Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

Інститут електроенергетики
Факультет інформаційних технологій
Кафедра безпеки інформації та телекомунікацій

ПОЯСНЮВАЛЬНА ЗАПИСКА
кваліфікаційної роботи ступеня магістра

студента *Явтушенко Артема Олексійовича*

академічної групи *125м-22-2*

спеціальності *125 Кібербезпека*

спеціалізації

за освітньо - професійною програмою *Кібербезпека*

на тему *Виявлення загроз інформаційній безпеці підприємства на основі згорткових нейронних мереж*

Керівники	**Прізвище, ініціали**	**Оцінка за шкалою**		**Підпис**
		рейтинговою	**інституційною**	
кваліфікаційної роботи	к.т.н. доц. Мацюк С.М.			
розділів:				
спеціальний	к.т.н. доц. Мацюк С.М.			
економічний	к.е.н. доц. Пілова Д.П.	90	відмінно	
Рецензент				
Нормоконтролер	ст. викл. Мєшков В.І.			

Дніпро
2023

ЗАТВЕРДЖЕНО:
завідувач кафедри
безпеки інформації та телекомунікацій
_____ д.т.н., проф. Корнієнко В.І.
«_____»_____20___року

ЗАВДАННЯ
на кваліфікаційну роботу
ступеня магістра

студенту *Явтушенко Артема Олексійовича*

академічної групи *125м-22-2*

спеціальності *125 Кібербезпека*

за освітньо - професійною програмою *Кібербезпека*

на тему *Виявлення загроз інформаційній безпеці підприємства на основі згорткових нейронних мереж*

затверджену наказом ректора НТУ «Дніпровська політехніка» від _____ № _____

Розділ	Зміст	Термін виконання
Розділ 1	Огляд основних понять предметної області	
Розділ 2	Проектування і розробка додатку для виявлення загроз підприємству на основі згорткових нейронних мереж	
Розділ 3	Економічний аналіз доцільності розробки системи	

Завдання видано _____ _____
(підпис керівника) (прізвище, ініціали)

Дата видачі завдання: _____

Дата подання до екзаменаційної комісії: _____

Прийнято до виконання _____ Явтушенко А.О
(підпис студента) (прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка: 87с., 66 рис., 1 табл., 28 джерел.

Об’єкт розробки — ІТС підприємства.

Предмет розробки — метод виявлення загроз підприємству на основі згорткових нейронних мереж.

Метою роботи є розробка методу виявлення загроз підприємству на основі згорткових нейронних мереж.

У першому розділі розглянуте поняття нейронної мережі, її базова структура, види архітектур, основні компоненти та завдання. Ознайомились зі згортковими нейронними мережами у вирішенні задач мультикласової сегментації, проаналізовано основні напрямки застосування згорткових нейронних мереж у галузі кібербезпеки.

Розібрано процес розпізнавання об`єктів на зображеннях та відео, проаналізовано основні задачі розпізнавання об`єктів.

Проведено огляд існуючих методів використання згорткових нейронних мереж у кібербезпеці та сформовано постановку задачі.

У другому розділі проведено проектування і розробку моделі нейронної мережі за архітектурою UNet для виявлення загроз підприємству. Проведено навчання моделі з різними конфігураціями епох та вхідних даних. Протестовано роботу навченої нейронної мережі та підраховано метрики втрат та точності її роботи.

Третій розділ присвячено економічному аналізу доцільності розробки моделі нейронної мережі за архітектурою UNet для виявлення загроз підприємству, прорахована економічна доцільність створення та використання алгоритмів з машинним навчанням.

ІНФОРМАЦІЙНА БЕЗПЕКА, ЗГОРТКОВІ НЕЙРОННІ МЕРЕЖІ, МУЛЬТИКЛАСОВА СЕГМЕНТАЦІЯ, КОМП’ЮТЕРНИЙ ЗІР, ВИЯВЛЕННЯ ЗАГРОЗ.

ABSTRACT

Explanatory note: 87p., 66 fig., 1 tables., 28 sources.

The object of development is the ITS of the enterprise.

The subject of development is a method of detecting threats to the enterprise based on convolutional neural networks.

The purpose of the work is to develop a method for detecting threats to the enterprise based on convolutional neural networks.

The first chapter examines the concept of a neural network, its basic structure, types of architectures, main components and tasks. We got acquainted with convolutional neural networks in solving multiclass segmentation problems, analyzed the main areas of application of convolutional neural networks in the field of cyber security.

The process of object recognition in images and videos is analyzed, and the main tasks of object recognition are analyzed.

A review of the existing methods of vicinity of the hypothalamic neural measures in a cybersecurity specialist was carried out and a problem statement was formed.

The second section carried out the design and development of a neural network model behind the UNet architecture to identify threats to business. A model was developed with different configurations of epochs and input data. The third section is devoted to a cost-effective analysis of the integrity of developing a neural network model behind the UNet architecture to identify threats to business, the cost-effectiveness of the development and recovery of machine-generated algorithms is examined.

INFORMATION SECURITY, CONVERSION NEURAL NETWORKS, MULTICLASS SEGMENTATION, COMPUTER VISION, THREATS DETECTION.

СПИСОК УМОВНИХ СКОРОЧЕНЬ

AN — Attention Networks

BCE — Binary cross entropy

CapsNet — Capsule Networks

CAE — Convolutional Autoencoders

CNN — Convolutional Neural Networks, CNN

FNN — Feedforward Neural Networks, FNN

GAN — Generative Adversarial Networks

GNN — Graph Neural Networks

LSTM — Long Short-Term Memory

ReLU — Rectified Linear Unit

RNN — Recurrent Neural Networks, RNN

ШІ — штучний інтелект

ЗМІСТ

с.

ВСТУП 8

РОЗДІЛ 1АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ 10

1.1 Поняття нейронної мережі 10

1.2 Архітектури нейромереж 15

1.3 Комп'ютерний зір та розпізнавання об`єктів на зображеннях 17

1.5 Задача сегментації об`єктів на зображенні та її підвиди 20

1.6 Застосування згорткових нейронних мереж у кібербезпеці 22

1.7 Основні архітектури згорткових нейронних мереж для вирішення задач мультикласової сегментаці 24

1.8 Постановка задачі 26

Висновки до розділу 1 28

РОЗДІЛ 2СПЕЦІАЛЬНА ЧАСТИНА 29

2.1 Огляд інструментальних засобів розробки 29

2.2 Розробка системи 33

2.2.1 Розробка вхідного конвеєра даних TensorFlow 33

2.2.2 Розробка згорткової нейронної мережі за архітектурою UNet41

2.3 Функції втрат та метрики 46

2.4 Дослідження роботи моделі 51

2.4.1 Дослідження роботи моделі з 5 епохами навчання та кількістю даних, що 50 раз більше, ніж початкова кількість зображень. 51

2.4.2 Дослідження роботи моделі з десятьма епохами навчання та кількістю даних, що у п`ятдесят разів більше, ніж початкова кількість зображень. 53

2.4.3 Дослідження роботи моделі з 25 епохами навчання та кількістю даних, що 40 раз більше, ніж початкова кількість зображень. 56

Висновки до розділу 2 59

РОЗДІЛ 3 60

ЕКОНОМІЧНЕ ОБГРУНТУВАННЯ ДОЦІЛЬНОСТІ РОЗРОБКИ 60

3.1 Розрахунок витрат 60

3.1.1 Трудомісткість 60

3.1.2 Розрахунок витрат на створення програмного засобу для виявлення загроз підприємству на основі згорткової UNet нейронної мережі 63

3.1.3 Розрахунок поточних (експлуатаційних) витрат 66

3.2 Оцінка Величини збитку 69

3.3 Визначення та аналіз показників економічної ефективності 72

Висновки до розділу 3 74

ВИСНОВКИ 75

ПЕРЕЛІК ПОСИЛАНЬ 76

..... 79

ДОДАТОК А. Відомість матеріалів кваліфікаційної роботи 79

ДОДАТОК Б. Перелік документів на фізичному носії 80

ДОДАТОК Г. Відгук керівника економічного розділу 82

ВСТУП

В епоху високорозвинених технологій та швидкозмінних цифрових парадигм, інформаційна безпека підприємств набуває критичного значення у світлі постійного зростання кількості загроз. Збільшення обсягів цифрових даних та їх важливість для ефективного функціонування підприємств вимагають нових підходів та технологій для виявлення потенційних загроз інформаційній безпеці.

Ця робота спрямована на вирішення вищезазначених викликів за допомогою згорткових нейронних мереж для розпізнавання об'єктів на зображеннях в контексті забезпечення безпеки підприємства. Згорткові нейронні мережі визначаються своєю високою ефективністю у сфері обробки зображень, і їх потенціал застосування для виявлення аномалій та ідентифікації підозрілих об'єктів стає ключовим чинником в удосконаленні захисту підприємства.

В межах дослідження планується детально розглядати можливості застосування згорткових нейронних мереж для виявлення загроз інформаційній безпеці, а саме на їхній здатності розпізнавати потенційно неьезпечні об'єкти на зображеннях. Це дозволить не лише підвищити рівень захисту підприємства від потенційних загроз, але й проллє світло на нові можливості в розвитку інноваційних методів в області кібербезпеки.

Такий підхід визначається як важливий крок у напрямку створення ефективних та передових засобів виявлення загроз безпеці підприємства, що відповідає вимогам сучасних викликів.

Мета і завдання роботи обиралися повністю спираючись на все вище зазначене. Метою роботи є розробка програмного забезпечення для виявлення загроз безпеці підприємства на основі згорткових нейронних мереж. Для виконання поставленої мети слід дослідити види архітектур згорткових нейронних мереж, розібратися в задачах розпізнавання об`єктів на зображеннях та проаналізувати методи застосування згорткових нейронних

мереж у галузі кібербезпеки. Також необхідно буде побудувати програмно обрану у ході дослідження архітектуру згорткової нейронної мережі для поставленої задачі.

Об’єкт дослідження — ІТС підприємства.

Предмет дослідження — метод виявлення загроз підприємству на основі згорткових нейронних мереж.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Поняття нейронної мережі

Нейромережа — це математична модель, яка імітує структуру та функціонування біологічних нейронних мереж з метою вирішення різноманітних задач, таких як класифікація, регресія, прогнозування та генерація. В основі нейромереж лежать штучні нейрони, які об'єднуються в графові структури і передають сигнали один одному через ваги зв'язків.

Завдяки процесу навчання, під час якого ваги та зміщення між нейронами оптимізуються, нейромережі стають здатними до виявлення закономірностей та залежностей у вхідних даних. Нейромережі активно використовуються в таких галузях, як комп'ютерний зір, машинний переклад, розпізнавання мови та інше. [1]

Базова структура та компоненти нейромережі

Структура нейромережі складається з трьох основних типів шарів: вхідний шар, приховані шари та вихідний шар.

Шар у нейромережі — це група нейронів, які працюють разом і виконують певну функцію в мережі. Уявіть собі нейромережу як мультиповерховий будинок, де кожен поверх містить кімнати (нейрони).

Шари з'єднані між собою, як сходи між поверхами. Кожен шар має свою роль у процесі обробки інформації.

Є три основні типи шарів у нейромережі:

- вхідний шар: Це "перший поверх" будинку, де ми починаємо.

  Вхідний шар приймає дані ззовні, наприклад, зображення або текст, і передає їх у наступні шари. Вхідний шар не змінює дані, а лише служить точкою входу для них;

- приховані шари: Це "середні поверхи" будинку.

  Приховані шари забезпечують обробку вхідних даних і передачу інформації між шарами. Вони називаються "прихованими", оскільки їхні

результати не відображаються напряму на виході мережі. Кількість прихованих шарів та нейронів у них може варіюватися в залежності від складності задачі та архітектури мережі;

- вихідний шар: Це “останній поверх” будинку.

Вихідний шар формує результат, який нейромережа передбачає на основі вхідних даних. Результат може бути класифікацією, числовим значенням або іншою інформацією, залежно від типу задачі.

Отже, шари в нейромережі – це групи нейронів, які працюють разом і відповідають за різні етапи обробки інформації. Вони забезпечують нейромережі можливість адаптуватися до різних задач.

Усі нейрони в шарах з’єднані між собою через ваги зв’язків. Ваги відіграють важливу роль у навчанні нейромережі, оскільки вони визначають силу впливу одного нейрона на інший. В процесі навчання, ваги оптимізуються, щоб мінімізувати помилку передбачення мережі.

Крім ваг зв’язків, кожен нейрон має так зване зміщення (bias), яке дозволяє регулювати активацію нейрона незалежно від вхідного сигналу. Зміщення допомагає нейромережі легше адаптуватися до різних даних та виконувати більш гнучкі перетворення на вхідних даних.

Ще одним ключовим компонентом нейромережі є функція активації. Вона застосовується до кожного нейрона у прихованих та вихідних шарах, щоб визначити його активність на основі суми вхідних сигналів, помножених на відповідні ваги та додавання зміщення. Функція активації може бути лінійною або не лінійною, в залежності від типу задачі та архітектури мережі.

Деякі з найпопулярніших функцій активації включають сигмоїду, гіперболічний тангенс, ReLU (Rectified Linear Unit) та Softmax.

Навчання нейромережі полягає в оптимізації ваг зв’язків та зміщень на основі навчального набору даних. Для цього зазвичай використовують метод зворотного поширення помилки (backpropagation), який базується на градієнтному спуску. Процес навчання може тривати довгий час, залежно від розміру набору даних, архітектури мережі та складності задачі.

Отже, базова структура та компоненти нейромережі включають вхідний, прихований та вихідний шари, нейрони з вагами зв'язків та зміщеннями, а також функції активації. Разом вони сприяють адаптації нейромережі до вхідних даних та вирішенню складних задач. Зрозуміння цих компонентів та їх взаємодії допоможе новачкам краще розібратися в основах нейромереж та їх застосуваннях.[1]

Типи завдань, які виконують нейромережі

Нейромережі можуть виконувати різні типи завдань, залежно від архітектури мережі та навчальних даних. До основних видів завдань належать:

- класифікація:

  Визначення категорії, до якої належить певний об'єкт або подія, на основі його характеристик;
- регресія: Прогнозування числового значення на основі вхідних даних;
- генерація тексту:

  Створення тексту на основі навчальних даних, зазвичай використовується для автоматичного створення описів, статей або відповідей на запитання;
- обробка зображень:

  Розпізнавання об'єктів, тексту або осіб на зображеннях;[19]

Галузі застосування нейромереж

Нейромережі знаходять широке застосування в різних галузях промисловості та сферах життя завдяки своїй спроможності навчатися, адаптуватися та вирішувати складні проблеми. Наприклад:

- обробка мови та переклад:

  У сфері обробки природної мови нейромережі допомагають в розпізнаванні мови, аналізі емоцій, генерації тексту та автоматичному перекладі. Прикладом може служити Google Translate, який використовує

нейромережі для перекладу текстів між різними мовами з високою точністю;

- розпізнавання образів та комп'ютерний зір:

Нейромережі широко використовуються для розпізнавання образів, від класифікації зображень до виявлення об'єктів на відео. Такі системи можна зустріти в системах безпеки, автономних автомобілях та медичній діагностиці;

- рекомендаційні системи:

Нейромережі допомагають рекомендаційним системам пропонувати користувачам товари та послуги на основі їх інтересів та взаємодії з платформами. Наприклад, Netflix та Amazon використовують нейромережі для аналізу пристрастей користувачів та надання відповідних рекомендацій;

- фінанси та біржовий ринок:

У фінансовій сфері нейромережі застосовуються для прогнозування курсів валют, ринкових трендів та оцінки кредитоспроможності клієнтів. Це допомагає фінансовим установам приймати обґрунтовані рішення та знижувати рівень ризику;

- медицина:

У медичній галузі нейромережі використовуються для розпізнавання патологічних змін на зображеннях, отриманих за допомогою МРТ, КТ, рентгену та інших діагностичних методів. Вони також допомагають в аналізі генетичних даних, прогнозуванні результатів лікування та розробці нових лікарських засобів;

- геологія та кліматологія:

У геології та кліматології нейромережі застосовуються для аналізу та прогнозування землетрусів, повеней та інших природних катаклізмів. Це допомагає науковцям та організаціям планувати заходи безпеки та зменшувати наслідки стихійних лих;

- відеоігри та віртуальна реальність:

У відеоіграх та віртуальній реальності нейромережі використовуються для створення реалістичних штучних інтелектів, які контролюють персонажів, ворогів та інші аспекти ігрового середовища. Це забезпечує глибше занурення та вищу якість ігрового процесу;

- маркетинг та реклама:

У маркетингу та рекламі нейромережі допомагають аналізувати дані про поведінку споживачів, прогнозувати тенденції та розробляти ефективні рекламні кампанії. Вони також можуть оптимізувати розміщення реклами на веб-сайтах та в соціальних медіа, що збільшує конверсію та віддачу від рекламних інвестицій;

- смарт-міста:

У смарт-містах нейромережі використовуються для управління розумним освітленням, контролю транспорту, розподілу енергії та забезпечення безпеки. Це допомагає оптимізувати ресурси, підвищити енергоефективність та забезпечити високий рівень комфорту для мешканців міст;

- робототехніка:

Нейромережі використовуються в робототехніці для навчання роботів розуміти та інтерпретувати дії людей, навігації в незнайомому середовищі та виконання складних завдань. Це відкриває можливості створення роботів, які можуть працювати поряд з людьми, надавати допомогу та виконувати різні завдання в промисловості та домашньому середовищі;

- Біотехнології:

У біотехнологіях нейромережі використовуються для розуміння складних хімічних реакцій, моделювання біологічних процесів та розробки нових лікарських засобів. Вони можуть прискорити процес відкриття нових

препаратів та допомогти науковцям знаходити інноваційні рішення в лікуванні хвороб;

