_ResNet50_Weights.COCO_WITH_VOC_LABELS_V1
seg_model = deeplabv3_resnet50(weights=seg_weights).to(device)
seg_model.eval()

# Класи COCO/VOC
COCO_CLASSES = [
    "фон", "аероплан", "велосипед", "птах", "човен",
    "пляшка", "автобус", "автомобіль", "кіт", "стілець",
    "корова", "обідній стіл", "собака", "кінь", "мотоцикл",
    "людина", "горщична рослина", "вівця", "диван", "потяг",
    "монітор"
]

# Кольорова палітра для візуалізації
def get_color_palette(num_classes):
    """Генерує палітру кольорів для візуалізації масок."""
    palette = np.zeros((num_classes, 3), dtype=np.uint8)
    for i in range(num_classes):
        r, g, b = 0, 0, 0
        c = i
        for j in range(8):
            r |= ((c >> 0) & 1) << (7 - j)
            g |= ((c >> 1) & 1) << (7 - j)
            b |= ((c >> 2) & 1) << (7 - j)
            c >>= 3
        palette[i] = [r, g, b]
    return palette

PALETTE = get_color_palette(len(COCO_CLASSES))
```

5.2. Виконайте сегментацію зображення.

```
seg_transform = transforms.Compose([
    transforms.Resize(520),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406],
                         std=[0.229, 0.224, 0.225])
])

def segment_image(model, image_path, transform, device):
    """Виконує семантичну сегментацію зображення."""
    image = Image.open(image_path).convert("RGB")
    input_tensor = transform(image).unsqueeze(0).to(device)
```

```
    with torch.no_grad():
        output = model(input_tensor)["out"]
        predictions = output.argmax(1).squeeze().cpu().numpy()

    return image, predictions

def visualize_segmentation(image, predictions, alpha=0.6):
    """Візуалізує результати сегментації."""
    # Створення кольорової маски
    color_mask = PALETTE[predictions]
    color_mask_img = Image.fromarray(color_mask).resize(image.size, Image.NEAREST)

    fig, axes = plt.subplots(1, 3, figsize=(20, 7))

    axes[0].imshow(image)
    axes[0].set_title("Оригінальне зображення")
    axes[0].axis("off")

    axes[1].imshow(color_mask_img)
    axes[1].set_title("Маска сегментації")
    axes[1].axis("off")

    # Накладення маски на оригінал
    blended = Image.blend(image.resize(color_mask_img.size),
                          color_mask_img.convert("RGB"), alpha=0.4)
    axes[2].imshow(blended)
    axes[2].set_title("Накладення маски")
    axes[2].axis("off")

    # Легенда: які класи виявлені
    unique_classes = np.unique(predictions)
    detected = [f"{COCO_CLASSES[c]} (id={c})" for c in unique_classes if c > 0]
    if detected:
        print(f"Виявлені класи: {', '.join(detected)}")
    else:
        print("Виявлено лише фон")

    plt.tight_layout()
    plt.show()
```

5.3. Застосуйте сегментацію до зображень з вашого набору даних.

```
# Застосуйте сегментацію до зображень з вашого набору даних.
# Оскільки Oxford-IIIT Pet містить котів та собак (які є класами COCO),
# DeepLabV3 має коректно їх сегментувати.
for i in range(min(3, len(val_data))):
    img, label = val_data[i]
    # Зберігаємо тимчасово для функції segment_image
    tmp_path = f"tmp_seg_{i}.jpg"
    img.convert("RGB").save(tmp_path)
    image, predictions = segment_image(seg_model, tmp_path, seg_transform, device)
    visualize_segmentation(image, predictions)
```

5.4. Сегментуйте додаткові зображення з інших класів набору даних.

Для більш різноманітних результатів сегментації, спробуйте зображення інших порід або категорій з вашого набору даних (не лише з двох класів вашого варіанта):

