ДОДАТОК А

Код функцій розрахунку оцінок

Лістинг А.1 – Код функцій розрахунку оцінок

```
def estimate_cultural_value(other_info,
default_building_type="residential"):
score = 0
protection_scores = {"unesco": 5, "national": 4,"local": 3,
None: 0}
protection_status = None
if other_info.get("ЮНЕСКО спадщина", "-") != "-":
protection_status = "unesco"
elif other_info.get("Пам'ятка національного значення",
"- ") != "-":
protection_status = "national"
elif other_info.get("Пам'ятка місцевого значення", "-") !=
"-":
protection_status = "local"
score += protection_scores.get(protection_status)
construction_year = None
for key in ["Кінець будівництва:", "Рік будівництва:",
"Дата будівництва:", "Будівництво:", "Будівництво"]:
if key in other_info and other_info[key]:
try: construction_year = int(other_info[key])
break
except (ValueError, TypeError):
continue
if construction_year:
if construction_year < 1800:
score += 4
elif 1800 <= construction_year < 1900:
score += 3
elif 1900 <= construction_year < 1950:
score += 2
else: score += 1
```

```
rare_styles = ["бароко", "класицизм", "еклектика",
"неоренесанс", "псевдоготичний", "неокласицизм",
"неовізантійський"]
common_styles = ["конструктивізм", "функціоналізм",
"модернизм", "ампір"]
architectural_style = other_info.get("Стиль:", None)
if architectural_style:
style = architectural_style.lower().split("/")[0].strip()
if any(rare_style in style for rare_style in rare_styles):
score += 2
elif any(common_style in style for common_style in
common_styles): score += 1
famous_architects = [
"Бекетов", "Ржепішевський", "Васильєв",
"Кушнарьов", "Величко", "Цауне"]
architect = other_info.get("Архітектори:", None)
if architect:
architect = architect.lower().strip()
if any(famous in architect for famous in
famous_architects): score += 2
else: score += 1
building_type = default_building_type
if architectural_style or architect or protection_status:
building_type = "cultural"
if building_type.lower() == "cultural":
score += 1
elif building_type.lower() == "residential":
score -= 1
max_score = 12
normalized_score = (score / max_score) * 10
final_score = max(1, min(10, round(normalized_score)))
return final_score
def estimate_functional_value(main_info,
project_info=None):
score = 0
building_purpose = main_info.get("Ім'я/мета:", "").lower()
```

```
building_type = "other"
extra_function = False
if any(keyword in building_purpose for keyword in
["житлов", "будинок", "комплекс"]):
building_type = "residential"
score += 7
if "з магазином" in building_purpose or "з комерційними" in
building_purpose:
extra_function = True
score += 0.5
elif any(keyword in building_purpose for keyword in
["лікарн", "медичн"]):
building_type = "hospital"
score += 8
elif any(keyword in building_purpose for keyword in
["школ", "гімназ", "ліцей", "інст", "універ","сад"]):
building_type = "school"
score += 7.5
elif any(keyword in building_purpose for keyword in
["магазин", "банк", "офіс", "театр", "бібліотек"]):
building_type = "other"
score += 5
elif any(keyword in building_purpose for keyword in
["склад", "завод"]):
building_type = "industrial"
score += 4
elif any(keyword in building_purpose for keyword in
["приватн", "особняк"]):
building_type = "private_house"
score += 2
condition = main_info.get("Поточний стан:", "").lower()
condition_scores = {
"використаний": 2,
"триває будова": 0.5,
"триває реконструкція": 0.5,
"не у вжитку (закинуто)": -2}
```

```
score += condition_scores.get(condition, 0)
if building_type in ["residential", "private_house"] and
project_info:
apartments_str = project_info.get("Квартири в будинку",
None)
num_apartments = 0
if apartments_str:
try:
import re
match = re.search(r"всього\s*-\s*(\d+)", apartments_str)
if match:
num_apartments = int(match.group(1))
except (ValueError, AttributeError):
num_apartments = 0
if building_type == "private_house":
num_apartments = 1
if num_apartments > 50:
score += 2
elif 20 <= num_apartments <= 50:
score += 1
if building_type == "residential":
score += 1
elif building_type in ["hospital", "school"]:
score += 0.5
max_score = 13.5
normalized_score = (score / max_score) * 10
final_score = max(1, min(10, round(normalized_score)))
return final_score
def estimate_damage_severity(counts, project_info=None):
damage_weights = {
"Collapsed building part": 20,
"Collapsed wall": 15,
"Big-Crack": 10,
"Damaged roof": 8,
"Medium-Crack": 7,
"Damaged facade": 6,
```

```
"Broken balcony": 5,
"Burn marks": 4,
"Small-Crack": 3,
"Broken Window": 2,
"Broken door": 2}
total_score = 0
for damage_type, count in counts.items():
if damage_type not in damage_weights or count <= 0:
continue
base_contribution = damage_weights[damage_type] * (1 +
math.log10(max(1, count)))
if count > 20:
base_contribution *= 1.5
material_factor = 1.0
bearing_factor = 1.0
roof_factor = 1.0
if project_info:
external_walls = project_info.get("Зовнішні стіни",
"").lower()
if damage_type in ["Big-Crack", "Medium-Crack", "Small-
Crack", "Damaged facade", "Collapsed wall"]:
if "цегла" in external_walls:
material_factor *= 0.9
elif "к/б панель" in external_walls:
material_factor *= 1.0
elif "гіпсобетон" in external_walls:
material_factor *= 1.1
if damage_type == "Collapsed building part":
internal_walls = project_info.get("Внутрішні стіни",
"").lower()
if "з/б" in internal_walls:
material_factor *= 0.9
elif "гіпсобетон" in internal_walls or "гіпсоблок" in
internal_walls:
material_factor *= 1.1
```

```
bearing_walls = project_info.get("Несучі стіни", "").lower()
if damage_type in ["Collapsed wall", "Collapsed building part"]:
if "поздовжні та поперечні" in bearing_walls and "стовпами" in bearing_walls:
bearing_factor *= 1.2
elif "поздовжні" in bearing_walls or "поперечні" in bearing_walls:
bearing_factor *= 1.1
if damage_type == "Damaged roof":
tech_floor = project_info.get("Техповерх", "").lower()
if tech_floor == "горище":
roof_factor *= 0.8
else:
roof_factor *= 1.1
contribution = base_contribution * material_factor * bearing_factor * roof_factor
total_score += contribution
max_score = 40
normalized_score = (total_score / max_score) * 10
final_score = max(1, min(10, round(normalized_score)))
return final_score
```