Отже, нейромережі відіграють важливу роль у вирішенні проблем, виконуючи різні типи завдань у багатьох галузях промисловості та сферах життя. Завдяки своїй спроможності навчатися і адаптуватися до різних даних, нейромережі стають все більш популярними та корисними інструментами для сучасного світу.[20]

1.2 Архітектури нейромереж

Існує декілька основних архітектур нейромереж, кожна з яких призначена для різних типів задач та застосувань. Ось деякі з них:

- штучні нейронні мережі прямого поширення (Feedforward Neural Networks, FNN):

Це найпростіша архітектура нейромережі, в якій інформація передається в одному напрямку, від входу до виходу через різні шари. Вони не мають циклів або зворотних зв'язків і складаються з одного або кількох прихованих шарів;

- згорткові нейронні мережі:

(Convolutional Neural Networks, CNN): Згорткові нейромережі розроблені спеціально для роботи з даними, що мають просторову структуру, такими як зображення. Вони використовують згорткові шари для автоматичного виявлення особливостей зображень, замість ручного інженерного проектування особливостей;

- рекурентні нейронні мережі:

(Recurrent Neural Networks, RNN): Рекурентні нейромережі розроблені для роботи з послідовними даними, такими як текст або часові ряди. Вони мають зворотні зв'язки, які дозволяють пам'ятати інформацію з попередніх кроків. Це дозволяє їм краще працювати з даними, де контекст важливий;

- довга короткочасна пам’ять:

(Long Short-Term Memory, LSTM): LSTM – це різновид рекурентної нейромережі, який вирішує проблему затухання градієнта, яка виникає при навчанні традиційних RNN. LSTM має спеціальну структуру вузлів, які дозволяють моделі “пам’ятати” або “забувати” інформацію на протязі тривалого періоду часу;

- мережі згорткового автоенкодера:

(Convolutional Autoencoders, CAE): Автоенкодери — це нейромережі, що навчаються кодувати вхідні дані в компактному представленні, а потім реконструювати вихідні дані з цього представлення. Згорткові автоенкодери використовують згорткові шари для роботи з просторовими даними, такими як зображення, і зазвичай використовуються для виявлення особливостей або відтворення зображень;

- генеративно-змагальні мережі:

(Generative Adversarial Networks, GAN): Генеративно-змагальні мережі складаються з двох окремих нейромереж, які працюють разом: генератора, який створює синтетичні дані, і дискримінатора, який навчається відрізняти справжні дані від синтетичних. GAN-и зазвичай використовуються для генерації зображень, текстів та інших видів даних;

- мережі капсул:

(Capsule Networks, CapsNet): Мережі капсул — це тип згорткових нейромереж, які використовують спеціальні капсульні шари для збереження інформації про просторові відносини між об’єктами на зображенні. Вони можуть краще розуміти ієрархічні структури даних і зазвичай працюють краще, ніж традиційні згорткові мережі, для задач розпізнавання об’єктів;

- мережі з увагою:

(Attention Networks): Мережі з увагою — це архітектура, яка дозволяє моделям приділяти більше уваги важливим частинам вхідних даних.

Механізм уваги зазвичай використовується у комбінації з рекурентними або трансформаторними мережами, особливо в задачах обробки природного мовлення, таких як машинний переклад та генерація тексту;

- мережі граф-нейронів:

(Graph Neural Networks, GNN): GNN – це архітектура нейромереж, яка працює з графічними структурами даних. Вони здатні обробляти відносини між об’єктами і агрегувати інформацію з сусідніх вузлів. GNN часто використовуються в рекомендаційних системах, аналізі соціальних мереж та хімічному моделюванні;

- спайкові нейронні мережі:

Це клас мереж, які намагаються більш точно моделювати динаміку спайкових нейронів біологічного мозку. Вони роблять це шляхом впровадження темпоральної компоненти в активаційні функції нейронів.[8][9]

1.3 Комп’ютерний зір та розпізнавання об`єктів на зображеннях

Комп'ютерний зір - це галузь штучного інтелекту (ШІ), яка дозволяє комп'ютерам та системам отримувати значущу інформацію з цифрових зображень, відео та інших візуальних вхідних даних, а також вживати заходи або надавати рекомендації на основі цієї інформації. Якщо ШІ дозволяє комп'ютерам мислити, то комп'ютерне бачення дозволяє їм бачити, спостерігати і розуміти.

Комп'ютерний зір працює приблизно так само, як і людське бачення, за винятком того, що люди мають певну перевагу. Людське зорове сприйняття має перевагу в тому, що за допомогою десятиліть в контексті люди навчаються розрізняти об'єкти, визначати відстань до них, визначати, чи рухаються вони, і чи є щось неправильне на зображенні.

Комп'ютерний зір навчає машини виконувати ці функції, проте вони повинні робити це набагато швидше, використовуючи камери, дані та

алгоритми, а не сітківки ока, зорові нерви та зорову кору. Оскільки система, навчена оглядати продукцію або відслідковувати виробничий ресурс, може аналізувати тисячі продуктів чи процесів за хвилину, помічаючи непомітні дефекти чи проблеми, вона може швидко перевершити людські можливості.

Комп'ютерний зір використовується в промисловості, що охоплює енергетику та комунальні послуги, виробництво та автомобільну галузь, і цей ринок продовжує рости.[4]

Як працює комп'ютерний зір?

Комп'ютерному зору потрібно багато даних. Система з комп`ютерним зором проводить аналіз даних знову і знову, поки не розрізнить відмінності у даних і, в кінцевому рахунку, не розпізнає зображення або об`єкти на зображенні. Наприклад, щоб навчити комп'ютер впізнавати автомобільні шини, йому потрібно подавати величезні кількості зображень шин та пов'язаних з ними предметів, щоб алгоритм міг вивчити відмінності і впізнавати шини, особливо ті, що не мають дефектів.

Для досягнення цього використовуються дві ключові технології: глибоке навчання, та згорткова нейронна мережа (CNN).

Машинне навчання використовує алгоритмічні моделі, які дозволяють комп'ютеру самостійно вивчати контекст візуальних даних. Якщо через модель подається достатньо даних, комп'ютер буде аналізувати дані і навчиться розрізняти одне зображення від іншого. Алгоритми дозволяють машині самостійно навчатися, уникаючи необхідності людині програмувати її для розпізнавання зображень.[13]

Згорткова нейронна мережа (CNN) допомагає моделі машинного навчання або глибокого навчання розрізняти об`єкти на зображенні, розбиваючи зображення на пікселі, яким присвоюються теги або мітки. Вона використовує ці мітки для проведення згорток (математична операція над двома функціями для отримання третьої функції) та робить прогнози щодо того, що вона "бачить". Нейронна мережа виконує згортки та перевіряє

точність своїх прогнозів в серії ітерацій до того моменту, поки прогнози не починають дійсно збуватися. Після цього вона починає впізнавати або "бачити" зображення подібно до людей.

Схоже на те, як людина розглядає зображення на відстані, CNN спочатку визначає жорсткі контури та прості форми, а потім заповнює інформацію під час ітерацій своїх прогнозів. CNN використовується для розуміння окремих зображень. Рекурентна нейронна мережа (RNN) використовується аналогічним чином для відео застосувань, щоб допомогти комп'ютерам розуміти, як зображення в серії кадрів пов'язані одне з одним.[14]

1.4 Задачі розпізнавання об`єктів на зображенні

Розпізнавання в штучному інтелекті включає в себе різні види задач, які можуть бути вирішені за допомогою комп'ютерних алгоритмів і моделей машинного навчання. Основні види задач розпізнавання об`єктів на зображенні виглядають так:

- виявлення об'єктів (Object Detection):

  Визначення та локалізація різних об'єктів на зображенні, присвоєння прямокутних рамок (bounding boxes) навколо них та призначення класу кожному об'єкту;

- класифікація об'єктів (Object Classification):

  Визначення класу або категорії, до якої належить об'єкт на зображенні. Це може бути один клас з кількох можливих;

- мультикласове виявлення об'єктів (Multiclass Object Detection):

  Розширення виявлення об'єктів для сценаріїв, де на зображенні може бути більше одного різного класу об'єктів;

- сегментація об'єктів (Object Segmentation):

  Визначення меж та виділення областей, які належать різним об'єктам на зображенні;[14]

1.5 Задача сегментації об`єктів на зображенні та її підвиди

Класифікація та сегментація зображень - це завдання на обробку вхідного зображення та виведення класу (кішка, собака тощо) або ймовірності класів, які найкраще описують зображення. Для людей цей процес є однією з перших навичок, яку ми засвоюємо з моменту свого народження, і така, яка приходить природним шляхом і без зусиль. Без великих зусиль людина може швидко та з великою точністю визначити оточення, в якому ми перебуваємо, а також об’єкти, які нас оточують. Коли ми бачимо зображення або спостерігаємо навколишній світ, більшу частину часу ми можемо відразу охарактеризувати сцену і надати кожному об’єкту мітку, навіть не усвідомлюючи цього.[3]

Припустимо, є зображення красивого міського пейзажу. Що бачить людина? На перший погляд, людина бачить багато будівель та кольорів, але комп’ютер обробляє таке зображення дещо по іншому[2, с.1]. Рисунок 1.1 демонструє приклад зображення.

Рисунок 1.1 — Приклад зображення

Коли комп’ютер аналізує зображення (приймає зображення як вхід), він бачить багатовимірний масив. Залежно від роздільної здатності та розміру зображення, він побачить масив чисел розмірності h * w * ch, де:

- h - висота зображення в пікселях;
- w - ширина зображення в пікселях;

- ch - кількість кольорових каналів.

Рисунок 1.2 демонструє інтерпретацію зображення комп’ютером.

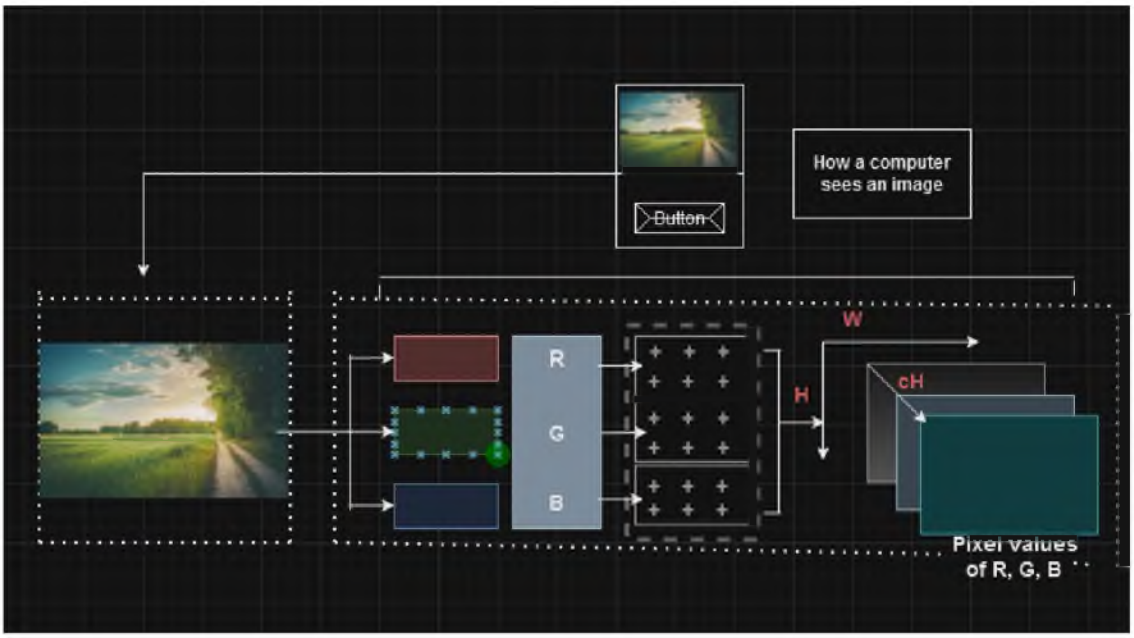


Рисунок 1.2 — Інтерпретація зображення комп`ютером

Сегментація в контексті обробки зображень та комп'ютерного зору - це процес розділення зображення на частини або "сегменти" з метою визначення та аналізу окремих об'єктів або областей в зображенні. Основна мета сегментації - визначити, які частини зображення відповідають різним об'єктам або структурам.

Основні види сегментації:

- бінарна сегментація:

Розділення зображення на два класи, зазвичай "об'єкт" та "фон". Одним з прикладів бінарної сегментації є виділення об'єктів на фотографії;

- мультикласова сегментація:

Розділення зображення на більше ніж два класи. Кожен піксель може бути призначений одному з кількох класів;

- сегментація за регіонами:

Розділення зображення на області, в яких кожен регіон може представляти окремий об'єкт чи структуру;

- семантична сегментація:

Призначення кожному пікселю на зображенні класу, що відповідає його семантичному значенню. Наприклад, визначення областей на зображенні як "людина", "автомобіль", "дерево" і т. д.;

- інстанційна сегментація:

Визначення окремих об'єктів на зображенні та виділення їхніх індивідуальних екземплярів.

Сегментація зображень має широкий спектр застосувань: медична діагностика, системи безпеки, автоматичне водіння, аналіз супутникових знімків, виробничий контроль, тощо.[3]

1.6 Застосування згорткових нейронних мереж у кібербезпеці

Згорткові нейронні мережі (Convolutional Neural Networks, або CNN) широко використовуються в кібербезпеці через їхню здатність ефективно впізнавати шаблони та аномалії у великих наборах даних. Ось кілька областей застосування згорткових нейронних мереж у кібербезпеці:

1) Виявлення загроз та атак:

- виявлення вторгнень:

CNN може бути використаний для аналізу мережевого трафіку та виявлення аномалій, що можуть свідчити про вторгнення або інші загрози безпеці;

- виявлення зловмисних програм:

CNN може аналізувати виконуваний код або мережевий трафік для виявлення характеристик, що вказують на наявність зловмисних програм.

2) Фільтрація спаму та шахрайства:

- фільтрація електронної пошти:

CNN може аналізувати текст та вкладення електронної пошти для виявлення спаму, фішингових повідомлень та інших шахрайських схем;

- виявлення шахрайства в онлайн-транзакціях:

Аналіз транзакційних даних за допомогою CNN допомагає виявляти підозрілі операції та шахрайські схеми.

3) Аналіз трафіку та виявлення вразливостей:

- моніторинг мережевого трафіку:

CNN може аналізувати пакети мережі для виявлення аномалій або атак на рівні пакетів;

- сканування вразливостей:

Використання CNN для аналізу великих обсягів даних може допомогти виявляти патерни, що вказують на вразливості в системах або програмному забезпеченні.

4) Аналіз великих обсягів даних:

- виявлення аномалій в журналах:

CNN може аналізувати журнали подій для виявлення незвичайних патернів або підозрілих активностей;

- класифікація поведінки користувачів:

Використання CNN для аналізу користувацької активності допомагає виявляти підозрілі дії та аномалії в поведінці користувачів.

5) Виявлення загроз на зображеннях, відео та обробка медіаданих:

- аналіз зображень та відео для виявлення загроз:

CNN може використовуватися для виявлення загроз на зображеннях та відео файлах за допомогою мультикласової сегментації об`єктів на медіа файлах в онлайн режимі.

Згорткові нейронні мережі є дуже корисними у кібербезпеці завдяки своїй здатності автоматично визначати важливі функціональні особливості та здатності розпізнавати складні патерни в даних. Особливо слід виділити використання згорткових нейромереж для аналізу зображень та відео. Даний тип нейромереж чудово працює з мультикласовою сегментацією об`єктів у медіафалах з досить високим рівнем точності.

1.7 Основні архітектури згорткових нейронних мереж для вирішення задач мультикласової сегментації

Для вирішення задачі мультикласової сегментації часто використовуються згорткові нейронні мережі (CNN). Ось деякі з основних архітектур, які здійснюють успішні результати в цьому напрямку:

U-Net

Дана архітектура побудована на високорівневих згортках для екстракції ознак і згортках із зменшеною роздільною здатністю (transposed convolutions або upsampling) для відновлення просторових розмірів. Ця архітектура є ефективною для завдань мультикласової сегментації, а також для аналізу біомедичних зображень.[19] Схема архітектури U-Net представлена на рисунку 1.3.