```
# Завантажуємо додаткові зображення з набору даних для різноманітності
# Наприклад, для Oxford-IIIT Pet — зображення інших порід
extra_dataset = OxfordIIITPet(root="data", split="trainval", download=False)
# Візьмемо кілька довільних зображень
import random
random.seed(123)
extra_indices = random.sample(range(len(extra_dataset)), 5)

for idx in extra_indices:
    img, label = extra_dataset[idx]
    tmp_path = f"tmp_seg_extra_{idx}.jpg"
    img.convert("RGB").save(tmp_path)
    print(f"Зображення: {extra_dataset.classes[label]}")
    image, predictions = segment_image(seg_model, tmp_path, seg_transform, device)
    visualize_segmentation(image, predictions)
```

5.5. Порівняйте результати Grad-CAM та сегментації.

Оберіть одне зображення, до якого ви застосовували Grad-CAM на етапі 4. Тепер застосуйте до нього семантичну сегментацію. Порівняйте:

- Теплову карту Grad-CAM (розмита область уваги класифікатора)
- Маску сегментації (чіткі піксельні межі об'єктів)

```
def compare_gradcam_and_segmentation(cls_model, seg_model, image_path,
                                     cls_transform, seg_transform,
                                     cls_classes, device):
    """Порівнює Grad-CAM класифікатора з семантичною сегментацією."""
    image = Image.open(image_path).convert("RGB")

    # Grad-CAM
    grad_cam = GradCAM(cls_model, cls_model.layer4[-1].conv2)
    cls_input = cls_transform(image).unsqueeze(0).to(device)
    cls_input.requires_grad = True
    cam = grad_cam.generate(cls_input)
    cam_resized = np.array(Image.fromarray(
        (cam * 255).astype(np.uint8)
    ).resize(image.size, Image.BILINEAR)) / 255.0

    # Сегментація
    seg_input = seg_transform(image).unsqueeze(0).to(device)
    with torch.no_grad():
        seg_output = seg_model(seg_input)["out"]
        seg_preds = seg_output.argmax(1).squeeze().cpu().numpy()
    seg_mask = PALETTE[seg_preds]
    seg_mask_img = Image.fromarray(seg_mask).resize(image.size, Image.NEAREST)

    # Візуалізація
    fig, axes = plt.subplots(1, 4, figsize=(24, 6))

    axes[0].imshow(image)
```

```
    axes[0].set_title("Оригінал")
    axes[0].axis("off")

    axes[1].imshow(image)
    axes[1].imshow(cam_resized, cmap="jet", alpha=0.5)
    axes[1].set_title("Grad-CAM (класифікатор)")
    axes[1].axis("off")

    axes[2].imshow(seg_mask_img)
    axes[2].set_title("Семантична сегментація")
    axes[2].axis("off")

    axes[3].imshow(image)
    axes[3].imshow(np.array(seg_mask_img), alpha=0.4)
    axes[3].set_title("Сегментація (накладення)")
    axes[3].axis("off")

    plt.tight_layout()
    plt.show()

# Застосуйте до зображень з вашого набору даних
# Збережіть зображення з val_data у файл і передайте шлях до функції
```

Етап 6. Комбінований конвеєр: сегментація + класифікація

На цьому етапі ви побудуєте конвеєр, що спочатку сегментує зображення для виділення областей інтересу, а потім класифікує ці області.

6.1. Реалізуйте конвеєр.

```
def extract_regions_by_class(image, predictions, target_classes, min_area=500):
    """
    Витягує прямокутні регіони зображення,
    що відповідають вказаним класам сегментації.
    """
    image_np = np.array(image)
    regions = []

    for cls_id in target_classes:
        mask = (predictions == cls_id)
        if mask.sum() < min_area:
            continue

        # Масштабуємо маску до розміру зображення
        mask_resized = np.array(
            Image.fromarray(mask.astype(np.uint8) * 255).resize(
                image.size, Image.NEAREST
            )
        ) > 0

        # Знаходимо bounding box
        rows = np.any(mask_resized, axis=1)
        cols = np.any(mask_resized, axis=0)
        rmin, rmax = np.where(rows)[0][[0, -1]]
        cmin, cmax = np.where(cols)[0][[0, -1]]