ДОДАТОК Б

Код маршрутів серверної частини Flask

Лістинг Б.1 – Код маршрутів серверної частини Flask

```
@app.route("/", methods=["GET"])
def list_buildings():
fn = 'saved_buildings.json'
if os.path.exists(fn):
with open(fn, 'r', encoding='utf-8') as f:
buildings = json.load(f)
else: buildings = []
return render_template('list.html', buildings=buildings)
@app.route('/add', methods=["GET"])
def add_building():
return render_template("index.html")
@app.route('/upload', methods=['POST'])
def upload():
file = request.files['file']
full_address: str = request.form['text']
if not file or not full_address:
return "Зображення або адреса не вказані!", 400
filename = file.filename
filepath = os.path.join(UPLOAD_FOLDER, filename)
file.save(filepath)
model_output = predict_and_visualize_return_data(filepath)
session['result_image'] = model_output['result_image']
session['counts'] = model_output['counts']
parts = [part.strip() for part in full_address.split(',')]
street_name = ""
house_number = ""
for i, part in enumerate(parts):
if part.strip().isdigit():
house_number = part.strip()
if i + 1 < len(parts):
street_name = parts[i + 1].strip() break
```

```
if not street_name or not house_number:
building_info = {'Помилка': 'Не вдалося витягти вулицю і номер будинку з адреси.'}
else: city_id = 70
translated_street = translate_street_name(street_name)
print(f"Was: {street_name}")
print(f"Become (translated): {translated_street}")
raw_info = get_building_info(city_id, translated_street, house_number)
if isinstance(raw_info, dict): building_info = {
translate_to_ukrainian(k): translate_to_ukrainian(v)
for k, v in raw_info.items()
if k not in ['Місцезнаходження:', 'Етажність:', '', 'Адреса', 'Серія-посилання']}
project_name = raw_info.get('Проект:')
series_url = raw_info.get('Серія-посилання')
if series_name:
print(f"[INFO] Знайдена серія: {series_name}, проєкт: {project_name}")
project_info = get_project_info(
series_name.strip(),
project_name.strip() if project_name else None,
series_url = series_url)
session['project_info'] = project_info
else: session['project_info'] = {'Помилка': 'Серія не знайдена'}
else: building_info = {'Помилка': raw_info}
session['project_info'] = {'Помилка': 'Не вдалося отримати інформацію про проект.'}
search_address = street_name + ", " + str(house_number)
monument_info = find_monument_info(search_address)
building_info.update(monument_info)
session['building_info'] = building_info
session['address'] = str(street_name+", " + str(house_number))
return redirect(url_for('info'))
```

```
@app.route('/info')
def info():
building_info = session.get('building_info', {})
project_info = session.get('project_info', {})
counts = session.get('counts')
result_image = session.get('result_image')
main_keys = ['Серія', 'Проєкт', 'Поточний стан',
'Назва/призначення', 'Адреса']
main_info = {}
other_info = {}
for key, value in building_info.items():
normalized_key = key.replace(":", "").strip().lower()
if any(normalized_key == mk.lower() for mk in main_keys):
main_info[key] = value
else: other_info[key] = value
session['other_info'] = other_info
session['main_info'] = main_info
return render_template('info.html',
result_image=result_image,
counts=counts,
main_info=main_info,
other_info=other_info,
project_info=project_info)
@app.route('/scores')
def scores():
cultural = estimate_cultural_value(session["other_info"])
functional =
estimate_functional_value(session["main_info"],
session["project_info"])
damage = estimate_damage_severity(session["counts"],
session["project_info"])
return jsonify({
"damage": damage,
"functional": functional,
"cultural": cultural})
@app.route('/list', methods=['GET'])
```

```
def show_saved_buildings():
with open('saved_buildings.json', 'r', encoding='utf-8') as
f:
buildings = json.load(f)
return render_template('list.html', buildings=buildings)
@app.route('/save', methods=['POST'])
def save_building():
address = session.get('address', "")
counts = session.get('counts', {})
damage = int(request.json.get('damage', 0))
functional = int(request.json.get('functional', 0))
cultural = int(request.json.get('cultural', 0))
record = {
"address": address,
"damage_counts": counts,
"scores": {
"damage": damage,
"functional": functional,
"cultural": cultural}}
fn = 'saved_buildings.json'
if os.path.exists(fn):
with open(fn, 'r', encoding='utf-8') as f: saved = []
saved.append(record)
with open(fn, 'w', encoding='utf-8') as f:
json.dump(saved, f, ensure_ascii=False, indent=2)
return jsonify({"status": "ok"}), 200
@app.route('/topsis', methods=['POST'])
def rank_topsis():
with open('saved_buildings.json', 'r', encoding='utf-8') as
f: buildings = json.load(f)
ranked = topsis_rank(buildings, ideal=(4, 10, 5),
anti_ideal=(8, 1, 1))
with open('saved_buildings.json', 'w', encoding='utf-8') as
f: json.dump(ranked, f, ensure_ascii=False, indent=2)
return redirect(url_for('show_saved_buildings'))
```