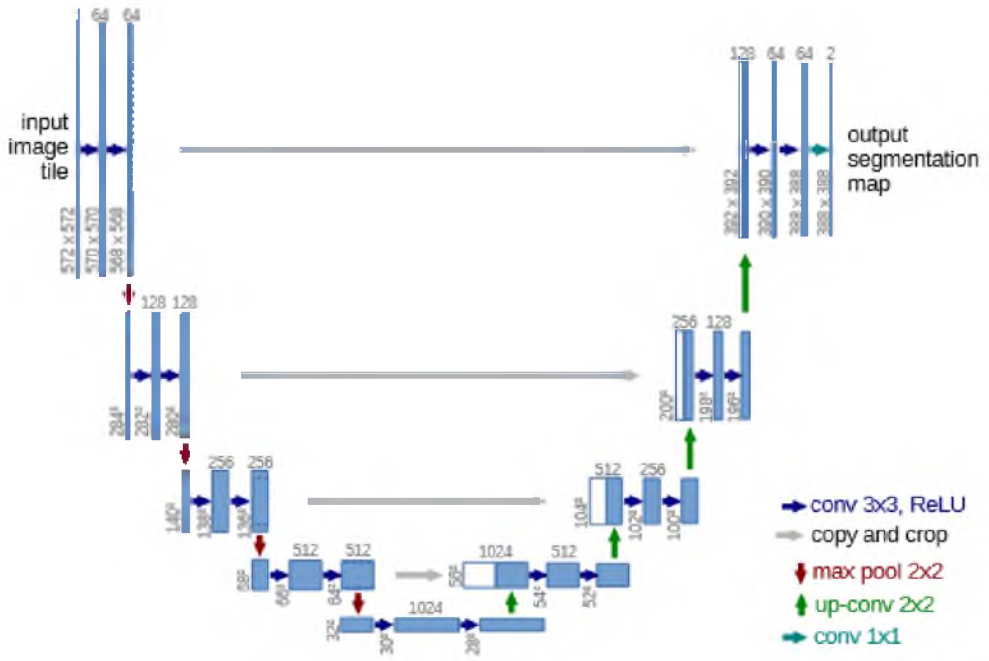


Рисунок 1.3 — Схема архітектури U-Net[19]

DeepLab

Ця архітектура базується на основі мережі згорткових шарів та асиметричних декодерів для отримання детальної інформації. DeepLab зазвичай використовується для сегментації зображень високої роздільної здатності. Схема архітектури DeepLab представлена на рисунку 1.4.[17]

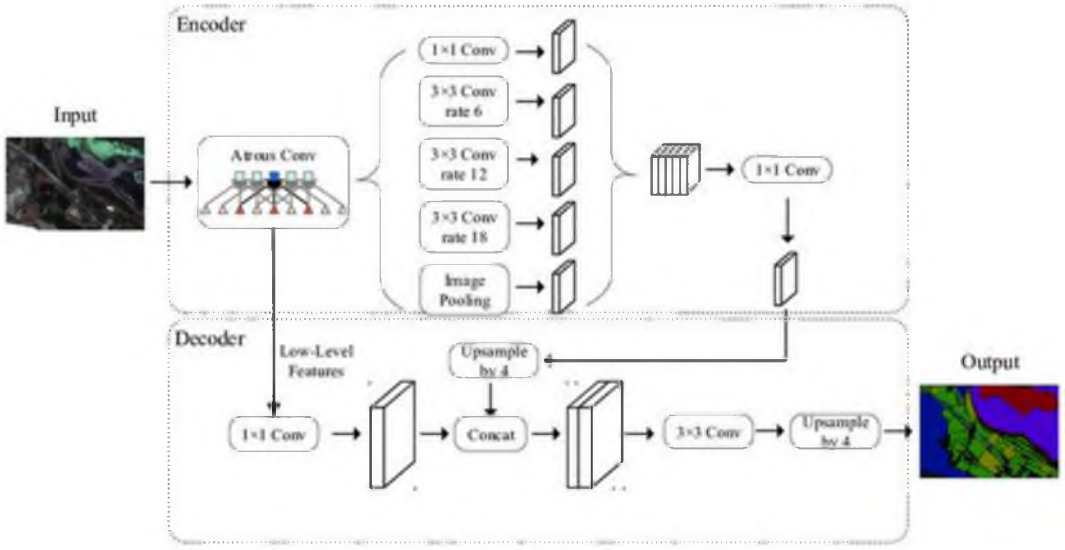


Рисунок 1.4 — Схема архітектури DeepLab[17]

SegNet

Архітектура SegNet використовує енкодер-декодер, де використовуються підгрупи згорткових і пулінгових шарів для екстракції та відновлення інформації. Ця архітектура є більш зручною для задач, що потрібно виконувати у реальному часі, завдяки меншій кількості параметрів. Схема архітектури SegNet представлена на рисунку 1.5.[16]

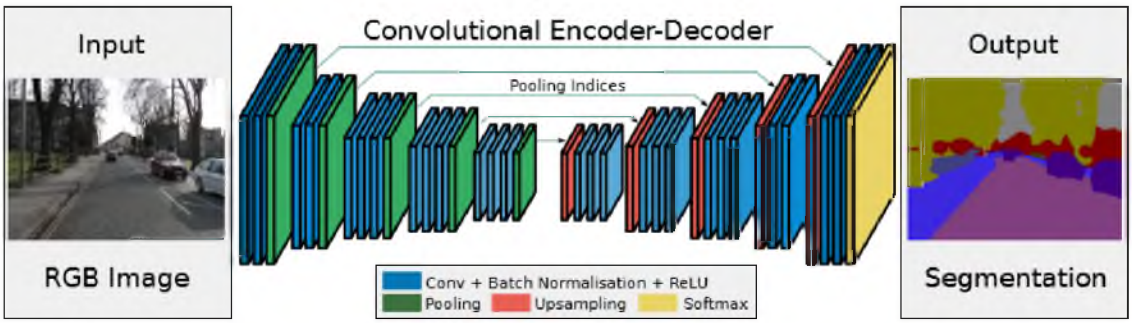


Рисунок 1.5 — Схема архітектури SegNet

FCN (Fully Convolutional Network):

FCN архітектура використовує лише згорткові та транспоновані згорткові шари для сегментації, замість повністю пов'язаних шарів. Ця архітектура є досить універсальною, тому використовується для сегментації зображень будь-якого розміру. Схема архітектури FCN представлена на рисунку 1.6.[16]

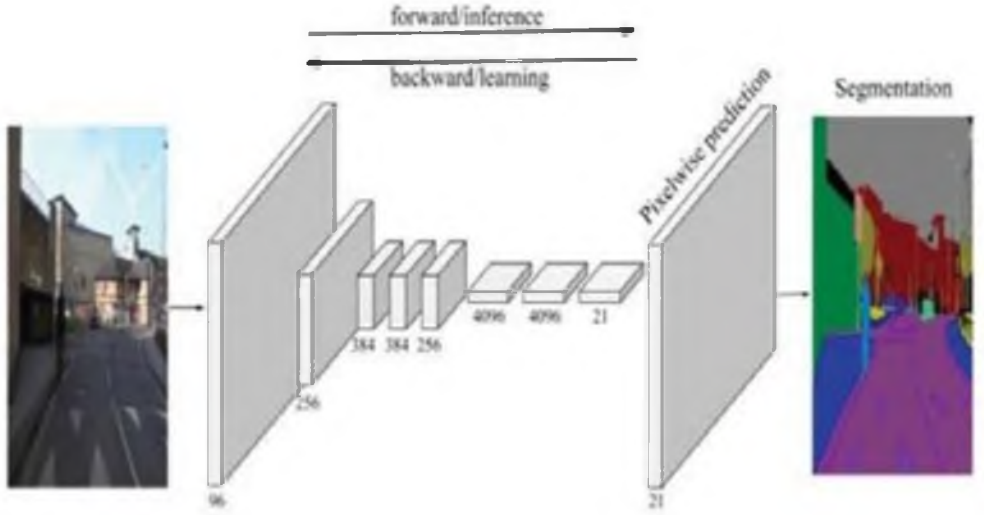


Рисунок 1.6 — Схема архітектури FCN

PSPNet (Pyramid Scene Parsing Network):

Дана архітектура включає пірамідальний модуль, який дозволяє моделі аналізувати зображення на різних рівнях роздільної здатності. PSPNet зазвичай використовують для аналізу широкого контексту в зображенні для кращої сегментації. Схема архітектури PSPNet представлена на рисунку 1.7.

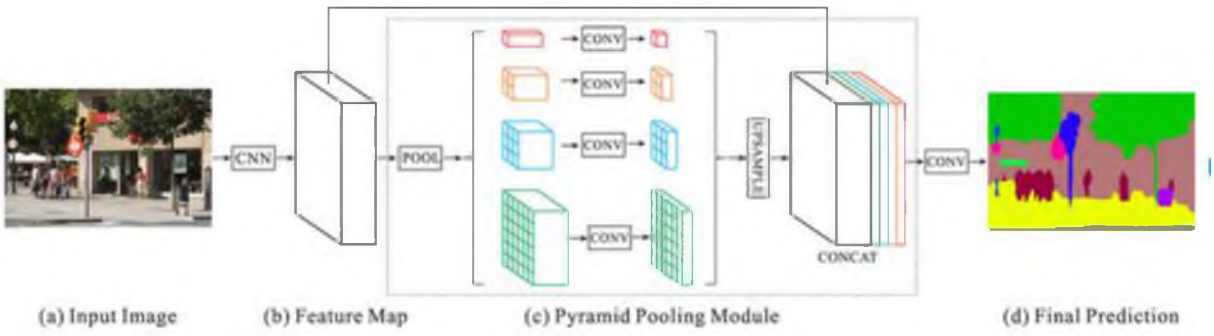


Рисунок 1.7 — Схема архітектури PSPNet

1.8 Постановка задачі

Обрана згорткова нейронна мережа має бути побудована за архітектурою U-Net та приймати на вхід зображення, що формуються з камер відеоспостереження, розпізнавати небезпечні об`єкти на території підприємства та помічати їх на вихідних зображеннях. Розпізнавання об’єктів

на зображеннях небхідно виконати за допомогою мультикласової сегментації для ситуацій, коли на зображенні може бути і декілька потенційно небезпечних об’єкти.

Схема підприємства зображена на рисунку 1.8

Рисунок 1.8 — Схема підприємства

Навчання нейронної мережі буде проводитися на власноруч створеному датасеті, що являє собою відео з довільними потенційними небезпечними об’єктами. Відео слід поділити на зображення та помітити на них потенційно небезпечні об’єкти, що будуть розмічені за допомогою бітових масок. Отже навчальний датасет буде складатися з отриманих з відео зображень та відповідних до них міток (бітових масок) зі значенням класу для кожної. Таким чином, основна задача полягає в розробленні програмного модуля для виявлення загроз підприємству з використанням згорткової нейронної мережі архітектури U-Net, який буде використовуватися на віддаленому сервері, до якого підключені камери відеоспостереження підприємства. Також проведемо тестування навченної нейронної мережі, підрахуємо функції втрат та метрики для задач мультикласової сегментації (Binary Cross Entropy та Dice) [12]

Висновки до розділу 1

В ході написання першого розділу було проаналізовано основні поняття предметної області, такі як згорткові нейронні мережі та їх застосування у галузі кібербезпеки, проаналізовано основні види архітектур згорткових нейронних мереж і задачі, які можно виконувати з ними. Детально розглянуто архітектуру U-Net, поняття комп'ютерного зору та задачі мультикласової сегментації зображень та відео.

Проведено огляд існуючих методів використання згорткових нейронних мереж у кібербезпеці.

Результатом розділу стала постановка задачі на розробку програмного забезпечення для виявлення загроз підприємству з використанням згорткової нейронної мережі архітектури U-Net та мультикласової сегментації зображень з метою підвищення рівня безпеки підприємства.

РОЗДІЛ 2
СПЕЦІАЛЬНА ЧАСТИНА

2.1 Огляд інструментальних засобів розробки

Мова програмування Python була створена Гвідо ван Россумом та була вперше випущена у 1991 році. Її створення було обумовлене бажанням створити мову, яка б поєднувала простоту, читабельність коду та ефективність програмування. Назва "Python" походить від ім'я британського комедійного телешоу "Monty Python's Flying Circus", яке було улюбленим Гвідо ван Россума.

Python пройшов через кілька версій, із збереженням сумісності. Зокрема, версія 2.x і версія 3.x існують паралельно, і розробники мають можливість вибору між ними залежно від вимог проекту.

Особливості синтаксису мови програмування Python:

- читабельність:

  Синтаксис Python приділяється великою увагою читабельності коду. Використання пробілів для відступів замість фігурних дужок чи ключових слів дозволяє писати код, який легше читати;

- динамічна типізація:

  Змінні в Python не потребують зазначення типів, оскільки тип визначається автоматично під час виконання програми;

- широкий вибір бібліотек:

  Python має велику кількість бібліотек для різних завдань, що дозволяє розробникам використовувати готові рішення і швидше розробляти програми;

- об'єктно-орієнтований:

  Python підтримує об'єктно-орієнтоване програмування, що дозволяє розробникам використовувати концепції класів і об'єктів;

Порівняння Python з іншими популярними мовами програмування:

- Java та C#:

Python має простіший синтаксис порівняно з Java та C#, а також менше вимог до декларації типів;

- C++:

Python менше витратний у плані розробки, але може бути менш ефективним у виконанні в порівнянні з C++, особливо для обчислювально важливих завдань;

- JavaScript:

Python є серверною мовою, тоді як JavaScript використовується для розробки клієнтської частини веб-додатків.

Де на сьогодні використовують Python:

- веб-розробка:

Python використовується для розробки веб-застосунків за допомогою фреймворків, таких як Django та Flask;

- аналіз даних та наукове програмування:

Python є популярним в аналізі даних, науковому програмуванні та статистичному моделюванні завдяки бібліотекам NumPy, Pandas, SciPy і Matplotlib;

- штучний інтелект та машинне навчання:

Python використовується для розробки та навчання моделей машинного навчання за допомогою бібліотек, таких як TensorFlow, PyTorch, Scikit-learn і Keras;

- системне адміністрування та автоматизація:

Python широко використовується для написання скриптів для автоматизації системних задач та конфігурації.

Переваги та недоліки мови Python

1) Переваги:

- простота вивчення та використання;
- велика кількість бібліотек та фреймворків;

- висока читабельність коду;
- крос-платформенність;
- активна та велика спільнота.

2) Недоліки:

- низька швидкодія порівняно з деякими іншими мовами;
- обмежена підтримка мобільної розробки;
- гранульність GIL може призводити до проблем в багатозадачних програмах.

Використання Python у задачах машинного навчання

Python є однією з основних мов програмування для машинного навчання. Велика кількість бібліотек та фреймворків дозволяє легко створювати, навчати та впроваджувати моделі машинного навчання. TensorFlow і PyTorch є основними бібліотеками для розробки глибоких нейронних мереж, а Scikit-learn надає інструменти для класичного машинного навчання. Python також використовується для обробки та аналізу даних, що є важливою частиною цілого процесу машинного навчання.

Загалом — функціонал Python найкраще підходить під дану задачу розробки, за рахунок широкої підтримки бібліотек для машинного навчання, простоти розробки та зрозумілого інтерфейсу.

PyCharm — це потужне інтегроване середовище розробки (IDE), розроблене компанією JetBrains спеціально для мови програмування Python.

Це інструмент, що відзначається своєю ефективністю та високим функціоналом.

Основні функції та особливості PyCharm:

- автоматичне завершення коду:

  PyCharm вражає своїм інтуїтивно зрозумілим автоматичним завершенням коду, що полегшує написання програм та зменшує кількість помилок;

- аналіз коду:

Інтегрована система аналізу дозволяє виявляти помилки, проводити код-рев'ю та надавати рекомендації для покращення якості коду;

- відлагодження:

Вбудований відладчик PyCharm забезпечує можливість крокування по коду та виявлення помилок під час виконання програми.

- підтримка віртуальних середовищ:

PyCharm легко інтегрується з інструментами для управління віртуальними середовищами, такими як virtualenv та Conda, що дозволяє ефективно використовувати ізольовані середовища для проектів;

- підтримка фреймворків:

Інтеграція з популярними фреймворками, такими як Django, Flask, FastAPI, OpenCV та Tensorflow робить PyCharm ідеальним вибором як для розробки веб-додатків, так і для розробки алгоритмів машинного навчання на Python;

- рефакторинг коду:

Інструменти для автоматизованого рефакторингу дозволяють ефективно змінювати структуру коду та поліпшувати його читабельність;

- розширена підтримка та інтеграція:

Окрім Python, PyCharm також надає підтримку інших мов, таких як JavaScript, HTML, CSS, SQL, що робить його універсальним для розробки різноманітних флгоритмів та додатків;

- інтеграція з системами контролю версій:

PyCharm взаємодіє з різними системами контролю версій, зокрема Git, надаючи зручний інтерфейс для відслідковування та управління версіями коду;

- підтримка Docker:

Вбудована підтримка Docker дозволяє легко конфігурувати, запускати та відлагоджувати додатки в контейнерах, спрощуючи роботу з розгортанням;

- підтримка інших інструментів:

PyCharm інтегрується з різноманітними інструментами розробки, такими як Jupyter Notebook та IPython, забезпечуючи різні можливості для налагодження та аналізу коду.

PyCharm виправдовує свою популярність завдяки своєму розширеному функціоналу, що робить його необхідним інструментом для розробників Python. Зручний інтерфейс, потужні інструменти відладки та підтримка широкого спектру технологій роблять PyCharm невід'ємною частиною середовища розробки для Python-проектів.

Дане середовище розробки було обрано, спираючись на цсі критерії:

- підтримка мови Python;
- інтуїтивно зрозумілий інтерфейс;
- можливості графічного редактору, що підходять до поставленої задачі.

Так як усі критерії задовільнено — середовищем розробки було обрано саме PyCharm 2023.

2.2 Розробка системи

2.2.1 Розробка вхідного конвеєра даних TensorFlow

Для тренування нейронної мережі нам знадобляться тренувальні зображення з бітовими масками, які будуть передаватися у конвеєр даних TensorFlow для подальшого опрацювання. На рисунку 2.1 зображено схему конвеєру даних TensorFlow.

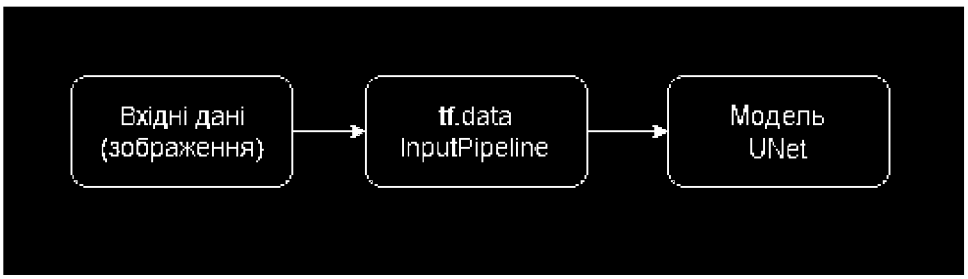


Рисунок 2.1 — Схема конвеєру даних TensorFlow

Щоб згенерувати тренувальні зображення, запишемо довільне відео з декількома різними об’єктами в одному фреймі. Це робиться для того, щоб

нейромережа, після тренування, могла працювати у реальних умовах, коли в об'єктив камери може потрапити багато різних об'єктів, що потрібно буде обробляти. На рисунку 2.2 зображено один із фреймів відео (одне із навчальних зображень).

