МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Чорноморський національний університет
імені Петра Могили
Факультет комп'ютерних наук
Кафедра інтелектуальних інформаційних систем

ДОПУЩЕНО ДО ЗАХИСТУ
В.о. завідувача кафедри інтелектуальних інформаційних систем, канд. техн. наук, доцент
_____Є. В. Сіденко
«____»_____202_ р.

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

ДОСЛІДЖЕННЯ НЕЙРОННИХ МЕРЕЖ ДЛЯ РОЗПІЗНАВАННЯ ВІЙСЬКОВОЇ ТЕХНІКИ НА СУПУТНИКОВИХ ЗНІМКАХ

Спеціальність 122 «Комп'ютерні науки»

122 – МКР – 601.1710213

Виконав студент 6-го курсу, групи 601
_____ ***О. П. Ковалів***
«15» лютого 2023 р.

Керівник: канд. техн. наук, доцент
_____***Є. В. Сіденко***
«15» лютого 2023 р.

Миколаїв – 2023

Чорноморський національний університет ім. Петра Могили
Факультет комп'ютерних наук
Кафедра інтелектуальних інформаційних систем

Освітньо-кваліфікаційний рівень **магістр**
Галузь знань **12 «Інформаційні технології»**
(шифр і назва)
Спеціальність **122 «Комп'ютерні науки»**
(шифр і назва)

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри інтелектуальних інформаційних систем, канд. техн. наук, доцент
_____ Є. В. Сіденко
«____» _____20 р.

ЗАВДАННЯ
на магістерську кваліфікаційну роботу
Коваліву Олександру Павловичу
(прізвище, ім'я, по батькові)

1. Тема магістерської кваліфікаційної роботи «Дослідження нейронних мереж для розпізнавання військової техніки на супутникових знімках».
Керівник роботи Сіденко Євген Вікторович, канд. техн. наук, доцент.
Затв. наказом Ректора ЧНУ ім. Петра Могили від «03» листопада 2022 р. № 199
2. Строк подання студентом роботи 15 лютого 2023 р.
3. Вхідні (початкові) дані до роботи: архітектури нейронних мереж XCeption та VGG; критерії для оцінювання результатів навчання нейронних мереж. Очікуваний результат: порівняння результатів навчання різних архітектур нейронних мереж.
4. Зміст пояснювальної записки (перелік питань, які потрібно розглянути):
   - аналіз сучасного стан задачі розпізнавання об'єктів, зокрема військової техніки на супутникових знімках;
   - дослідження існуючих технологій та засобів для розпізнавання військової техніки на супутникових знімках; реалізація моделі нейронних мереж для виявлення військової техніки;;

- дослідження показників навчання моделей нейронних мереж для розпізнавання та виявлення об’єктів на супутникових і аерофотознімках.

5. Перелік графічного матеріалу: презентація.

6. Завдання до спеціальної частини:
Забезпечення вимог охорони праці у приміщенні серверної кімнати закладу ресторанного типу

7. Консультанти:

Розділ	**Прізвище, ініціали та посада консультанта**	**Підпис**
Спеціальна частина з охорони праці		
Методична частина		

Керівник роботи __канд. техн. наук, доцент. Сіденко Є. В._____
(наук. ступінь, вчене звання, прізвище та ініціали)

_____
(підпис)

Завдання прийнято до виконання ___Ковалів О. П._____
(прізвище та ініціали)

_____
(підпис)

Дата видачі завдання «__07__» _____листопада____ 2022 р.