        # Вирізаємо регіон
        region = image_np[rmin:rmax+1, cmin:cmax+1]
        regions.append({
            "image": Image.fromarray(region),
            "class_id": cls_id,
            "class_name": COCO_CLASSES[cls_id],
            "bbox": (cmin, rmin, cmax, rmax)
        })

    return regions


def pipeline(cls_model, seg_model, image_path,
             cls_transform, seg_transform, cls_classes, device,
             target_seg_classes=None):
    """
    Комбінований конвеєр:
    1. Сегментує зображення
    2. Виділяє регіони за класами сегментації
    3. Класифікує кожен регіон власним класифікатором
    """
    image = Image.open(image_path).convert("RGB")
```

```
    # Крок 1: Сегментація
    seg_input = seg_transform(image).unsqueeze(0).to(device)
    with torch.no_grad():
        seg_output = seg_model(seg_input)["out"]
        seg_preds = seg_output.argmax(1).squeeze().cpu().numpy()

    # Крок 2: Витягуємо регіони
    if target_seg_classes is None:
        unique = np.unique(seg_preds)
        target_seg_classes = [c for c in unique if c > 0]  # без фону

    regions = extract_regions_by_class(image, seg_preds, target_seg_classes)

    if not regions:
        print("Не знайдено об'єктів для класифікації.")
        return

    # Крок 3: Класифікуємо кожен регіон
    print(f"Знайдено {len(regions)} регіон(ів) для класифікації:\n")

    fig, axes = plt.subplots(1, min(len(regions), 4), figsize=(20, 5))
    if len(regions) == 1:
        axes = [axes]

    for i, region in enumerate(regions[:4]):
        region_tensor = cls_transform(region["image"]).unsqueeze(0).to(device)
        cls_model.eval()
        with torch.no_grad():
            output = cls_model(region_tensor)
            probs = torch.softmax(output, dim=1)
            pred = probs.argmax(dim=1).item()
            conf = probs[0, pred].item()

        title = (f"Сегмент: {region['class_name']}\n"
                 f"Класифікація: {cls_classes[pred]} ({conf:.2%})")
        axes[i].imshow(region["image"])
        axes[i].set_title(title, fontsize=10)
        axes[i].axis("off")
        print(f"  Регіон {i+1}: сегмент '{region['class_name']}' → "
              f"класифіковано як '{cls_classes[pred]}' ({conf:.2%})")

    plt.tight_layout()
    plt.show()

# Застосуйте конвеєр до тестового зображення
# pipeline(model, seg_model, "test_segmentation/test_0.jpg",
#          val_transform, seg_transform, full_dataset.classes, device)
```

6.2. Протестуйте конвеєр на кількох зображеннях та проаналізуйте результати у звіті.

Етап 7 (бонусний, за бажанням). Тонке налаштування легкої моделі сегментації

УВАГА! Цей етап є необов'язковим і потребує додаткового часу GPU. Виконуйте його лише за наявності достатнього ліміту GPU (рекомендовано мати щонайменше 1 годину вільного GPU часу).

На цьому етапі ви спробуєте натренувати (fine-tune) легку модель сегментації на власних даних.

7.1. Підготуйте набір масок для тренування.

Для тренування моделі сегментації потрібні маски — зображення, де кожен піксель позначений класом. Ви можете використати маски, згенеровані попередньо натренованою моделлю DeepLabV3, як «псевдо-мітки» для тренування легшої моделі:

```
from torchvision.models.segmentation import fcn_resnet50, FCN_ResNet50_Weights

# Використовуємо легшу модель FCN замість DeepLabV3
light_model = fcn_resnet50(weights=None, num_classes=21).to(device)

# Або MobileNet-based модель (ще легша):
# from torchvision.models.segmentation import lraspp_mobilenet_v3_large
# light_model = lraspp_mobilenet_v3_large(weights=None, num_classes=21).to(device)
```

7.2. Створіть цикл тренування для сегментації.