Рисунок 2.2 — один із фреймів навчального відео

Розіб'ємо навчальне відео на зображення та накладемо бітові маски на об'єкти. Для цього зручно використовувати сервіс Supervisely, в якому є такий функціонал[https://app.supervisely.com].

На рисунку 2.3 зображено бітову маску для одного з навчальних зображень.

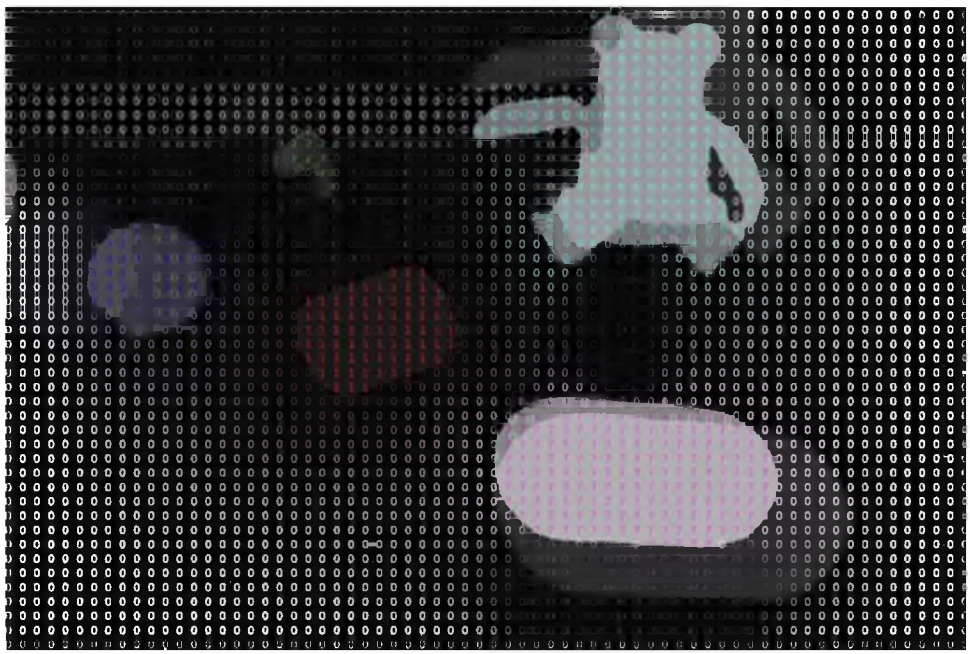


Рисунок 2.3 — бітова маска

Маючи навчальні дані, можна починати розробку. Перш за все, реалізуємо конвеєр даних TensorFlow.

Перша змінна, яку ми задаємо — “CLASSES” — це кількість всіх об’єктів, що присутні на сцені (у нас їх сім), та один додатковий клас для позначення заднього фону. Виходить — вісім.

Змінна “COLORS” — задає кольори контурів, що виділяють об’єкти на зображенні. Так на вихідному зображенні буде зрозуміло чи правильно нейронна мережа розпізнала та виділила об’єкт.

Змінна “SAMPLE_SIZE” — задає фіксований розмір вхідних зображень, які будуть використовуватися нейронною мережею.

Змінна “OUTPUT_SIZE” — задає фіксований розмір вихідних зображень, які будуть формуватися після фінальної обробки даних.

На рисунку 2.4 зображено створення змінних “CLASSES”, “COLORS”, “SAMPLE_SIZE” та “OUTPUT_SIZE”.

Рисунок 2.4 — створення змінних “CLASSES”, “COLORS”, “SAMPLE_SIZE” та “OUTPUT_SIZE”

Далі для створення конвеєру даних TensorFlow нам необхідно створити дві функції: “load_images” та “augmentate_images”.

“load_images” приймає шлях до зображень та бітових масок відповідно.

Функція отримує зображення та маски за заданим шляхом та конвертує їх у необхідний формат за допомогою спеціальних функцій TensorFlow. А саме:

1) Зчитує файл;

2) Змінює розмір;

3) Нормалізує значення;

Блок коду для обробки зображень та бітових масок зображено на рисунках 2.5 та 2.6 відповідно.

```
image = tf.io.read_file(image)
image = tf.io.decode_jpeg(image)
image = tf.image.resize(image, OUTPUT_SIZE)
image = tf.image.convert_image_dtype(image, tf.float32)
image = image / 255.0
```

Рисунок 2.5 — блок коду для обробки зображень

```
mask = tf.io.read_file(mask)
mask = tf.io.decode_png(mask)
mask = tf.image.rgb_to_grayscale(mask)
mask = tf.image.resize(mask, OUTPUT_SIZE)
mask = tf.image.convert_image_dtype(mask, tf.float32)
```

Рисунок 2.6 — блок коду для обробки бітових масок

Далі функція “load_images” розбиває зображення бітових масок з декількома об’єктами на декілька каналів. По каналу на кожен клас об’єкту на зображенні.

На рисунку 2.7 зображено блок коду для реалізації цієї функціональності.

```
masks = []

for i in range(CLASSES):
    masks.append(tf.where(tf.equal(mask, float(i)), 1.0, 0.0))

masks = tf.stack(masks, axis=2)
masks = tf.reshape(masks, OUTPUT_SIZE + (CLASSES,))
```

Рисунок 2.7 — блок коду для розбиття зображень бітових масок на декілька каналів

Таким чином, замість одномірного масиву з бітовими масками, отримали n-мірний масив, з глибиною, що дорівнює кількості класів об’єктів. Тобто сім. І в кожному шарі такого масиву будуть зберігатися тільки нулі для

фону та одиниці для об’єкта, що буде показувати чи є шуканий об’єкт на зображенні.

Приклад такого шару з одним об’єктом зображено на рисунку 2.8.

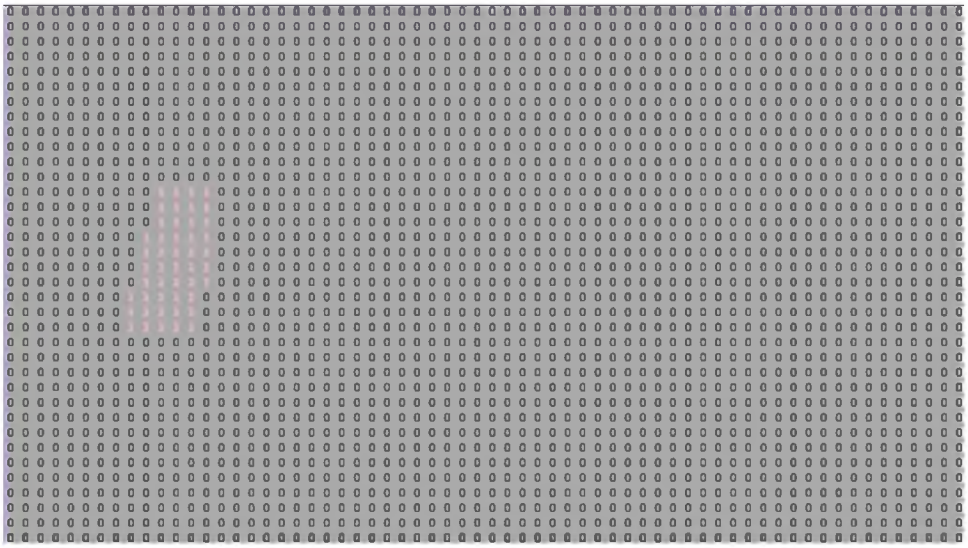


Рисунок 2.8 — приклад шару з одним об’єктом

У результаті, функція “load_images” повертає масив відформатованих зображеннь та відповідних бітових масок.

Повний код функції “load_images” зображено на рисунку 2.9.

```
def load_images(image, mask):
    image = tf.io.read_file(image)
    image = tf.io.decode_jpeg(image)
    image = tf.image.resize(image, OUTPUT_SIZE)
    image = tf.image.convert_image_dtype(image, tf.float32)
    image = image / 255.0

    mask = tf.io.read_file(mask)
    mask = tf.io.decode_png(mask)
    mask = tf.image.rgb_to_grayscale(mask)
    mask = tf.image.resize(mask, OUTPUT_SIZE)
    mask = tf.image.convert_image_dtype(mask, tf.float32)

    masks = []

    for i in range(CLASSES):
        masks.append(tf.where(tf.equal(mask, float(i)), 1.0, 0.0))

    masks = tf.stack(masks, axis=2)
    masks = tf.reshape(masks, OUTPUT_SIZE + (CLASSES,))

    return image, masks
```

Рисунок 2.9 — код функції “load_images”

Оскільки дані для тренування нейронної мережі створювалися вручну, їх об’єм не такий вже і великий. Тому необхідно збільшити кількість зображень для тренування нейронної мережі для більш точної її роботи. Для цього створимо функцію “augmentate_images”.

“augmentate_images” приймає на вхід трьоканальні RGB зображення та багатоканальні бітові маски, які ми створили у функції “load_images”.

Функція аугментує дані та збільшує їх варіативність. Тобто ми штучно збільшуємо кількість даних, що позитивно впливає на точність рорботи нейронної мережі.

Код функції “augmentate_images” зображено на рисунку 2.10

```
55  def augmentate_images(image, masks):
56      random_crop = tf.random.uniform((), 0.3, 1)
57      image = tf.image.central_crop(image, random_crop)
58      masks = tf.image.central_crop(masks, random_crop)
59
60      random_flip = tf.random.uniform((), 0, 1)
61      if random_flip >= 0.5:
62          image = tf.image.flip_left_right(image)
63          masks = tf.image.flip_left_right(masks)
64
65      image = tf.image.resize(image, SAMPLE_SIZE)
66      masks = tf.image.resize(masks, SAMPLE_SIZE)
67
68      return image, masks
```

Рисунок 2.10 — код функції “augmentate_images”

Розберемо трохи детальніше перетворення зображень, що робить дана функція.

У першому блоці застосуємо функцію “central_crop” для зображень та бітових масок. Ця функція вилучає центральний фрагмент з випадковим значенням масштабування.

Блок коду з функцією “central_crop” зображено на рисунку 2.11.

```
random_crop = tf.random.uniform((), 0.3, 1)
image = tf.image.central_crop(image, random_crop)
masks = tf.image.central_crop(masks, random_crop)
```

Рисунок 2.11 — код з функцією “central_crop”

У другому блоці коду застосуємо випадкове відображення зображень та бітових масок по горизонталі.

Другий блок коду для відображення зображень знаходиться на рисунку 2.12.

```
    random_flip = tf.random.uniform((), 0, 1)
    if random_flip >= 0.5:
        image = tf.image.flip_left_right(image)
        masks = tf.image.flip_left_right(masks)
```

Рисунок 2.12 — код для відображення зображень по горизонталі

У третьому і останньому блоці функції “augmentate_images” встановимо вихідний розмір зображкнь і бітових масок.

Після завершення роботи, функція повертає масив зображень і відповідних їм бітових масок уже у збільшеній кількості.

Третій блок коду функції “augmentate_images” зображено на рисунку 2.13.

```
    image = tf.image.resize(image, SAMPLE_SIZE)
    masks = tf.image.resize(masks, SAMPLE_SIZE)

    return image, masks
```

Рисунок 2.13 — код для останнього блоку функції “augmentate_images”

Після створення цих функцій, слід завантажити підготовлені зображення та маски з диску. Загрузка файлів з диску у змінні зображена на рисунку 2.14

```

images = sorted(glob.glob('Data/dataset/images/*.jpg'))
masks = sorted(glob.glob('Data/dataset/masks/*.png'))
```

Рисунок 2.14 — завантаження зображень та масок з диску

Далі, за допомогою функцій TensorFlow, сформуємо датасет з завантажених зображень і бітових масок.

```
74  images_dataset = tf.data.Dataset.from_tensor_slices(images)
75  masks_dataset = tf.data.Dataset.from_tensor_slices(masks)
76
77  dataset = tf.data.Dataset.zip((images_dataset, masks_dataset))
```

Рисунок 2.15 — створення датасету

Після цього використаємо створену раніше функцію “load_images” та завантажуємо дані у ОЗУ.

```
dataset = dataset.map(load_images, num_parallel_calls=tf.data.AUTOTUNE)
```

Рисунок 2.16 — виклик функції “load_images”

Збільшимо кількість даних у п’ятдесят разів копіюванням за допомогою функції TensorFlow.

```
80  dataset = dataset.repeat(50)
```

Рисунок 2.17 — збільшення кількості даних

І для всіх отриманих таким чином даних, застосуємо функцію “augmentate_images”, яку створили раніше. Тепер кожне зображення з цього набору буде унікальним.

```
81  dataset = dataset.map(augmentate_images, num_parallel_calls=tf.data.AUTOTUNE)
```

Рисунок 2.18 — виклик функції “augmentate_images”

Таким чином, ми збільшили невеликий початковий набір даних у п’ятдесят разів.

За допомогою функцій TensorFlow “take” та “skip” розділимо отриманий набір даних на навчальний та тестовий. Отримані датасети закешуємо в пам’яті.

```
106 train_dataset = dataset.take(2000).cache()
107 test_dataset = dataset.skip(2000).take(100).cache()
```

Рисунок 2.19 — створення навчального та тестового датасетів

Встановимо розмір пакету, якими будемо навчати нейронну мережу

```
train_dataset = train_dataset.batch(16)
test_dataset = test_dataset.batch(16)
```

Рисунок 2.19 — встановлення розміру пакету

На цьому створення вхідного конвеєру даних TensorFlow завершено.

2.2.2 Розробка згорткової нейронної мережі за архітектурою UNet

Архітектура UNet складається з частини, що звужується (зліва) та частини, що розширюється. Зазвичай, таку архітектуру називають енкодер-декодер. Приклад такої архітектури зображено на рисунку 2.20. Де зелений блок зліва — вхідне зображення, а червоний блок — вихідне зображення.

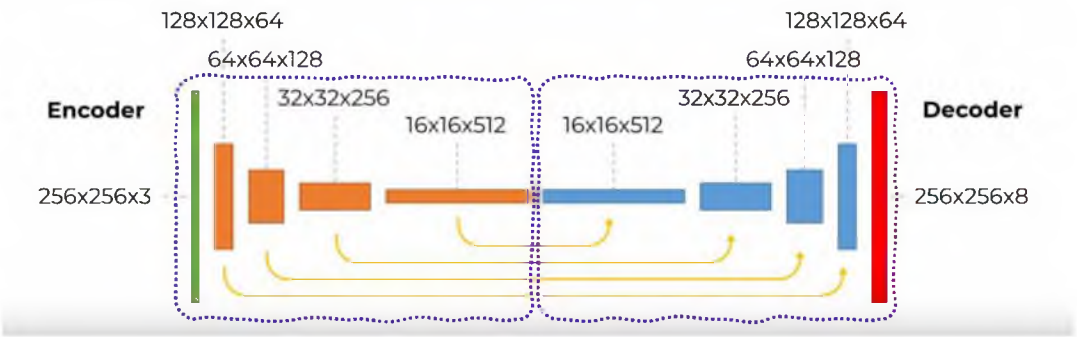


Рисунок 2.20 — схема UNet архітектури

Створимо кожний блок цієї архітектури послідовно. Першим необхідно сворити блок вхідного шару нейронної мережі. Назвемо його “input_layer”. Цей шар можна створити за допомогою функції у TensorFlow. Додатково передамо у цю функцію розмір вхідного зображення, який задавали на початку розробки. Код вхідного шару нейронної мережі зображено на рисунку 2.21.

```
1 usage
def input_layer():
    return tf.keras.layers.Input(shape=SAMPLE_SIZE + (3,))
```

Рисунок 2.21 — код вхідного шару нейронної мережі

Далі по порядку — згортковий шар (енкодер). Його також можна створити за допомогою функцій TensorFlow. Функцію для цього шару назвемо “downsample_block”. Код функції згорткового шару зображено на рисунку 2.22.

```
def downsample_block(filters, size, batch_norm=True):
    initializer = keras.initializers.GlorotNormal()

    result = keras.Sequential()

    result.add(
        keras.layers.Conv2D(filters, size, strides=2, padding='same',
                            kernel_initializer=initializer, use_bias=False))

    if batch_norm:
        result.add(keras.layers.BatchNormalization())

    result.add(keras.layers.LeakyReLU())
    return result
```

Рисунок 2.22 — код згорткового шару (енкодеру)

Всередені функції використовуємо комбінацію згорткових шарів “Conv2D”, шари пакетної нормалізації “BatchNormalization” та активаційну функцію “LeakyReLU”. А також у функції використано метод ініціалізації вагових коефіцієнтів — “GlorotNormal”.

Далі за схемою шар розгортки. Функію для нього назвемо “upsample_block”. Код шару розгортки зображено на рисунку 2.23.

```
def upsample_block(filters, size, dropout=False):
    initializer = keras.initializers.GlorotNormal()

    result = keras.Sequential()

    result.add(
        keras.layers.Conv2DTranspose(filters, size, strides=2, padding='same',
                                     kernel_initializer=initializer, use_bias=False))

    result.add(keras.layers.BatchNormalization())

    if dropout:
        result.add(keras.layers.Dropout(0.25))

    result.add(keras.layers.ReLU())
    return result
```

Рисунок 2.23 — код шару розгортки

Всередені функції “upsample_block” використовуємо комбінацію згорткових transpose-шарів “Conv2DTranspose”, шарів пакетної нормалізації “BatchNormalization”, dropout-шарів “Dropout” та та активаційну функцію “ReLU”. Також у функції використано метод ініцалізації вагових коефіцієнтів — “GlorotNormal”.

“Dropout” шари — інструмент, необхідний для уникання перенавчання нейронної мережі.

“BatchNormalization” шари — інструмент, необхідний для підвищення продуктивності мережі та зробити її роботу більш стабільною.

Згорткові transpose-шари “Conv2DTranspose” використовуються для того, щоб провести протилежну операцію згортки та для підвищення розмірності вихідних даних.