КАЛЕНДАРНИЙ ПЛАН
Виконання магістерської кваліфікаційної роботи

Тема: Дослідження нейронних мереж для розпізнавання військової техніки на супутникових знімках

№	Найменування роботи	Початок	Закінчення	Примітки
1	Визначення керівника і теми МКР. Подання заяви на затвердження теми МКР	01.09.2022	20.10.2022	Виконано
2	Отримання завдання на виконання МКР	21.10.2022	10.11.2022	Виконано
3	Складання календарного плану на період виконання МКР	11.11.2022	15.11.2022	Виконано
4	Огляд літератури за темою дослідження	16.11.2022	27.11.2022	Виконано
5	Проходження переддипломної практики, збір та аналіз матеріалів до МКР	28.11.2022	18.12.2022	Виконано
6	Аналіз предметної області та розробка технічного завдання. Моделювання результатів	19.12.2022	12.01.2023	Виконано
7	Опис фахової частини МКР, зокрема дослідження публікацій щодо розпізнавання військової техніки на супутникових знімках, огляд існуючих архітектур штучних нейронних мереж для вирішення поставленої задачі, реалізація обраних технологій з аналізом отриманих результатів	13.01.2023	25.01.2023	Виконано
8	Розробка спеціальної частини з охорони праці та методичної частини	26.01.2023	02.02.2023	Виконано
9	Попередній захист МКР на засіданні комісії кафедри	03.02.2023	03.02.2023	Виконано
10	Коригування роботи за результатами попереднього захисту	04.02.2023	06.02.2023	Виконано
11	Остаточне оформлення пояснювальної записки та слайдів доповіді для захисту	07.02.2023	09.02.2023	Виконано
12	Подання МКР рецензенту	09.02.2023	10.02.2023	Виконано
13	Рецензування МКР	11.02.2023	12.02.2023	Виконано
14	Подання МКР, її електронної копії та інших	15.02.2023	16.02.2023	Виконано

	документів (відгуку, рецензії) до захисту			
15	Захист МКР перед екзаменаційною комісією (ЕК)	22.02.2023	23.02.2023	Виконано

Розробив студент Ковалів О. П. _____ _____
(прізвище та ініціали) (*підпис*)

Керівник роботи канд. техн. наук, доцент Сіденко Є. В. _____ _____
(*наук. ступінь, вчене звання, прізвище та ініціали)* *(підпис)*

«14» листопада 2022 р.

АНОТАЦІЯ

до магістерської кваліфікаційної роботи
студента групи 601 ЧНУ ім. Петра Могили
Коваліва Олександра Павловича
на тему: **«ДОСЛІДЖЕННЯ НЕЙРОННИХ МЕРЕЖ ДЛЯ РОЗПІЗНАВАННЯ ВІЙСЬКОВОЇ ТЕХНІКИ НА СУПУТНИКОВИХ ЗНІМКАХ»**

Актуальність даного дослідження полягає у необхідності виявлення та локалізація військових транспортних засобів для застосунків які мають завдання спостереження, відстеження чи безпеки. Ці програми вимагають точної ідентифікації та відстеження транспортних засобів, щоб військові транспортні засоби можна було легко відрізнити від не військових транспортних засобів на зображенні.

Об’єктом роботи є процеси розпізнавання військової техніки на супутникових знімках.

Предметом роботи є методи навчання нейронних мереж для розпізнавання військової техніки на супутникових знімках.

Метою роботи є дослідження та порівняння моделей нейронних мереж для розпізнавання військової техніки на супутникових знімках.

В результаті виконання роботи було досліджено один багатошаровий перцептрон, три моделі згорткових нейронних мереж, нейронну мереж за архітектурою VGG та XCeption, визначені основні їх переваги та недоліки, а та розроблено програмне забезпечення, з реалізованими нейронними мережами.

Дана робота складається з п’яти розділів. Кожен розділ відповідно присвячений: аналізу предметної області, нейронним мережам та моделям для

розпізнавання військової техніки на супутникових знімках, використаним у магістерській роботі, моделюванню і проектуванню нейронних мереж та аналізу отриманих результатів, охороні праці, методичній частині магістерської роботи. Загальний обсяг роботи – 132 сторінки. Магістерська кваліфікаційна робота містить, 84 рисунків, 4 таблиці і посилання на 55 літературних джерел.

Ключові слова: класифікація, штучний інтелект, нейромережі, аналіз зображень, Python.