```
seg_optimizer = optim.Adam(light_model.parameters(), lr=1e-4)
seg_criterion = nn.CrossEntropyLoss()

# Генеруємо псевдо-мітки з великої моделі
def generate_pseudo_labels(teacher_model, image_paths, transform, device):
    """Генерує маски сегментації за допомогою великої моделі."""
    teacher_model.eval()
    labels = []
    for path in image_paths:
        image = Image.open(path).convert("RGB")
        input_t = transform(image).unsqueeze(0).to(device)
        with torch.no_grad():
            output = teacher_model(input_t)["out"]
            pred = output.argmax(1).squeeze().cpu()
        labels.append(pred)
    return labels

# Тренуйте light_model на псевдо-мітках
# Деталі реалізації залишаються студенту як самостійне завдання
```

7.3. Порівняйте результати легкої моделі з оригінальною DeepLabV3.

Проаналізуйте:

- Швидкість виконання (час на одне зображення)
- Якість сегментації (візуально та за метрикою IoU, якщо можливо)

- Розмір моделі (кількість параметрів)

УМОВА ЗАВДАННЯ ДО ЛАБОРАТОРНОЇ РОБОТИ

1. Налаштувати середовище розробки (Google Colab, Kaggle або VS Code з віддаленим ядром).
2. Створити набір даних для бінарної класифікації зображень за темою індивідуального варіанта.
3. Побудувати та натренувати модель класифікації зображень на базі ResNet-18 з використанням передавального навчання.
4. Реалізувати Grad-CAM та проаналізувати, на які частини зображень звертає увагу модель.
5. Використати попередньо натреновану модель DeepLabV3 для семантичної сегментації зображень.
6. Порівняти результати Grad-CAM та семантичної сегментації.
7. Побудувати комбінований конвеєр «сегментація + класифікація».
8. (Бонус, за бажанням) Натренувати легку модель сегментації.

ІНДІВІДУАЛЬНІ ВАРІАНТИ ЗАВДАННЯ

Кожен студент має створити класифікатор для однієї з наступних пар класів. Оберіть **один** набір даних (Oxford-IIIT Pet **або** Caltech-101) та відповідну пару класів.

Варіанти з Oxford-IIIT Pet (породи котів та собак):

1. Siamese vs Persian
2. Beagle vs Boxer
3. Bengal vs Bombay
4. Maine Coon vs Ragdoll
5. Chihuahua vs Great Pyrenees
6. Pug vs Samoyed
7. Shiba Inu vs Yorkshire Terrier
8. British Shorthair vs Sphynx
9. American Bulldog vs Saint Bernard
10. German Shorthaired vs English Setter
11. Pomeranian vs Newfoundland
12. Russian Blue vs Abyssinian
13. Scottish Terrier vs Wheaten Terrier
14. Egyptian Mau vs Birman
15. Miniature Pinscher vs Leonberger
16. Basset Hound vs English Cocker Spaniel
17. Havanese vs Keeshond
18. Japanese Chin vs Staffordshire Bull Terrier

Варіанти з Caltech-101 (різноманітні категорії об'єктів):

19. airplanes vs motorbikes
20. butterfly vs dragonfly
21. elephant vs kangaroo
22. grand_piano vs accordion
23. laptop vs cellphone
24. lobster vs crab
25. sunflower vs lotus

ЗМІСТ ЗВІТУ

1. Тема та мета роботи
2. Теоретичні відомості (стислий виклад основних концепцій)
3. Постановка завдання (індивідуальний варіант)
4. Хід виконання роботи:
    - Скріншоти налаштування середовища розробки
    - Код та пояснення для підготовки набору даних
    - Скріншоти прикладів зображень з набору даних
    - Код та пояснення для створення та тренування моделі класифікації
    - Графіки навчання (крива втрат та точності)
    - Результати класифікації тестових зображень
    - Візуалізації Grad-CAM для кількох зображень кожного класу
    - Аналіз теплових карт Grad-CAM
    - Результати семантичної сегментації
    - Порівняння Grad-CAM та сегментації
    - Результати роботи комбінованого конвеєру
5. Результати роботи
6. Висновки

КОНТРОЛЬНІ ПИТАННЯ

1. Що таке навчання з учителем і чим воно відрізняється від навчання без учителя?
2. Які основні задачі комп'ютерного зору ви знаєте? Порівняйте класифікацію, виявлення об'єктів та сегментацію.