Функція “output_layer” задає вихідний шар, розмірність якого дорівнює кількості класів об’єктів на зображені та використовує сигмоїдну активаційну функцію. Код блоку вихідного шару зображено на рисунку 2.24.

```
1 usage
def output_layer(size):
    initializer = keras.initializers.GlorotNormal()
    return keras.layers.Conv2DTranspose(CLASSES, size, strides=2, padding='same',
                                        kernel_initializer=initializer, activation='sigmoid')
```

Рисунок 2.24 — код вихідного шару

Далі, за допомогою функції “downsample_block” створимо масив “downsample_stack”, який представляє енкодер.

```
inp_layer = input_layer()

downsample_stack = [
    downsample_block(64, 4, batch_norm=False),
    downsample_block(128, 4),
    downsample_block(256, 4),
    downsample_block(512, 4),
    downsample_block(512, 4),
    downsample_block(512, 4),
    downsample_block(512, 4),
]
```

Рисунок 2.25 — масив “downsample_stack”

Також створимо масив “upsample_stack” за допомогою функції “upsample_block” — це буде декодер.

```
upsample_stack = [
    upsample_block(512, 4, dropout=True),
    upsample_block(512, 4, dropout=True),
    upsample_block(512, 4, dropout=True),
    upsample_block(256, 4),
    upsample_block(128, 4),
    upsample_block(64, 4)
]

out_layer = output_layer(4)
```

Рисунок 2.26 — масив “upsample_stack”

Разом ці масиви пердставляють собою skip connections.

Skip connections в архітектурі UNet встановлюють зв'язки між відповідними шарами кодера та декодера. Це означає, що вихідні дані з верхніх рівнів кодера (менших роздільних здатностей) прямують безпосередньо до відповідних рівнів декодера (більших роздільних здатностей).

Це дозволяє мережі зберігати більше контексту та деталей зображення під час процесу відновлення. Використання skip connections допомагає подолати проблему втрати просторової інформації, що може виникнути при глибокому зменшенні розмірності зображення під час кодування та подальшого відновлення його розміру.

Далі нам необхідно з’єднати блоки енкодера та декодера та реалізувати міжшарові зв’язки за допомогою операці конкатенації. Після цього, створимо модель нейромережі за допомогою функцій TensorFlow та вкажемо там вхідні та вихідні шари. Блок коду для з’єднання енкодера та декодера, та реалізація міжшарових зв’язків представлено на рисунку 2.27 та 2.28 відповідно. Код для створення моделі зі віхдним та вихідним шарами зображено на рисунку 2.29.

```
x = inp_layer

downsample_skips = []

for block in downsample_stack:
    x = block(x)
    downsample_skips.append(x)

downsample_skips = reversed(downsample_skips[:-1])

for up_block, down_block in zip(upsample_stack, downsample_skips):
    x = up_block(x)
    x = keras.layers.Concatenate()([x, down_block])

out_layer = out_layer(x)

unet_like = keras.Model(inputs=inp_layer, outputs=out_layer)

keras.utils.plot_model(unet_like, show_shapes=True, dpi=72)
```

Рисунок 2.27 — з’єднання енкодера та декодера

```
    x = keras.layers.Concatenate()([x, down_block])
```

Рисунок 2.28 — реалізація міжшарових зв’язків

```
unet_like = keras.Model(inputs=inp_layer, outputs=out_layer)
```

Рисунок 2.29 — створення моделі нейронної мережі

Таким чином, блок - схема загальної роботи алгоритму системи буде виглядати так, як зображено на рисунку 2.29.1.

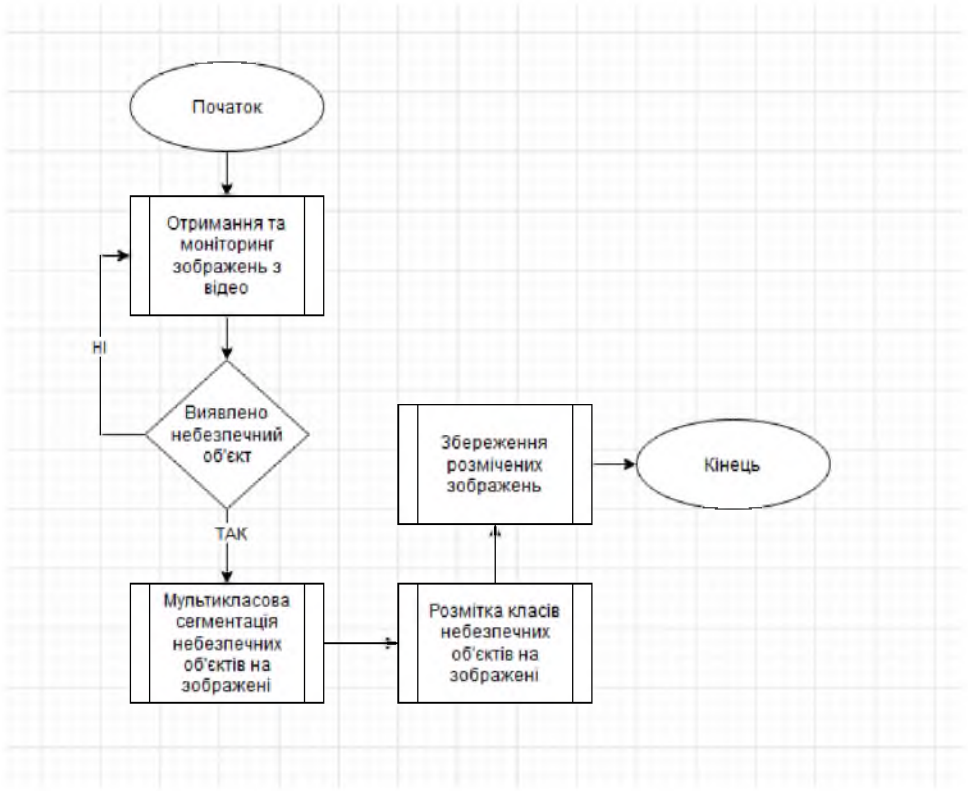


Рисунок 2.29.1 — блок - схема загальної роботи алгоритму системи

2.3 Функції втрат та метрики

Алгоритми глибокого навчання використовують методи градієнтного спуску — це метод дял знаходження мінімуму або максімуму від цільової функціїї. А цільова функція — це математичне представлення того, наскільки нейронна мережа правильно працює. Для вирішення задач сегментації, використовуються різні функції та їх комбінації. Найпопулярніші з них — Binary cross entropy (BCE) та Dice.

Binary Cross Entropy (BCE) та Dice (або Dice Coefficient) є функціями втрат, які часто використовуються у нейронних мережах для задач семантичної сегментації та розпізнавання об'єктів. Обидві функції відображають різні аспекти якості прогнозів моделі.

BCE використовується для бінарної класифікації, де кожен піксель або піксель області на зображенні класифікується як присутній (1) або відсутній (0). Функція втрат BCE вимірює різницю між прогнозованими й істинними

мітками (які мають значення 0 або 1) для кожного пікселя та обчислює середню крос-ентропію для всього зображення.

Формула BCE зображена на рисунку 2.30, де N — кількість пікселів, y — істинне значення, y^ — прогнозоване значення.

$$L_{BCE}(y, \hat{y}) = -(ylog(\hat{y}) + (1 - y)log(1 - \hat{y}))$$

Рисунок 2.30 — формула BCE

Dice Coefficient вимірює подібність між прогнозованим та істинними масками об'єктів. Формула функції втрат Dice визначається зображена на рисунку 2.31, де N — кількість пікселів, yi — істинне значення, y^ — прогнозоване значення.

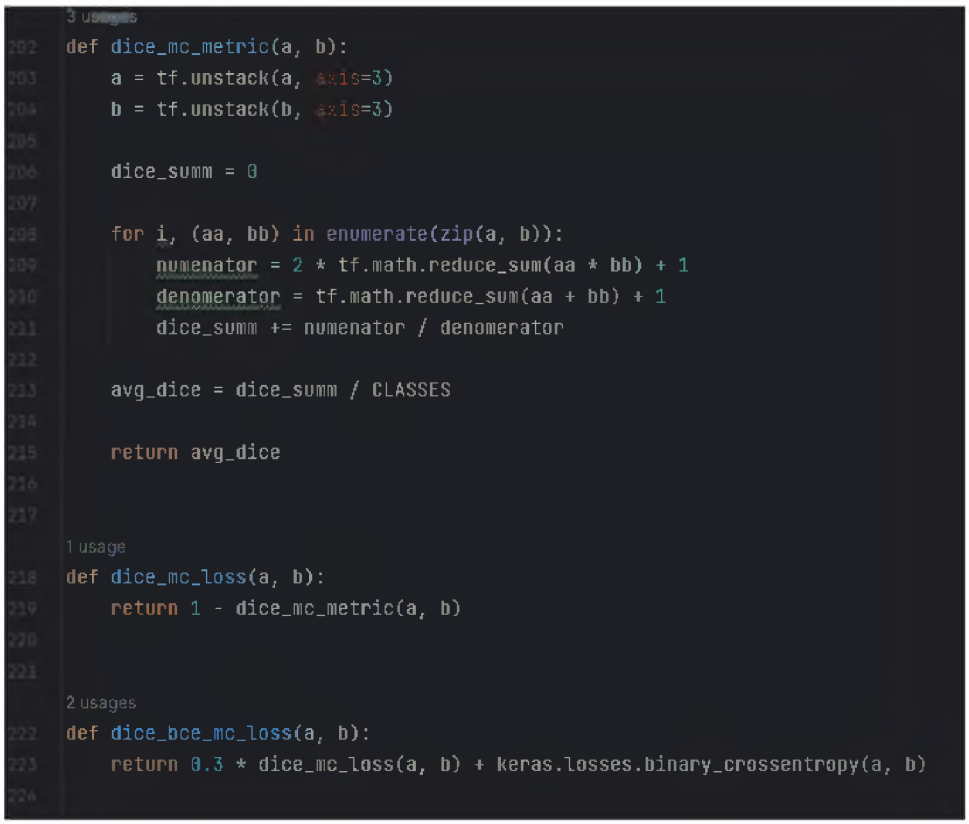
$$1 - \frac{2 * y_i * p_i + 1}{y_i + p_i + 1}$$

Рисунок 2.31 — формула Dice

Коефіцієнт приймає значення від 0 до 1, де 1 вказує на ідентичність масок.

Зазвичай, для задач семантичної та мульикласової сегментації використовують комбінації цих функцій втрат, наприклад, BCE + 0.3 * Dice, для кращої оптимізації та забезпечення якісного навчання моделі. Саме цю формулу і будемо використовувати у розрахунку функції втрат. А для розрахунку точності роботи нейронної мережі будемо використовувати коефіцієнт Dice.

На рисунку 2.32 зображено програмний код для підрахунку необхідних метрик.



```
3 usages
def dice_mc_metric(a, b):
    a = tf.unstack(a, axis=3)
    b = tf.unstack(b, axis=3)

    dice_summ = 0

    for i, (aa, bb) in enumerate(zip(a, b)):
        numenator = 2 * tf.math.reduce_sum(aa * bb) + 1
        denomerator = tf.math.reduce_sum(aa + bb) + 1
        dice_summ += numenator / denomerator

    avg_dice = dice_summ / CLASSES

    return avg_dice


1 usage
def dice_mc_loss(a, b):
    return 1 - dice_mc_metric(a, b)


2 usages
def dice_bce_mc_loss(a, b):
    return 0.3 * dice_mc_loss(a, b) + keras.losses.binary_crossentropy(a, b)
```

Рисунок 2.32 — розрахунок метрик

Тепер можна скомпілювати модель, навчити нейронну мережу та зберегти результати. Для цього використаємо вбудовані у TensorFlow функції.

У параметрах функції “fit” задамо кількість епох навчання, датасет з зображеннями та бітовими масками. Для тестування ефективності навчання, проведемо тест з декількома сценаріями: 25 епох, 10 епох, 5 епох. Також протестуємо модель з різною кількостю вхідних даних. Для цього будемо зменшувати або збільшувати кількість вхідних даних на етапі створення датасету. Таким чином, ми зможемо дізнатися яка кількість даних та епох навчання є оптимальною для дієвого навчання та подальшого вдалого розпізнавання об’єктів на зображенні.

Програмна реалізація функцій компіляції, навчання нейронної мережі та її збереження зображено на рисунках 2.33 та 2.34 відповідно.

```
weights_directory = r'C:\Users\Артем\PycharmProjects\NeuralNetwork\Data\networks\unet_like'

# Check if the directory exists, create it if not
if not os.path.exists(weights_directory):
    os.makedirs(weights_directory)

# Compile the model
unet_like.compile(optimizer='adam', loss=[dice_bce_mc_loss], metrics=[dice_mc_metric])
```

Рисунок 2.33 — компіляція моделі

```
# Train the model
history_dice = unet_like.fit(train_dataset, validation_data=test_dataset, epochs=1, initial_epoch=0)

# Save weights
weights_path = os.path.join(weights_directory, 'unet_like_weights.h5')
unet_like.save_weights(weights_path)
```

Рисунок 2.34 — навчання та збереження моделі

Після завершення навчання моделі з різними конфігураціями епох, зафіксуємо результати метрик та безпосередньо результати розпізнавання об’єктів на датасеті для тестування мережі. Метрики зобразимо графіками та у числовому вигляді. Процесс навчання нейронної мережі та числові значення метрик зобразимо на рисунках для кожної конфігурації епох та кількості вхідних даних. Також додамо у програму дві додаткові функції: перша функція буде викликатись після створення датасету. Вона буде виводити декілька зображень з датасету і їх відповідні маски. Таким чином, ми будемо впевнені, що для кожного зображення буде використана вірна бітова маска. Другу функцію будемо викликати у кінці прогами: на етапі завершення тестування мережі тестовим датасетом. Ця функція буде виводити обчислення метрик та графіки цих метрик. Таким чином, ми зможемо дізнатись чи ефективне було навчання та наскиільки точно на зараз працює модель в поточній конфігурації епох навчання та вхідних даних. Код функцій для валідації бітових масок і зображень зі створенного датасету та створення графіків і обчислення метрик зображено на рисунках 2.35 та 2.36 відповідно.

```python
images_and_masks = list(dataset.take(5))


fig, ax = plt.subplots(nrows=2, ncols=5, figsize=(15, 5), dpi=125)

for i, (image, masks) in enumerate(images_and_masks):
    ax[0, i].set_title('Image')
    ax[0, i].set_axis_off()
    ax[0, i].imshow(image)

    ax[1, i].set_title('Mask')
    ax[1, i].set_axis_off()
    ax[1, i].imshow(image / 1.5)

    for channel in range(CLASSES):
        contours = measure.find_contours(np.array(masks[:, :, channel]))
        for contour in contours:
            ax[1, i].plot(contour[:, 1], contour[:, 0], linewidth=1, color=COLORS[channel])

plt.show()
plt.close()
```

Рисунок 2.35 — валідація бітових масок та зображень з датасету

```python
def plot_training_history(history):
    plt.figure(figsize=(12, 6))

    # Plot training & validation accuracy values
    plt.subplot(1, 2, 1)
    plt.plot(history.history['dice_mc_metric'], label='Train Dice Metric')
    plt.plot(history.history['val_dice_mc_metric'], label='Validation Dice Metric')
    plt.title('Training and Validation Dice Metric')
    plt.xlabel('Epochs')
    plt.ylabel('Dice Metric')
    plt.legend()

    # Plot training & validation loss values
    plt.subplot(1, 2, 2)
    plt.plot(history.history['loss'], label='Train Loss')
    plt.plot(history.history['val_loss'], label='Validation Loss')
    plt.title('Training and Validation Loss')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.legend()

    plt.tight_layout()
    plt.show()

# Compile the model
unet_like.compile(optimizer='adam', loss=[dice_bce_mc_loss], metrics=[dice_mc_metric])

# Train the model
history_dice = unet_like.fit(train_dataset, validation_data=test_dataset, epochs=10, initial_epoch=0)

# Save weights
weights_path = os.path.join(weights_directory, 'unet_like_weights.h5')
unet_like.save_weights(weights_path)

# Load weights
unet_like.load_weights(weights_path)

# Plot training history
plot_training_history(history_dice)
```

Рисунок 2.36 — функція для побудови графіків і розрахунку метрик

2.4 Дослідження роботи моделі

2.4.1 Дослідження роботи моделі з 5 епохами навчання та кількістю даних, що 50 раз більше, ніж початкова кількість зображень.

На рисунку 2.37 та 2.38 зображено відповідні налаштування функцій.

```
dataset = dataset.map(load_images, num_parallel_calls=tf.data.AUTOTUNE)
dataset = dataset.repeat(50)
dataset = dataset.map(augmentate_images, num_parallel_calls=tf.data.AUTOTUNE)
```

Рисунок 2.37 — збільшення кількості даних у 50 разів

```
# Train the model
history_dice = unet_like.fit(train_dataset, validation_data=test_dataset, epochs=5, initial_epoch=0)
```

Рисунок 2.38 — п’ять епох навчання

Запустимо навчання моделі та протестуємо її на датасеті для тесту моедлі. Процес навчання, валідація зображень та бітових масок, метрики, графіки метрик та підсумкові зображення після обробки буде зображено на рисунках 2.39 - 2.44 відповідно.