ABSTRACT

to the master's qualification work by the student of the group 601 of Petro Mohyla Black Sea National University

Kovaliv Oleksandr

«RESEARCH OF NEURAL NETWORKS FOR RECOGNIZING MILITARY VEHICLES ON SATELLITE IMAGES»

The relevance of this research lies in the need to identify and locate military vehicles for applications that have the task of surveillance, tracking or security. These applications require accurate vehicle identification and tracking so that military vehicles can be easily distinguished from non-military vehicles in an image.

The object of the work is the process of recognizing military equipment on satellite images.

The subject of the work is neural network training methods for recognizing military equipment on satellite images.

The method of work is research and comparison of neural network models for recognizing military equipment on satellite images.

As a result of the work, one multilayer perceptron, three models of convolutional neural networks, neural networks based on the architecture of VGG and XCeption were investigated, their main advantages and disadvantages were determined, and software with implemented neural networks was developed.

This work consists of five sections. Each chapter is respectively dedicated to: analysis of the subject area, neural networks and models for recognizing military equipment on satellite images used in the master's work, modeling and design of neural networks and analysis of the obtained results, labor protection, methodical part of the master's work. The total amount of work is 132 pages. Master's thesis contains 84 figures, 4 tables and references to 55 literary sources.

Keywords: classification, artificial intelligence, neural networks, image analysis, Python.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

ШІ – штучний інтелект

ШНМ – штучна нейронна мережа

AI – artificial intelligence

CNN – convolutional neural network

DCNN – deep convolutional neural networks

GAN – generative adversarial network

HOG – histogram of oriented gradients

ML – machine learning

MLP – multilayer perceptron

R-CNN – region-based convolutional neural network

ReLU – rectified linear unit

RPN – region proposal network

SSD – single shot detector

VGG – Visual Geometry Group

VOTT – visual object tagging tool

YOLO – you only look once

Пояснювальна записка

до магістерської кваліфікаційної роботи

на тему:

«ДОСЛІДЖЕННЯ НЕЙРОННИХ МЕРЕЖ ДЛЯ РОЗПІЗНАВАННЯ ВІЙСЬКОВОЇ ТЕХНІКИ НА СУПУТНИКОВИХ ЗНІМКАХ»

Спеціальність 122 «Комп’ютерні науки»

122 – МКР – 601.1710213

Виконав студент 6-го курсу, групи 601
_____ ***О. П. Ковалів***
«15» лютого 2023 р.

Керівник: канд. техн. наук, доцент
_____***Є. В. Сіденко***
«15» лютого 2023 р.

Миколаїв – 2023

ВСТУП

Виявлення та локалізація військових транспортних засобів життєво важливі для застосунків які мають завдання спостереження, відстеження чи безпеки. Ці програми вимагають точної ідентифікації та відстеження транспортних засобів, щоб військові транспортні засоби можна було легко відрізнити від не військових транспортних засобів на зображенні. Виявлення транспортних засобів на супутникових зображеннях є життєво важливим як для цивільних, так і для військових. Таким чином, тема є актуальною.

Об’єктом роботи є процеси розпізнавання військової техніки на супутникових знімках.

Предметом роботи є методи навчання нейронних мереж для розпізнавання військової техніки на супутникових знімках.

Метою роботи є дослідження та порівняння моделей нейронних мереж для розпізнавання військової техніки на супутникових знімках. Для досягнення цієї мети необхідно було побудувати моделі класифікації зображень та знайти найкращий класифікатор з моделей, протестувати моделі протягом кількох ітерацій та оцінити їх щодо остаточного набору даних.

Було поставлено наступні задачі:

1) проаналізувати сучасний стан задачі розпізнавання об’єктів, зокрема військової техніки на супутникових знімках;

2) дослідити існуючі технології та засоби для розпізнавання військової техніки на супутникових знімках; реалізувати моделі нейронних мереж для виявлення військової техніки;

3) дослідити показники навчання моделей нейронних мереж для розпізнавання та виявлення об’єктів на супутникових і аерофотознімках.