3. Що таке згорткова нейронна мережа? Опишіть основні типи шарів CNN.
4. Що таке передавальне навчання і чому воно ефективне при обмеженому обсязі даних?
5. Поясніть принцип роботи залишкових з'єднань (skip connections) в архітектурі ResNet.
6. Що таке аугментація даних? Які трансформації використовувались у лабораторній роботі?
7. Опишіть принцип роботи Grad-CAM. Які кроки необхідні для генерації теплової карти?
8. Чому Grad-CAM застосовується саме до останнього згорткового шару? Що станеться, якщо застосувати його до ранніших шарів?
9. Що таке семантична сегментація? Чим вона відрізняється від сегментації екземплярів?
10. Опишіть архітектуру DeepLabV3. Що таке atrous convolution і ASPP?
11. Порівняйте теплові карти Grad-CAM та маски семантичної сегментації. В чому принципова різниця?

12. Що таке функція втрат (loss function)? Яка функція використовувалась для класифікації?
13. Що таке перенавчання (overfitting)? Які методи його запобігання використані в лабораторній роботі?
14. Що таке learning rate scheduler? Для чого він використовується?
15. Чому для тренувальної та валідаційної вибірок використовуються різні трансформації?
16. Що таке batch size і як він впливає на процес навчання?
17. Опишіть метрику IoU (Intersection over Union). Чому вона є основною для оцінки сегментації?
18. Як можна використати комбінований конвеєр (сегментація + класифікація) в ігровому застосунку? Наведіть приклади.
19. Які переваги та недоліки використання хмарних GPU (Colab, Kaggle) порівняно з локальним обладнанням?
20. Поясніть різницю між «заморожуванням» шарів та повним тренуванням моделі при передавальному навчанні.

СПИСОК ЛІТЕРАТУРИ

1. He K. et al. Deep Residual Learning for Image Recognition // IEEE Conference on Computer Vision and Pattern Recognition (CVPR). -- 2016.
2. Selvaraju R.R. et al. Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization // International Journal of Computer Vision. -- 2020.
3. Chen L.-C. et al. Rethinking Atrous Convolution for Semantic Image Segmentation // arXiv preprint arXiv:1706.05587. -- 2017.
4. PyTorch Documentation [Електронний ресурс]. -- Режим доступу: https://pytorch.org/docs/stable/index.html
5. torchvision Documentation [Електронний ресурс]. -- Режим доступу: https://pytorch.org/vision/stable/index.html
6. Google Colab [Електронний ресурс]. -- Режим доступу: https://colab.research.google.com/
7. Kaggle [Електронний ресурс]. -- Режим доступу: https://www.kaggle.com/
8. Howard J., Gugger S. Deep Learning for Coders with fastai and PyTorch. -- O'Reilly Media, 2020.
9. Chollet F. Deep Learning with Python (2nd Edition). -- Manning Publications, 2021.
10. Stanford CS231n: Convolutional Neural Networks for Visual Recognition [Електронний ресурс]. -- Режим доступу: https://cs231n.stanford.edu/
11. Parkhi O. M. et al. Cats and Dogs // IEEE Conference on Computer Vision and Pattern Recognition (CVPR). -- 2012. (Oxford-IIIT Pet Dataset)

12. Fei-Fei L., Fergus R., Perona P. Learning Generative Visual Models from Few Training Examples: An Incremental Bayesian Approach Tested on 101 Object Categories // CVPR Workshop. -- 2004. (Caltech-101)

13. Lin T.-Y. et al. Microsoft COCO: Common Objects in Context // European Conference on Computer Vision (ECCV). -- 2014.

14. Deng J. et al. ImageNet: A Large-Scale Hierarchical Image Database // IEEE Conference on Computer Vision and Pattern Recognition (CVPR). -- 2009.

15. Goodfellow I., Bengio Y., Courville A. Deep Learning. -- MIT Press, 2016.

16. Simonyan K., Vedaldi A., Zisserman A. Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps // ICLR Workshop. -- 2014.

17. Google Colab is coming to VS Code [Електронний ресурс]. -- Режим доступу: https://developers.googleblog.com/google-colab-is-coming-to-vs-code/