Рисунок 2.39 — валідація вхідних зображень та бітових масок

```
125/125 [==============================] - 359s 3s/step - loss: 0.2652 - dice_mc_metric: 0.4380 - val_loss: 0.7113 - val_dice_mc_metric: 0.2500
Epoch 2/5
125/125 [==============================] - 231s 2s/step - loss: 0.0777 - dice_mc_metric: 0.8307 - val_loss: 0.3819 - val_dice_mc_metric: 0.4542
Epoch 3/5
125/125 [==============================] - 219s 2s/step - loss: 0.0257 - dice_mc_metric: 0.9436 - val_loss: 0.0771 - val_dice_mc_metric: 0.8397
Epoch 4/5
125/125 [==============================] - 213s 2s/step - loss: 0.0164 - dice_mc_metric: 0.9647 - val_loss: 0.0773 - val_dice_mc_metric: 0.8724
Epoch 5/5
125/125 [==============================] - 214s 2s/step - loss: 0.0133 - dice_mc_metric: 0.9719 - val_loss: 0.0185 - val_dice_mc_metric: 0.9579
```

Рисунок 2.40 — процесс навчання мережі з метриками за кожну епоху

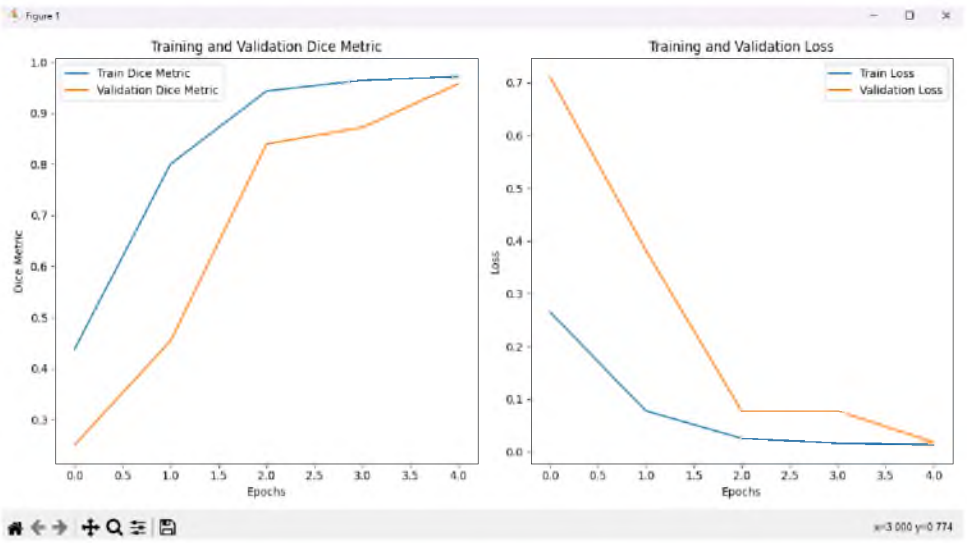


Рисунок 2.41 — графіки зміни значень метрик за кожну епоху

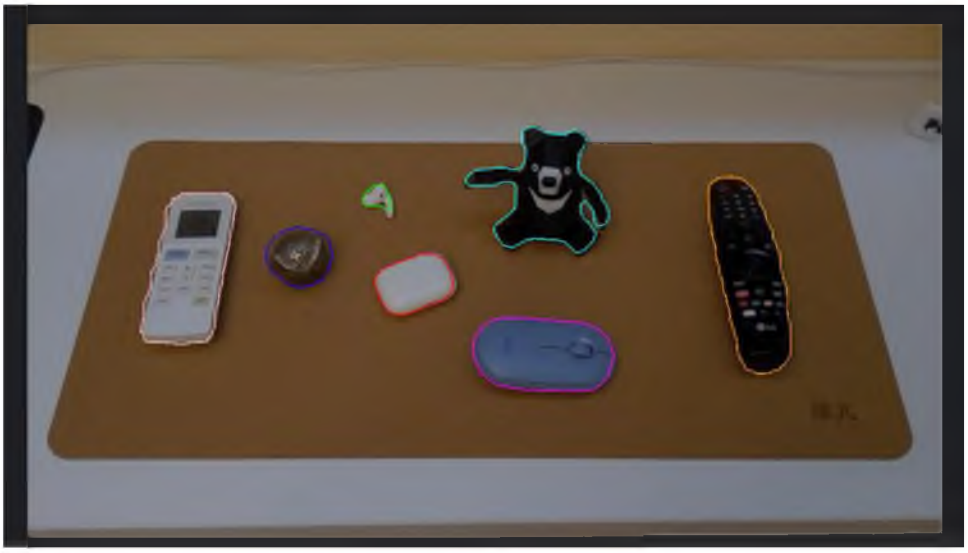
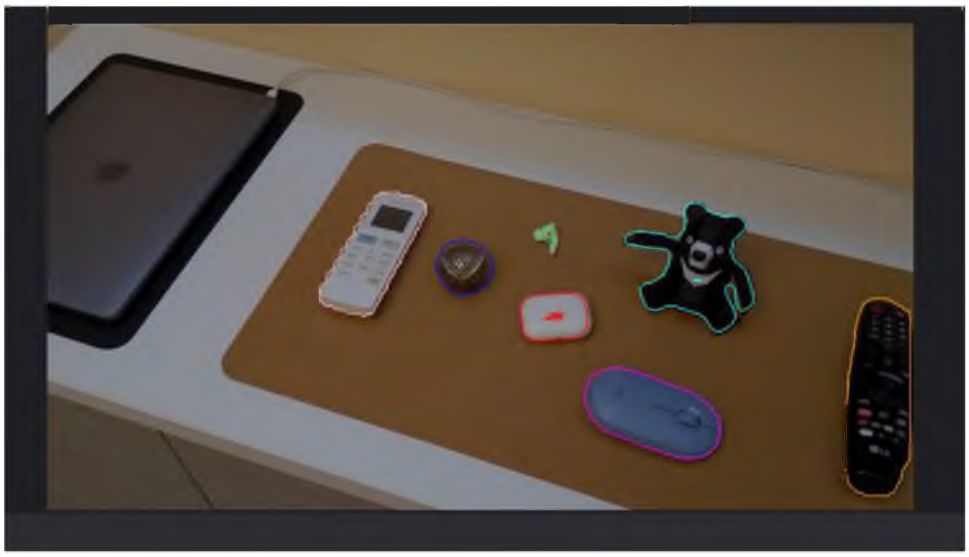


Рисунок 2.42 — результат розпізнавання об’єкту на зображенні після п’яти епох навчання



Як можа бачити за результатами метрик, після п’яти епох навчання, модель нейронної мережі демонструє гарнут очніть і досить низький показник втрат. А саме: 0.0185 втрат та 0.9579 точність.

Рисунок 2.43 — результат розпізнавання об’єкту на зображенні під кутом після п’яти епох навчання

2.4.2 Дослідження роботи моделі з десятьма епохами навчання та кількістю даних, що у п`ятдесят разів більше, ніж початкова кількість зображень.

На рисунку 2.44 та 2.45 зображено відповідні налаштування функцій.

```
dataset = dataset.map(load_images, num_parallel_calls=tf.data.AUTOTUNE)
dataset = dataset.repeat(50)
dataset = dataset.map(augmentate_images, num_parallel_calls=tf.data.AUTOTUNE)
```

Рисунок 2.44 — збільшення кількості даних у 50 разів

```
# Train the model
history_dice = unet_like.fit(train_dataset, validation_data=test_dataset, epochs=10, initial_epoch=0)
```

Рисунок 2.45 — десять епох навчання

Запустимо навчання моделі та протестуємо її на датасеті для тесту моедлі. Процес навчання, валідація зображень та бітових масок, метрики, графіки метрик та підсумкові зображення після обробки буде зображено на рисунках 2.46 - 2.50 відповідно.

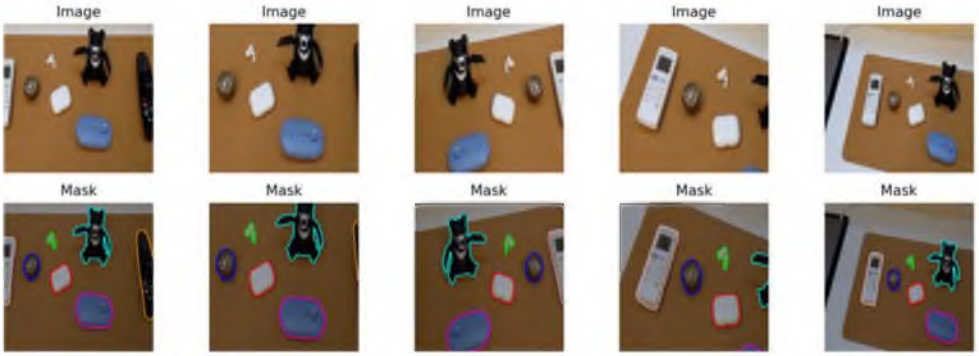


Рисунок 2.46 — валідація вхідних зображень та бітових масок

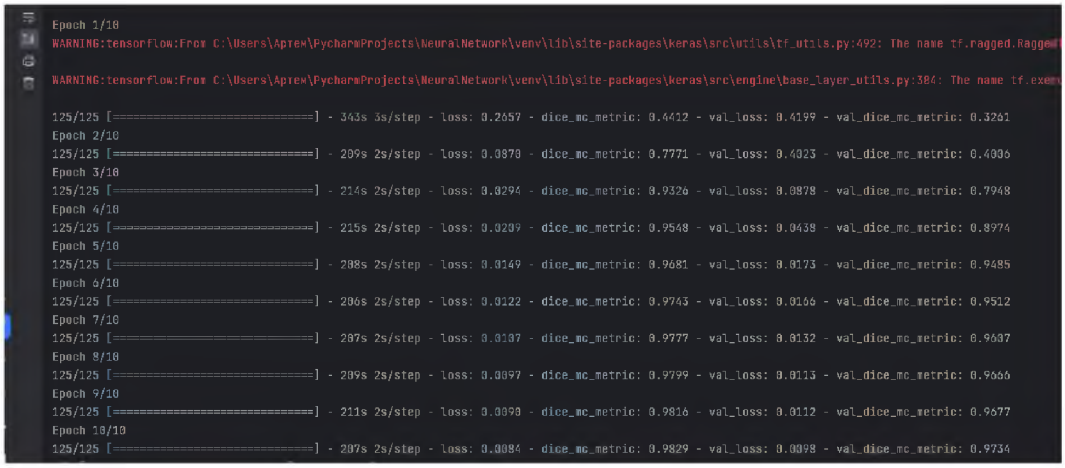


Рисунок 2.47 — процесс навчання мережі з метриками за кожну епоху

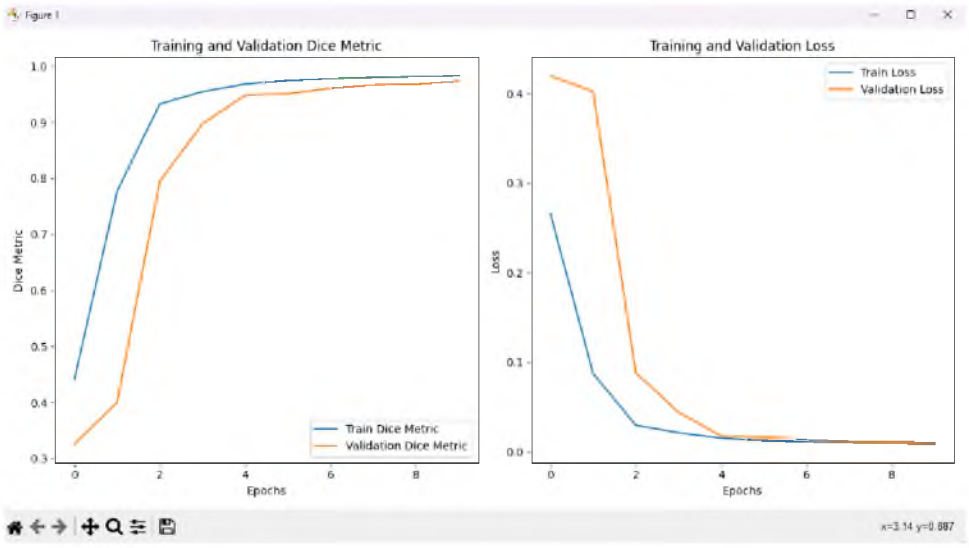


Рисунок 2.48 — графіки зміни значень метрик за кожну епоху

Рисунок 2.49 — результат розпізнавання об’єкту на зображенні після десяти епох навчання

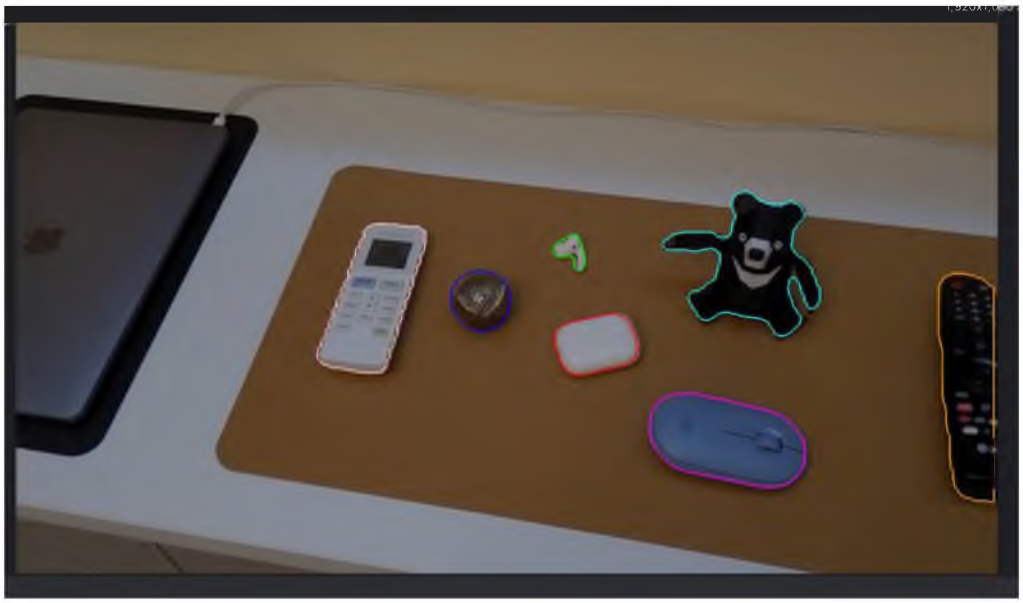


Рисунок 2.50 — результат розпізнавання об’єкту на зображенні під кутом після десяти епох навчання

Як можа бачити за результатами метрик, після десяти епох навчання, модель нейронної мережі демонструє покращення точності ще більше зменщення показника втрат у порівнянні з п’ятьма епохами навчання. А саме: 0.0098 втрат та 0.9734 точність.

2.4.3 Дослідження роботи моделі з 25 епохами навчання та кількістю даних, що 40 раз більше, ніж початкова кількість зображень.

На рисунку 2.51 та 2.52 зображено відповідні налаштування функцій.

```
79  dataset = dataset.map(load_images, num_parallel_calls=tf.data.AUTOTUNE)
80  dataset = dataset.repeat(40)
81  dataset = dataset.map(augmentate_images, num_parallel_calls=tf.data.AUTOTUNE)
82
```

Рисунок 2.51 — збільшення кількості даних у 50 разів

```
237  # Train the model
238  history_dice = unet_like.fit(train_dataset, validation_data=test_dataset, epochs=25, initial_epoch=0)
239
```

Рисунок 2.52 — двадцять п’ять епох

Запустимо навчання моделі та протестуємо її на датасеті для тесту моедлі. Процес навчання, валідація зображень та бітових масок, метрики, графіки метрик та підсумкові зображення після обробки буде зображено на рисунках 2.53 - 2.57 відповідно.

Рисунок 2.53 — валідація вхідних зображень та бітових масок

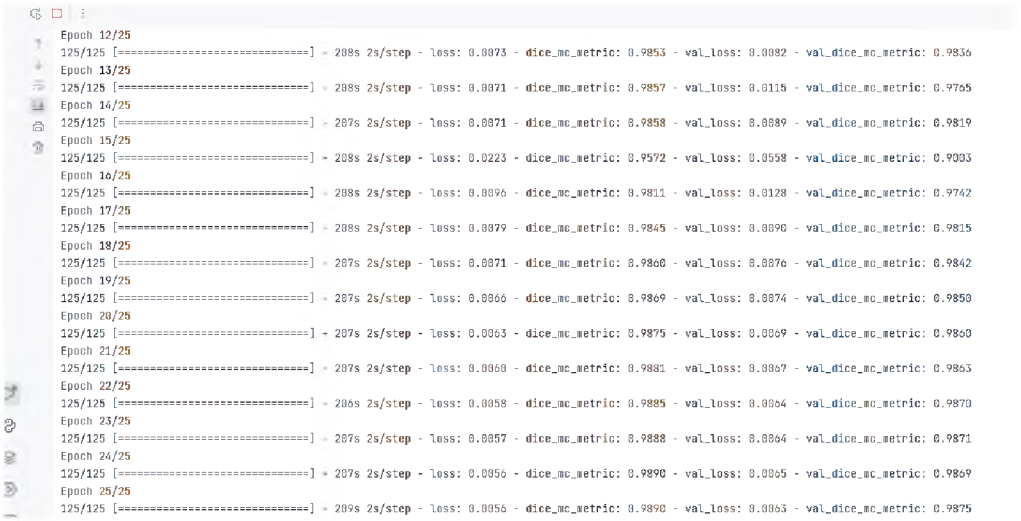


Рисунок 2.54 — процесс навчання мережі з метриками за кожну епоху

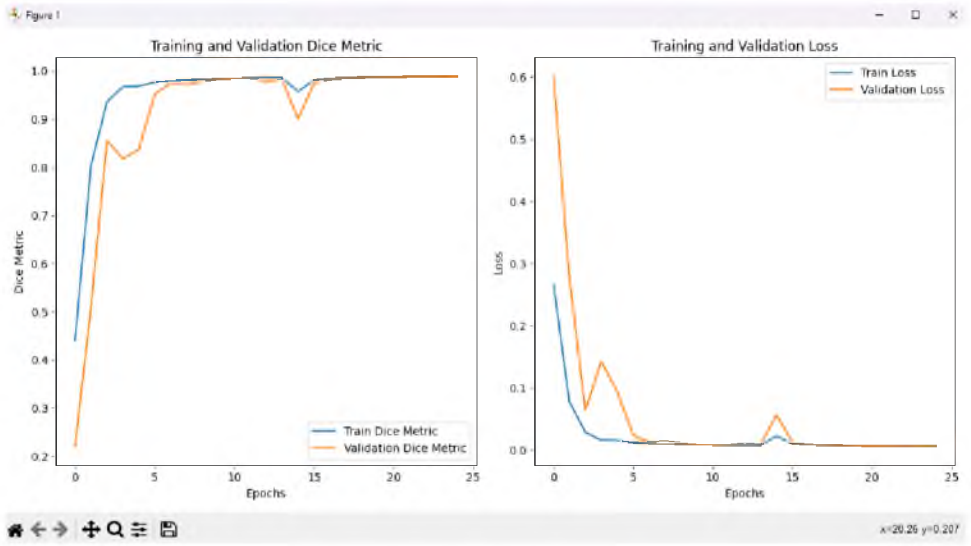


Рисунок 2.55 — графіки зміни значень метрик за кожну епоху

Рисунок 2.56 — результат розпізнавання об’єкту на зображенні після двадцяти п’яти епох навчання

Рисунок 2.57 — результат розпізнавання об’єкту на зображенні під кутом після двадцяти п’яти епох навчання

Як можа бачити за результатами метрик, після двадцяти п’яти епох навчання, модель нейронної мережі демонструє суттеве покращення показників точності та втрат у порівнянні з десятьма епохами навчання. А саме: 0.0063 втрат та 0.9875 точність.