ДОДАТОК В

Код кастомного DataLoader для Mask R-CNN

Лістинг В.1 – Код функцій кастомного DataLoader з урахуванням ваг класів для Mask R-CNN

```
def get_sample_weights(dataset_dicts, class_weights):
sample_weights = []
for data in dataset_dicts:
classes_in_sample = set()
for ann in data.get("annotations", []):
classes_in_sample.add(ann["category_id"])
weights = [class_weights[c] for c in classes_in_sample if
c < len(class_weights)]
sample_weight = sum(weights) / len(weights) if weights else
1.0
sample_weights.append(sample_weight)
return sample_weights
sample_weights = get_sample_weights(dataset_dicts,
class_weights)
sampler = WeightedRandomSampler(weights=sample_weights,
num_samples=len(sample_weights), replacement=True)
def custom_train_loader(cfg, dataset_dicts):
mapper = DatasetMapper(cfg, is_train=True)
dataset = DatasetFromList(dataset_dicts, copy=False)
return
DataLoader(datasetbatch_size=cfg.SOLVER.IMS_PER_BATCH,
sampler=sampler,
collate_fn=trivial_batch_collator,
num_workers=cfg.DATALOADER.NUM_WORKERS,
pin_memory=True)
class CustomTrainer(DefaultTrainer):
@classmethod
def build_train_loader(cls, cfg):
dataset_dicts = DatasetCatalog.get(cfg.DATASETS.TRAIN[0])
return custom_train_loader(cfg, dataset_dicts)
```

ДОДАТОК Г

Відомість кваліфікаційної роботи

Позначення	Найменування	Дод. відомості
	Текстові документи	
1.	Пояснювальна записка	121 с.
	Інші документи	
2.	Презентаційні матеріали	__ плакатів

.	.	Прізвище та ініціали.	Підп.	Дата			
Розробила		Пиріг Н. Я.			Аналіз руйнувань міської інфраструктури та визначення черговості її відновлення	Шифр групи	Код напр./спец.
Перевірила		Чала Л. Е.				ІТШІ-21-1	122
Н.контр.		Малєєва І.А.				ХНУРЕ кафедра ШІ	
Затв.		Золотухін О.В.					