Перший розділ стосується огляду наявних засобів для розпізнавання об’єктів на супутникових знімках, представлено актуальність роботи та визначено задачі роботи. Другий розділ стосується опису видів нейромереж та обґрунтовано їх вибір для розпізнавання військової техніки на супутникових знімках. Третій розділ представляє опис процесу розробки та навчання нейромережі, а також обрано метрики оцінки їх точності, проведено порівняння різних моделей нейронних мереж та розроблено згорткову нейронну мережу для поставлених задач.

1 АНАЛІЗ ПРЕДМЕТНОЇ СФЕРИ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Опис предметної сфери

Виявлення та локалізація військових транспортних засобів життєво важливі для застосунків які мають завдання спостереження, відстеження чи безпеки. Ці програми вимагають точної ідентифікації та відстеження транспортних засобів, щоб військові транспортні засоби можна було легко відрізнити від не військових транспортних засобів на зображенні. Щоб зменшити навантаження на фахівців з охорони та стеження, необхідно побудувати автоматичну систему виявлення військової техніки. Виявлення транспортних засобів на супутникових зображеннях є життєво важливим як для цивільних, так і для військових [1, 2].

Оскільки супутникові зображення стали дуже важливими в сучасних конфліктах, тема дослідження можливості використання моделі глибокого навчання для ідентифікації та точної класифікації різних типів військових транспортних засобів, які можуть бути знайдені на полі бою є актуальною. Дедалі більше використання супутникових зображень для відстеження конфліктів призвело до відповідного збільшення величезного обсягу створюваних зображень, і завдяки цьому, рішення, розроблене для автоматичної ідентифікації різних транспортних засобів, може допомогти спостерігачам переглядати тисячі зображень і швидше аналізувати ті, що містять предмети інтересу [2-4].

Виявлення транспортних засобів за аерофотознімками привернуло увагу в усьому світі. Це складно через невеликі розміри та змінну орієнтацію транспортних засобів. Аерофотознімки зі складним фоном ускладнюють виявлення та

класифікацію. Раніше виявлення транспортних засобів виконувалося із застосуванням методів, що складаються з ручних функцій і класифікатора або каскаду класифікаторів у рамках ковзного вікна (sliding window). Підхід пошуку через ковзне вікно та методи, засновані на дрібному навчанні, здебільшого використовувався для виявлення транспортних засобів. Підхід виявляє транспортні засоби за двома атрибутами (орієнтація та тип) на аерофотознімках [5]. Для визначення місцезнаходження транспортних засобів він використовує класифікатор AdaBoost у структурі м’якого каскаду та швидкий двійковий детектор із використанням ICF. Пізніше для класифікації орієнтації та типу транспортного засобу використовували ознаки HOG. Це забезпечує ефективне виявлення. Обчислення згорткових функцій окремо для кожного вікна-кандидата є дорогим. Після цього для класифікації регіонів-кандидатів використовувалися згорткові нейронні мережі (CNN). Нещодавно методи виявлення, засновані на R-CNN, показали хороші результати на зображеннях природних сцен [6].

Повністю згортка RPN використовується для створення об’єктоподібних областей у Faster R-CNN, а регіони-кандидати виводяться класифікатором після RPN. Продуктивність Faster R-CNN перевершує ефективність традиційних методів (на основі ковзних вікон) через високу швидкість і представлення функцій. Завдяки Fast R-CNN і Faster R-CNN витрати на обчислення для навчання та тестування були значно зменшені. Вони досягли хороших результатів на стандартних наборах даних тестів виявлення [7]. У цих методах лише одна карта згорткових функцій використовується для всього зображення замість обчислення згорткових функцій окремо. Але ефективність цих методів залежить від методів пропозиції об'єктів. Ці

детектори та їхні відповідні методи пропозиції об’єктів були розроблені для наборів даних, які відрізнялися від аерофотознімків [7-9].