Висновки до розділу 2

Проаналізувавши значення та графіки метрик роботи моделі нейронної мережі у різних конфігурація епох навчання, можна зробити висновок, що модель робить відчутний прогрес у точності та зменшенні втрат вже після десяти епох навчання. Але, все ж таки, чим більше навчатимемо модель - тим краще, що демонструють результати роботи моделі після двадцяти п'яти епох навчання. Едине, що треба ураховувати - це час на навчання. Як ми можемо бачити, в середньому, на кожну епоху навчання потрібно 210 секунд. Якщо взяти за мінімум двадцять п'ять епох навчання для отримвння гарних результатів роботи, то на це знадобиться (25 * 210) / 3600 = 1,458 години. Ця ситуація актуальна у випадку, якщо до моделі, наприклад, треба додати ще один об'єкт для пошуку. Дане обмеження слід ураховувати. Але ,звісно, можн овикористовувати комбіновані підходи до навчання і навчати модель поступово за декілька ітерацій, але з меншим часом, для зменшення часу простою системи виявлення загрох. На тестових зображеннях можна спостерігати, що модель успішно впоралась та без проблем виявила усі шукані об'єкти навіть після нетривалого навчання. Також досить показовим є те, що найперші епохи навчання моделі займають найбільше часу, а з ним і ресурсу сервера. Проте надалі модель починає вчитись куди швидше. З цього можна зробити висновок, що у ситуаціях, коли модель треба буде навчити розпізнавати нові об'єкти у додачу до вже вивченних, для цього необхідно буде задіяти досить потужний сервер і проводити навчання у час, коли підприємство або не працює, або тимчасово захищене іншими методами виявлення загроз на час навчання моделі.

Беручи до уваги інформацію вище, вважаю, що даний спосіб виявлення загроз підприємству на основі згорткової UNet нейронної мережі є ефективним.

РОЗДІЛ 3

ЕКОНОМІЧНЕ ОБГРУНТУВАННЯ ДОЦІЛЬНОСТІ РОЗРОБКИ

В даному розділі кваліфікаційної роботи проводиться економічне обґрунтування доцільності розробки програмного забезпечення. Зокрема, здійснюється розрахунок витрат на розробку програмного забезпечення, експлуатаційних витрат, ціни споживання проектного рішення. В заключній частині визначаються показники економічної ефективності нового програмного продукту, обґрунтовуються відповідні висновки.

Розроблене програмне забезпечення призначене для виявлення загроз підприємству на основі згорткової UNet нейронної мережі.

3.1 Розрахунок витрат

Трудомісткість розробки даної системи може бути розрахована на основі трудомісткості робіт, які виконуються.

3.1.1 Трудомісткість

Трудомісткість створення ПЗ визначається тривалістю кожної робочої операції, починаючи з складання технічного завдання і закінчуючи оформленням документації (за умови роботи одного програміста): Трудомісткість створення ПЗ визначається тривалістю кожної робочої операції, починаючи з складання технічного завдання і закінчуючи оформленням документації (за умови роботи одного програміста):

$$t = t_{тз} + t_{в} + t_{а} + t_{пр} + t_{опр} + t_{д}, \text{ годин},$$

де $t_{тз}$ – тривалість складання технічного завдання на розробку ПЗ,

де $t_{тз} = 7$ годин

$t_{в}$ – тривалість вивчення ТЗ, літературних джерел за темою тощо;

$t_{а}$ – тривалість розробки блок-схеми алгоритму;

$t_{пр}$ – тривалість програмування за готовою блок-схемою;

$t_{опр}$ – тривалість опрацювання програми на ПК;

$t_б$ – тривалість підготовки технічної документації на ПЗ.

Складові трудомісткості визначаються на підставі умовної кількості операторів у програмному продукті Q (з урахуванням можливих учточнень у процесі роботи над алгоритмом і програмою)

Умовна кількість операторів у програмі:

$$Q=q*c1+p, \text{ штук}$$

Де q – очікувана кількість операторів;

c – коефіцієнт складності програми. Коефіцієнт визначає відносну складність програми щодо типового завдання. Діапазон його зміни: 1.25…2

p – коефіцієнт корекції програми в процесі її опрацювання. Коефіцієнт визначає збільшення обсягу робіт за рахунок внесення змін в алгоритм або програму внаслідок уточнення технічного завдання. Діапазон его зміни 0,05…0,1

Очікувана кількість операторів (q): Код містить багато операторів, включаючи присвоєння, умовні конструкції, цикли, виклики функцій тощо. Усього їх приблизно 200.

Коефіцієнт складності програми (c): Код є досить складним, оскільки він включає обробку даних, машинне навчання та різні умовні конструкції. Візьмемо середнє значення: 1.5.

Коефіцієнт корекції програми (p): Припустимо, що зміни будуть помірними, та оберемо середнє значення 0.075.

За формолою (3.2) Розчитаємо кількість операторів у програмі

$$Q=200*1.5*1.075=322.5$$

Тривалість вивчення технічного завдання складається з опрацювання довідкової літератури з урахуванням уточнення ТЗ і кваліфікації програміста можливо оцінити за формолою:

$$t_в = Q * B75...85 * k, \text{ годин}$$

де В – коефіцієнт збільшення тривалості етапу внаслідок недостатнього опису завдання, В = 1,2…1,5;

k – коефіцієнт, що враховує кваліфікацію програміста і визначається стажем роботи за фахом:

- До 2 років – 0,8;
- Від 2 до 3 років 1,0;
- від 3 до 5 років 1,1…1,2;
- від 5 до 7 років - 1,3…1,4;
- понад 7 років - 1,5…1,6.

Внаслідок того що опис завдання буде дороблятися візьмемо середнє значення В = 1.3

Програміст, який зможе розробляти даний програмний код має мати кваліфакацію від 2 до 3 років, тому k = 1,0.

Розрахуємо тривалість вивчення ТЗ, літературних джерел за темою використовуючи формулу (3.3)

$t_в$= 6 годин

Тривалість розробки блок-схеми алгоритму:

$$t_a = Q20...25 * k, \text{ годин}$$

Розрахуємо тривалість розробки блок-схеми алгоритму за формулою (3.4):

t_a=14 годин

Тривалість складання програми за готовою блок-схемою:

$$t_{пр} = Q20...25 * k, \text{ годин}$$

Розрахуємо тривалість складання програми за готовою блок-схемою за формолою (3.5):

$t_{пр}$=13 годин

Тривалість опрацювання програми на ПК:

$$topр=1,5Q4...5*k, годин$$

Розрахуємо тривалість опрацювання програми на ПК за формолою (3.6):

$t_{опр}$ = 95 годин

Тривалість підготовки технічної документації на ПЗ:

$$tд=Q15...20*k+Q(15...20)*0,75, годин$$

Розрахуємо тривалість підготовки технічної документації на ПЗ (3.7):

$t_д$ = 24 годин.

Підрахуємо трудомісткість використовуючи формулу (3.1):

де $t_{тз}$ = 7 годин, $t_в$ = 6 годин, $t_а$ = 14 годин, $t_{пр}$ = 13 годин, $t_{опр}$ = 95 годин, $t_д$ = 24 годин.

$$t = 7 + 6 + 14 +13 +95 + 24 = 159 годин.$$

3.1.2 Розрахунок витрат на створення програмного засобу для виявлення загроз підприємству на основі згорткової UNet нейронної мережі

Витрати на розробку програмного засобу для виявлення загроз підприємству на основі згорткової UNet нейронної мережі $К_{пз}$ складаються з витрат на заробітну плату спеціаліста з інформаційної безпеки $З_{зп}$ і вартості витрат машинного часу, що необхідний для розробки системи захисту $З_{мч}$

$$Кпз = Ззп + Змч$$

Заробітна плата виконавця враховує основну і додаткову заробітну плату, а також відрахування на соціальне потреби (пенсійне страхування,

страхування на випадок безробіття, соціальне страхування тощо) і визначається за формулою:

$$Ззп = t * Зіб, грн$$

де t – загальна тривалість розробки системи захисту, годин;

Зіб – средньо-годинна заробітна плата спеціаліста з інформаційної безпеки з нарахуваннями, грн/годину.

Заробітна плата в місяць спеціаліста інформаційної безпеки становить 28800, відповідно щоб отримати час за годину потрібно цю суму розділити на 160 годин.

$$З_{ЗП} = 159 * 180 = 28\ 620$$

Вартість машинного часу для розробки політики безпеки інформації на ПК визначається за формулою:

$$Змч = topp * Смч+tд, грн$$

де t – трудомісткість розробки політики безпеки інформації на ПК, годин;

$C_{мч}$ – вартість 1 години машинного часу ПК, грн./година.

Вартість 1 години машинного часу ПК визначається за формулою:

$$Смч= P * t нал *Се + Фзал*НаFp + Клпз*НапзFp, грн$$

де P =– встановлена потужність ПК, кВт;

$t_{нал}$ – річний час використання ПК на рік у годинах

C_e – тариф на електричну енергію, грн/кВт година;

$Ф_{зал}$ – залишкова вартість ПК на поточний рік, грн.;

H_a – річна норма амортизації на ПК, частки одиниці;

$Н_{апз}$ — річна норма амортизації на ліцензійне програмне забезпечення, частки одиниці;

$К_{лпз}$ – вартість ліцензійного програмного забезпечення, грн.;

$F_р$ – річний фонд робочого часу (за 40-годинного робочого тижня $F_р$ = 1920).

$С_{мч}$ = 0.6 * 3 *2.64 + ((9000*0.25)/1920) + ((7000*0.2)/1920) = 6,65 грн

$З_{мч}$= 95*6.65 + 24 =656 грн

$К_{пз}$ = 656+28 620=29276 грн

Залишкова вартість ПК визначається виходячи з фактичного терміну його експлуатації як різниця між первісною вартістю та зносом за час використання.

Визначена таким чином вартість розробки політики безпеки *Крп* є частиною одноразових капітальних витрат разом з витратами на придбання і налагодження апаратури системи інформаційної безпеки.

Таким чином, капітальні (фіксовані) витрати на проектування та впровадження проектного варіанта системи інформаційної безпеки складають:

$$К = Крп + Кзпз + Кпз + Каз + Кнавч + Кн$$

де $К_{рп}$ – вартість розробки проекту інформаційної безпеки та залучення для цього зовнішніх консультантів, тис. гри. Зовнішні консультанти не наймалися ($К_{рп}$=1500)

$К_{зпз}$ — вартість закупівель ліцензійного основного й додаткового програмного забезпечення (ПЗ), тис. грн;

Таблиця 3.1 – Перелік придбаного Пз

№	Назва	Кількість	Вартість за 1 шт
1	Microsoft Windows 10	1	1050 грн

2	Microsotf visual studio	1	1000 грн
Всього: $К_{зпз}$ = 2050 грн			

$К_{рп}$ – вартість розробку системи захисту від DDoS-атак від ДДоС атак, тис. гри; ($К_{пз}$ =29276)

$К_{аз}$ – Створюється своє апаратне забезпечення яке буде продивлятися вхідний трафік та відфільтровувати його. Приблизна ціна 300грн

$К_{навч}$ — витрати на навчання технічних фахівців і обслуговуючого персоналу, тис. гри; ($К_{навч}$ =1200грн)

$К_{н}$ – витрати на встановлення обладнання та налагодження системи інформаційної безпеки, тис. гри. ($К_{н}$ = 800)

За формулою (3.12) розрахуємо капітальні витрати:

$$К =1500+2050+29276+1200+800 = 34826 \text{ грн}$$

3.1.3 Розрахунок поточних (експлуатаційних) витрат

Експлуатаційні витрати – це поточні витрати на експлуатацію та обслуговування об’єкта проектування за визначений період (наприклад, рік), що виражені у грошовій формі.

$$С = Св + Ск + Сак, тис. грн$$

Де $С_{в}$ – Витрати на Upgrade-відновлення й модернізацію системи інформаційної безпеки.

$С_{к}$ – витрати на керування системою в цілому;

$С_{ак}$ – «активність користувача», витрати викликані активністю користувачів системи інформаційної безпеки.

$С_{в}$ – орієнтовно визначимо виходячи із вагової частки статей витрат (21%) з сукупної вартості інформаційної системи

$$С_{в} = 34826 *0.21 = 7313 \text{ грн.}$$

Витрати на керування системою інформаційної безпеки ($С_{к}$) складають:

$$Ск = Сн + Са + Сз + Сєв + Сел + со + Стос, грн$$

Витрати на навчання адміністративного персоналу и кінцевих користувачів визначаються за даними організації з проведення тренінгів персоналу, курсів підвищення кваліфікації тощо (Сн=3000 грн).

Річний фонд амортизаційних відрахувань ($С_а$) визначається у відсотках від суми капітальних інвестицій за видами основних фондів і нематеріальних активів (ПЗ); Вартість купівлі ліцензійного ПЗ – 2050 грн., мінімальний срок дії користування - 2 роки

$$С_а = 2050/2 * 100 = 1025 \text{ грн/рік}$$

Річний фонд заробітної плати інженерно-технічного персоналу, що обслуговує систему інформаційної безпеки ($С_З$), складає:

$$Сз = Зосн + Здод, грн$$

Де $З_{осн}$ $З_{дод}$ – основна і додаткова заробітна плата відповідно, грн на рік.

Основна заробітна плата визначається, виходячи з місячного посадового окладу, а додаткова заробітна плата – в розмірі 8-10% від основної заробітної плати.

$$З_{дод} = 345\ 600 * 9\% = 7906 \text{ грн}$$

Основна Заробітна плата в місяць спеціаліста інформаційної безпеки становить ($З_{осн}$ = 28880 грн/місяць)

$$С_З = 345\ 600 + 27648 = 373248 \text{ грн}$$

До річного фонду заробітної плати додається єдиний внесок на загальнообов'язкове державне соціальне страхування – консолідований страховий внесок, збір якого здійснюється відповідно до класів професійного ризику виробництва, до яких віднесено платників єдиного внеску, з урахуванням видів їх економічної діяльності.

Розмір єдиного внеску на загальнообов'язкове державне соціальне страхування визначається на підставі встановленого чинним законодавством відсотка від суми основної та додаткової заробітної плати (за узгодженням з консультантом економічної частини дипломного проекту)

$$С_{єсв}= 28800 *0.22 = 6336 \text{ грн}$$

Вартість електроенергії', що споживається апаратурою системою інформаційної безпеки протягом року ($С_{ел}$), визначається за формулою:

$$Сел = P * FP * Це, грн$$

де P – встановлена потужність апаратури інформаційної безпеки, кВт;

P = 0,6 * 4 кВТ + (1 серве + 1 комп) * 0,7 Квт = 4 кВт

$F_р$ – річний фонд робочого часу системи інформаційної безпеки (працює кожен день с 9:00 до 18:00 10 годин на добу);

FP = 365 днів * 24 годин * (сервер та комп'ютер) + 1920 * 2 комп'ютера = 21 360 годин.

Це – тариф на електроенергію, грн/кВт-годин. (Це = 2.64)

$С_{ел}$ = 4 *2.64 * 21 360 =225 561 грн

Витрати на залучення сторонніх *організацій* для виконання деяких видів обслуговування, навчання та сертифікацію обслуговуючого персоналу визначаються за даними організації. ($С_о$ = 0)

Витрати на технічне й організаційне адміністрування та сервіс системи інформаційної безпеки ($С_{тос}$) визначаються за даними організації або у відсотках від вартості капітальних витрат (1-3%).

K = 34826 грн

$С_{тос}$ = 34826 * 2 % = 697 грн

Витрати, викликані активністю користувачів системи інформаційної безпеки ($С_{ак}$) можна орієнтовно визначити, користуючись даними табл. 1 про вагові частки статей витрат у сукупній вартості системи інформаційної безпеки.

$С_{ак}$ =34826 * 46% = 16019 грн

Розрахуємо витрати на керування системою інформаційної безпеки за формулою (3.14):

$С_к$ = 3000 + 1025 + 373248+ 6336 + 225 561 + 0 + 697 = 609 867 грн

У кожному конкретному випадку можуть бути враховані й інші види поточних витрат, що визначаються специфікою експлуатації проектованої системи інформаційної безпеки.