Через меншу кількість навчальних даних може виникнути проблема перенавчання в CNN для виявлення транспортних засобів на аерофотознімках. Погана продуктивність Faster R-CNN пояснюється двома причинами. По-перше, через грубі карти функцій RPN у Faster RCNN не підходять для виявлення малих транспортних засобів. По-друге, через менш жорсткі негативні приклади, класифікатор після RPN не в змозі належним чином розрізнити транспортні засоби та шум. Мета-архітектура SSD вирішує проблему розпізнавання об'єктів. Мережа прямої згортки використовується для створення набору обмежувальних прямокутників фіксованого розміру. Ця мережа об’єднує прогнози з кількох карт функцій з різною роздільною здатністю. Таким чином він здатний обробляти об’єкти різного розміру. SSD дозволяє уникнути створення пропозицій і заощадити обчислювальний час завдяки інкапсуляції процесу в єдину мережу [10, 11].

1.2 Огляд та аналіз наявних аналогів та публікацій

В ході виконання кваліфікаційної роботи було досліджено та проаналізовано існуючі рішення та публікації для розпізнавання об'єктів на супутникових знімках.

В публікації «Автоматизоване виявлення військової техніки за аерофотознімками з низької висоти» [4] розглядається створення нейронної мережі здатної відрізнити військову та не військову техніку. Починаючи з відео, що містять військову машину, було використано інструмент VOTT, щоб анотувати її кадр за кадром, щоб створити наш запропонований набір даних. Інструмент генерував

анотації у форматі PASCAL VOC для 11 типів транспортних засобів. Із загальної кількості 15086 зображень у наборі даних 11733 взято з інсценованого відео, а 3353 — зі справжнього відео. Зібраний набір даних анотовано для двох категорій транспортних засобів (військових та не військових). Зображення в наборі даних містять кілька об’єктів, що належать до кількох класів. Загалом в цій публікації розглядаються 13 класів, які поділяються на 2 основні категорії. Категорія транспортного засобу та не транспортного засобу. Загалом є 15086 зображень, які вручну позначили всі зображення обмежувальною рамкою та типом.

У дослідженні «Обробка аерофотознімків для виявлення автомобілів за допомогою згорткових нейронних мереж: порівняння швидших R-CNN і YoloV3» [6] автори порівняли найсучасніший алгоритм глибокого навчання, а саме Faster R-CNN, і YOLOv3 для виявлення автомобілів за аерофотознімками. Дослідники також вивчали вплив гіперпараметрів на різні алгоритми та прийшли до висновку, що на конкретному наборі даних Faster RCNN дав кращі результати щодо швидкості висновку, тоді як YOLOv3 показав кращі результати на наборі даних PSU із вхідними зображеннями розміром 320x320.

Дисертаційне дослідження на тему «Виявлення літаків, транспортних засобів і кораблів на аерофотознімках і супутникових знімках за допомогою еволюційного глибокого навчання» [7] спрямована на виявлення об’єктів, таких як літаки, транспортні засоби та кораблі, із супутникових зображень та аерофотознімків з використанням гіперпараметрів, отриманих за результатами впровадження генетичного алгоритму, цей процес пошуку гіперпараметрів за допомогою генетичного алгоритму називається еволюційним глибоким навчанням.

Автори статті «Виявлення транспортних засобів на супутникових зображеннях високої роздільної здатності за допомогою згорткових нейронних мереж з з'єднаним шаром» [8] запропонували метод об'єднаного шару глибоких згорткових нейронних мереж (JLDCNN) для виявлення транспортних засобів і виявили, що новий підхід, який об'єднує верхній і нижній рівні глибоких згорткових нейронних мереж (DCNN), продемонстрував покращення на 16% і 6% у точності та запам'ятовуванні порівняно з традиційним DCNN.

У досліджені «Виявлення кораблів на оптичних супутникових зображеннях за допомогою спрямованих обмежувальних прямокутників на основі передбачення центру корабля та орієнтації» [9] було запроваджено двоетапну техніку виявлення кораблів на основі CNN, яка фокусується на центрі корабля та орієнтації на зображеннях. Автори порівняли цей метод із найсучаснішими методами розпізнавання об'єктів, такими як Faster R-CNN, SSD і YOLO, і виявили, що новий підхід успішно перевершує інші методи.

Автори дослідження «Технологія дистанційного зондування зображення літака на основі глибокого навчання» [10] використовували YOLOv3 і алгоритм Faster R-CNN для виявлення літаків. Для попередньої обробки зображень використовувався подвійний пороговий алгоритм випадкової вибірки. Результати експерименту показують, що Faster R-CNN має кращу точність, але пропускає деякі дрібні цілі, тоді як YOLOv3 добре виявляє менші цілі на зображеннях.

У статті «Структура виявлення повітряних суден на основі підкріпленого навчання та згорткових нейронних мереж у зображеннях дистанційного зондування» [11] автори запропонували нову структуру виявлення літаків RL-CNN на основі навчання з підкріпленням і CNN. Виявлено, що цей метод використання системи на

основі винагороди продемонстрував найсучаснішу продуктивність для виявлення літаків.

У дослідженні «Точне виявлення об'єктів на основі швидкого R-CNN у зображеннях дистанційного зондування» [12] пропонується методика підвищення точності виявлення на зображеннях дистанційного зондування шляхом оптимізації опорних блоків у Faster RCNN з використанням мереж ResNet-50 і ResNet-101.

Автори в дослідженні «Виявлення судна з використанням мультиблокового детектора з однократним пострілом, навченого трансфером» [13] запропонували перенести навчити мультиглибокий детектор з однократним пострілом за допомогою трансферу для виявлення кораблів. Проведений експеримент продемонстрував, що цей метод досягає точності 87,9% і здатний працювати зі швидкістю 47 кадрів в секунду, таким чином задовольняючи вимогам реального часу.

1.3 Ресурси дослідження

Було використано набір даних для виявлення та розпізнавання рухомих і стаціонарних цілей (MSTAR), створений Агентством передових оборонних дослідницьких проектів США (DARPA) і Дослідницькою лабораторією ВПС США з 1995 по 1997 рік. Набір даних доступний для для загального використання на веб-сайті науково-дослідної лабораторії ВПС. Дані містять такі 8 класів:

- 2C1 Гвоздика — Самохідна артилерійська установка;
- ЗСУ-23–4 Шилка — зенітна самохідна установка;
- БРДМ-2 — броньована машина-розвідник-амфібія;
- БТР-60 — бронетранспортер;

– D7 — бульдозер Caterpillar;
– ЗІЛ-131 — військово-вантажний автомобіль;
– Т-62 — основний бойовий танк;
– SLICY — структура, що діє як укриття (а не транспортний засіб).