3.2 Оцінка Величини збитку

Для розрахунку вартості збитку необхідно застосувати спрощену модель оцінки

Необхідні такі дані для розрахунку:

$t_п$ - час простою вузла або сегмента корпоративної мережі внаслідок атаки, 4 годин;

$t_в$ - час відновлення після атаки персоналом, що обслуговує корпоративну мережу, 3 годин;

$t_{ви}$ - час повторного введення загубленої інформації співробітника атакованого вузла або сегмента корпоративної мережі, 3 годин

$З_с$ - заробітна плата співробітників атакованого вузла або сегмента корпоративної мережі, 20000 грн на місяць

$З_о$ - заробітна плата обслуговуючого персоналу (адміністраторів та ін.) 28000 грн на місяць

$Ч_о$ - чисельність обслуговуючого персоналу (адміністраторів та ін. осіб) 1 особи

$Ч_с$ - чисельність співробітників атакованого вузла або сегменту корпоративної мережі, 3 осіб

О - обсяг продажів атакованого вузла або сегмента корпоративної мережі, 650 000 тис. грн у рік;

$П_{зч}$ - вартість заміни встаткування або запасеих частин, 3000 грн;

I – число атакованих вузлів або сегментів корпоративної мережі 2;

N – середнє число атак на рік. 50

Упущена вигода від простою атакованого вузла або сегмента корпоративної мережі становить:

$$U = Пп + Пв + V$$

де $П_п$ – оплачувані втрати робочого часу та простої співробітників атакованого вузла або сегмента корпоративної мережі, грн;

$П_в$ – вартість відновлення працездатності вузла або сегмента корпоративної мережі (переустановлення системи, зміна конфігурації та ін.), грн'

V – втрати від зниження обсягу продажів за час простою атакованого вузла або сегмента корпоративної мережі, грн.

Втрати від зниження продуктивності співробітників атакованого вузла або сегмента корпоративної мережі являють собою втрати їхньої заробітної плати (оплата непродуктивної праці) за час простою внаслідок атаки:

$$Пп = \sum ЗсF*tп, грн$$

де F – місячний фонд робочого часу (при 40-а годинному робочому тижні становить 176 ч).

$З_с$ – заробітна плата співробітників атакованого вузла або сегмента корпоративної мережі, 20000 грн на місяць;

tп — час простою вузла або сегмента корпоративної мережі внаслідок атаки, 4 годин;

Пп= ((20000/176)*3)*4= 1363 грн

Витрати на відновлення працездатності вузла або сегмента корпоративної мережі включають кілька складових:

$$Пв = Пви + Ппв + Пзч$$

де $П_{ви}$ – витрати на повторне уведення інформації, грн;

$П_{пв}$ – витрати на відновлення вузла або сегмента корпоративної мережі,

$П_{зч}$ – вартість заміни устаткування або запасних частин, грн.

Витрати на повторне введення інформації $П_{ви}$ розраховуються виходячи з розміру заробітної плати співробітників атакованого вузла або сегмента корпоративної мережі $З_с$, які зайняті повторним введенням втраченої інформації, з урахуванням необхідного для цього часу $t_{ви}$: ($t_{ви} = 2$)

$$Пви = \sum ЗсF^* t_{ви}$$

$$Пви = ((20000/176)*3)*3 = 1022 \text{ грн}$$

Витрати на відновлення вузла або сегмента корпоративної мережі Ппв визначаються часом відновлення після атаки $t_В$ і розміром середньогодинної заробітної плати обслуговуючого персоналу (адміністраторів):

$$Ппв = \sum ЗоF * tв$$

$$П_{пв} = (28000/176)*3 = 477 \text{ грн}$$

$$П_в = 1022 + 477 + 3000 = 4499 \text{ грн}$$

Втрати від зниження очікуваного обсягу продажів за час простою атакованого вузла або сегмента корпоративної мережі визначаються виходячи із середньогодинного обсягу продажів і сумарного часу простою атакованого вузла або сегмента корпоративної мережі: (О=750 000)

$$V=OFr*(tп+tв+tви)$$

Де Fr – річний фонд часу роботи організації (52 робочих тижні, 5-ти денний робочий, тиждень 8-ми годинний робочий день) становить близько 2080 ч.

$$V = (650000/2080) *(4+3+3) = 3125 \text{ грн}$$

Упущена вигода від простою атакованого вузла або сегмента мережі становить:

$$U = 1363 + 4499 + 3125 = 8993 \text{ грн}$$

Таким чином загальний збиток від атаки на вузол або сегмент

корпоративноїмережі організації складе:

$$B = \sum i \sum nU$$

Де і – число атакованих вузлів, 2 комп'ютера;
N – середнє число атак на рік 30 рази;

$$B = 8993 * 2 * 50 = 899\,370 грн$$

Загальний ефект від впровадження системи інформаційної безпеки визначається з урахуванням ризиків порушення інформаційної безпеки і становить:

$$E = B*R - C$$

де B – загальний збиток від атаки на вузол або сегмент корпоративної мережі, тис. гри;

R – очікувана імовірність атаки на вузол або сегмент корпоративної мережі, 0,7 частки одиниці;

C – щорічні витрати на експлуатацію системи інформаційної безпеки, тис. гри.

$$E = 899\,370 * 0.7 - 609\,867 = 19643 грн$$

3.3 Визначення та аналіз показників економічної ефективності

Показник сукупноївартості володіння (ТСО) використовується, якщо величину відверненого збитку від атаки на вузол або сегмент корпоративної мережі важко або неможливо визначити у вартісній формі.

У цьому випадку необхідно порівняти сукупну вартість володіння, розраховану для двох варіантів проектного рішення щодо створення або удосконалення системи інформаційної безпеки, і вибрати варіант із найменшою з них.

Коефіцієнт повернення інвестицій ROSI показує, скільки гривень додаткового прибутку приносить одна гривня капітальних інвестицій на впровадження системи інформаційної безпеки.

Щодо до інформаційної безпеки говорять не про прибуток, а про запобігання можливих втрат від атаки на сегмент або вузол корпоративної мережі, а отже:

$$ROSI = EK, \text{ частка одиниці}$$

де Е – загальний ефект від впровадження системи інформаційної безпеки (розділ 3.2 методичних вказівок, формула 3.8), 19643 грн;

К – капітальні інвестиції за варіантами, що забезпечили цей ефект, тис. гри.

Якщо порівнюється два варіанти системи інформаційної безпеки, то обирається варіант з більшим значенням ROSI.

$$Rosi = 19643/34826 = 0.55$$

Для остаточної оцінки варіантів і вибору найбільш ефективного з них необхідно порівняти розрахункове значення ROSI з бажаним значенням показника ефективності Ен.

Проект системи інформаційної безпеки визнається доцільним за умови

$$ROSI > EH$$

При $ROSI<E_H$ варіант є збитковим і більш економічним визнається відмова від його реалізації.

Для вибраного варіанта визначається розрахунковий строк окупності капітальних інвестицій T_p.

Термін окупності капітальних інвестицій T_p показує, за скільки років капітальні інвестиції окупляться за рахунок загального ефекту від впровадження системи інформаційної безпеки:

$$To = KE = 1ROSI \text{ років}$$

Якщо варіанти економічно рівноцінні, то приймається варіант, що забезпечує більш високу надійність, поліпшення умов праці.

$$T_o = 34826/19643 = 1{,}81 \text{ років}$$

Висновки до розділу 3

Згідно з наведеними розрахунками, можна зробити висновок, що розробка засобу для виявлення загроз підприємству на основі згорткової нейронної мережі за архітектирою UNet є доцільною.

Капітальні витрати, які складають 34826 грн., дозволяють отримати ефект величиною 19643 грн. Відповідно до тотриманих значень показників екеномічної ефективності, можна зазначити, що запропонований підхід дозволить отримувати 0,55 економічного ефекту на 1 грн. Капітальних витрат (коефіцієнт повернення інвестицій ROSI складає 0,55 грн.). Термін окупності при цьому буде складати 1,81 років.

ВИСНОВКИ

Мета роботи полягла в створенні власної програмної реалізації системи для загроз підприємству на основі згорткових нейронних мереж.

Як показало тестування навченної моделі нейронної мережі за UNet архітектурою, модель працює досить точно і з низьким відсотком втрат. Всього протестували модель у трьох різних конфігураціях епох навчання та кількості вхідних даних. Перша конфігурація — п'ять епох навчання зі збільшенням кількості вхідних даних у п'ятдесят разів. Навіть з такою конфігурацією модель показувала непогану точність та низький рівень втрат, а саме: 0.0185 втрат та 0.9579 точність. Друга конфігурація вже налічувала десять епох навчання та ту ж кількість вхідних даних, збільшених у п'ятесят разів та мала такі показники: 0.0098 втрат та 0.9734 точність. Остання конфігурація — це двадцять п'ять епох навчання та збільшені у сорок разів вхідні дані. У цій конфігурації вдалось перевірити наскільки зменшення кількості вхідних даних може вплинути на кінцевий результат роботи моделі та чи є сенс навчати модель великою кількістю епох. Дана конфігурація мала такі результати: 0.0063 втрат та 0.9875 точність. Як ми бачимо, показники покращились, але на навчання моделі у кількості двадцяти п'яти епох потрібно майже втричі більше часу. Можна відмітити, що невелике зменшення кількості вхідних даних не сильно впливає на кінцевий результат. Це навіть не відчутно на преших епохах навчання. Тобто, розроблена моделель є досить гнучкою та дає можлиість балансувати між часом, що потрібно витратити на навчання моделі та необхідною кількості вхідних даних, при цьому зберігаючи гарні показники точності та низький процент втрат.

Модель успішно знайшла усі шукані об'єкти на тестових зображеннях, тому, враховуючи дані метрик та фактичні результати роботи, можна зробити висновок, що модель, створена у ході виконанняя спеціальної частини кваліфікаційної роботи, є досить ефективною.

ПЕРЕЛІК ПОСИЛАНЬ

1. Що таке нейронна мережа? [Електронний ресурс] – Режим доступу до ресурсу: https://termin.in.ua/neyromerezha/#Vidminnosti_miz_biologicnimi_ta_stucnimi_nejronnimi_merezami.
2. Комп'ютерний зір та його застосування [Електронний ресурс] – Режим доступу до ресурсу: https://uk.theastrologypage.com/computer-vision.
3. Multi-class Image Segmentation with Unet [Електронний ресурс] – Режим доступу до ресурсу: https://kiansoon.medium.com/semantic-segmentation-is-the-task-of-partitioning-an-image-into-multiple-segments-based-on-the-356a5582370e.
4. What is computer vision? [Електронний ресурс] – Режим доступу до ресурсу: https://www.ibm.com/topics/computer-vision.
5. Історія Python [Електронний ресурс] – Режим доступу до ресурсу: https://acode.com.ua/intro-python/.
6. Застосування Python [Електронний ресурс] – Режим доступу до ресурсу: https://dan-it.com.ua/uk/blog/python-chto-jeto-za-jazyk-programmirovanija-i-gde-ego-ispolzujut/.
7. Сервіс Supervisely [Електронний ресурс] – Режим доступу до ресурсу: https://app.supervisely.com/ecosystem/search?q=.
8. Штучні нейронні мережі [Електронний ресурс] – Режим доступу до ресурсу: https://learn.ztu.edu.ua/pluginfile.php/176771/mod_resource/content/1/%D0%A8%D0%86_%D0%9A%D0%91_%D0%9B-3_%D0%9D%D0%B5%D0%B9%D1%80%D0%BE%D0%BC%D0%B5%D1%80%D0%B5%D0%B6%D0%B8.pdf.
9. Відмінності між біологічними та штучними нейронними мережами [Електронний ресурс] – Режим доступу до ресурсу:

https://termin.in.ua/neyromerezha/#Vidminnosti_miz_biologicnimi_ta_stucnimi_nejronnimi_merezami.

10. What is computer vision? [Електронний ресурс] – Режим доступу до ресурсу: https://www.ibm.com/topics/computer-vision.
11. Accurate multi-class image segmentation using weak continuity constraints and neutrosophic set [Електронний ресурс] – Режим доступу до ресурсу: https://www.sciencedirect.com/science/article/abs/pii/S1568494621006803#preview-section-abstract.
12. Що таке комп'ютерний зір? [Електронний ресурс] – Режим доступу до ресурсу: https://www.unite.ai/uk/%D1%89%D0%BE-%D1%82%D0%B0%D0%BA%D0%B5-%D0%BA%D0%BE%D0%BC%D0%BF%27%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%B8%D0%B9-%D0%B7%D1%96%D1%80/.
13. Комп'ютерний зір – це майбутнє автоматизації [Електронний ресурс] – Режим доступу до ресурсу: https://www.zaptest.com/uk/%D0%BA%D0%BE%D0%BC%D0%BF%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%B8%D0%B9-%D0%B7%D1%96%D1%80-%D1%86%D0%B5-%D0%BC%D0%B0%D0%B9%D0%B1%D1%83%D1%82%D0%BD%D1%94-%D0%B0%D0%B2%D1%82%D0%BE%D0%BC.
14. How to Use Kaggle [Електронний ресурс] – Режим доступу до ресурсу: https://www.kaggle.com/docs/models.
15. Datasets with Kaggle [Електронний ресурс] – Режим доступу до ресурсу: https://www.kaggle.com/docs/datasets.
16. SegNet for Image Segmentation [Електронний ресурс] – Режим доступу до ресурсу: https://arxiv.org/pdf/1511.00561.pdf.
17. DeepLab: Semantic Image Segmentation with Deep Convolutional Nets [Електронний ресурс] – Режим доступу до ресурсу: https://arxiv.org/pdf/1606.00915.pdf.

18. A survey of loss functions for semantic segmentation [Електронний ресурс] – Режим доступу до ресурсу: https://arxiv.org/pdf/2006.14822.pdf.
19. U-Net [Електронний ресурс] – Режим доступу до ресурсу: https://arxiv.org/pdf/1505.04597.pdf.
20. Keras: The high-level API for TensorFlow [Електронний ресурс] – Режим доступу до ресурсу: https://www.tensorflow.org/guide/keras.
21. tf.data: Build TensorFlow input pipelines [Електронний ресурс] – Режим доступу до ресурсу: https://www.tensorflow.org/guide/data.
22. https://www.tensorflow.org/guide/checkpoint
23. TensorFlow Datasets: a collection of ready-to-use datasets [Електронний ресурс] – Режим доступу до ресурсу: https://www.tensorflow.org/datasets.
24. Random Forest segmentation [Електронний ресурс] – Режим доступу до ресурсу: https://www.researchgate.net/figure/Random-Forest-segmentation-Using-a-ab-space-left-a-random-forest-classifier-is_fig5_333719782.
25. FCN architecture [Електронний ресурс] – Режим доступу до ресурсу: https://www.researchgate.net/figure/Fully-Convolutional-Networks-FCN-architecture_fig1_326916801.
26. Datasets with Kaggle [Електронний ресурс] – Режим доступу до ресурсу: https://www.kaggle.com/docs/datasets.
27. Models with Kaggle [Електронний ресурс] – Режим доступу до ресурсу: https://www.kaggle.com/docs/models.
28. DeepLab architecture [Електронний ресурс] – Режим доступу до ресурсу: ttps://www.researchgate.net/figure/Structure-diagram-of-DeepLab-V3_fig3_346566581.

ДОДАТОК А. Відомість матеріалів кваліфікаційної роботи

№	Формат	Найменування	Кількість листів	Примітки
Документація				
1	А4	Реферат	1	
2	А4	Список умовних скорочень	1	
3	А4	Зміст	2	
4	А4	Вступ	2	
5	А4	Аналіз предметної області і постановка задачі	19	
6	А4	Проектування і створення власної реалізації	30	
7	А4	Економічне обґрунтування доцільності розробки	15	
8	А4	Висновки	2	
9	А4	Перелік посилань	3	
10	А4	Додаток А	1	
11	А4	Додаток Б	1	
12	А4	Додаток В	1	
13	А4	Додаток Г	1	

ДОДАТОК Б. Перелік документів на фізичному носії

1 Явтушенко_АО_125м-22-2_пз.docx

2 Явтушенко_АО_125м-22-2_пз.pdf

3 Явтушенко_АО_125м-22-2_дм.pptx

4 Явтушенко_АО_125м-22-2_пз.pdf.p7s

5 Програма

ДОДАТОК В. Відгук на кваліфікаційну роботу

студента групи 125м-22-2

Явтушенко Артема Олексійовича

на тему

«Виявлення загроз інформаційній безпеці підприємства на основі згорткових нейронних мереж»

Пояснювальна записка складається зі вступу, трьох розділів і висновків, розташованих на 77 сторінках.

Мета роботи є актуальною, оскільки вона спрямована на дослідження методів виявлення загроз на підприємствах за допомогою згорткових нейронних мереж.

При виконанні роботи автор продемонстрував добрий рівень теоретичних знань і практичних навичок. На основі аналізу архітектур згорткових нейронних мереж та задач мультикласової сегментації в ній сформульовані задачі, вирішенню яких присвячений спеціальний розділ. У ньому було запропоновано метод виявлення загроз на підприємствах за допомогою згорткової нейронної мережі за UNet архітектурою.

Практична цінність роботи полягає у тому, що представлений метод захисту є повноцінною можливістю уникнення загроз на підприємстві.

Рівень запозичень у кваліфікаційній роботі не перевищує вимог «Положення про систему виявлення та запобігання плагіату».

В цілому робота задовольняє усім вимогам, а її автор Явтушенко А. О. заслуговує на оцінку « » та присвоєння кваліфікації «Магістр з кібербезпеки» за спеціальністю 125 Кібербезпека.

ДОДАТОК Г. Відгук керівника економічного розділу

Економічний розділ виконаний відповідно до вимог, які ставляться до кваліфікаційних робіт, та заслуговує на оцінку 90 б. («відмінно»).

Керівник розділу _____ доц. Пілова Д.П.

(підпис) (ініціали, прізвище)