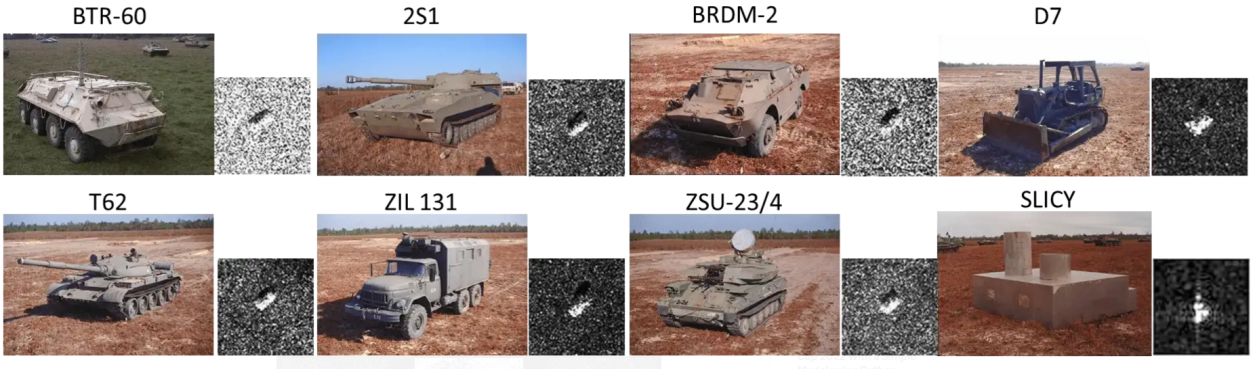


Рисунок 1.1 – Класи військової техніки та їх супутникові знімки

Зображення кожної техніки були зроблені за допомогою радара з синтетичною апертурою (SAR). Це форма супутникових зображень, яка вміє визначати техніки незважаючи на атмосферні умови, такі як хмари.

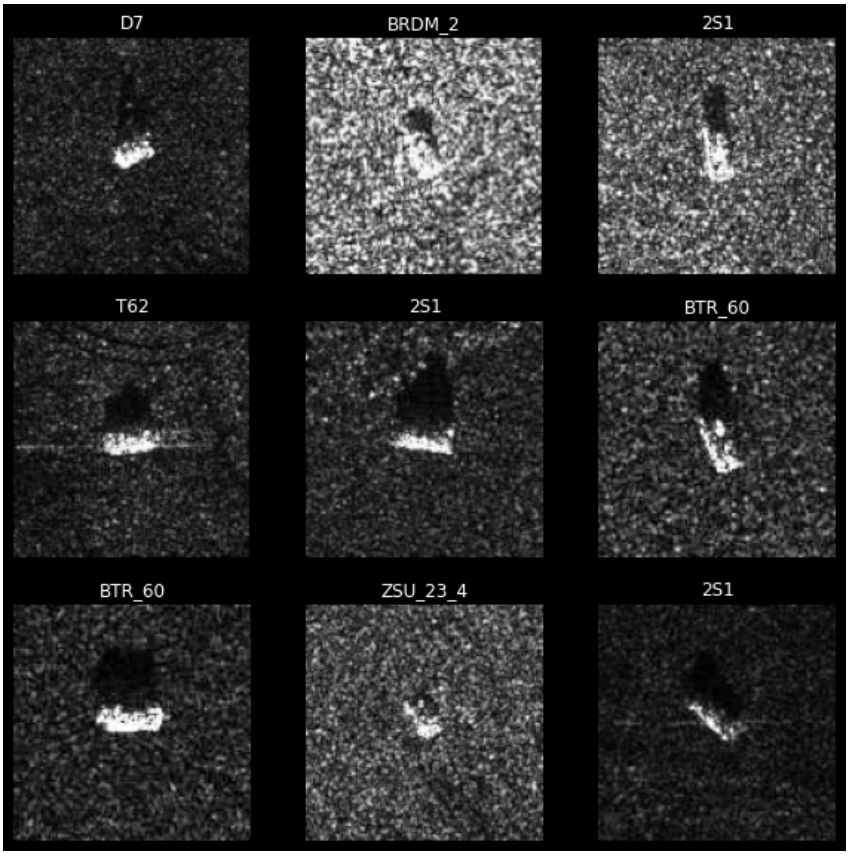


Рисунок 1.2 – Супутникові знімки класів військової техніки

В якості останнього тесту було перевірено, як працюють моделі на зображенні, отриманому поза набором даних MSTAR. Мета полягала в тому, щоб побачити, чи може модель узагальнити зображення, отримані з контрольованого середовища, чи знадобиться більше даних «реального світу», щоб зробити модель надійною.

Щоб сформулювати напрямок роботи, було використано показники, агрегованих у 8 класах, щоб керувати ітераціями:

– категорична точність: середнє значення показників точності для 9 класів;

– середньозважена оцінка F2: для цього випадку більший пріоритет було надано запам’ятовуванню над точністю, тому було використано середню оцінку F2 за категоріями.

– коефіцієнт кореляції Метьюза: числове представлення ефективності класифікатора. 1 = Ідеально, 0 = Випадкове вгадування. Цей показник інформував про зміни моделі.

Глибоке навчання вимагає ресурсів. Спроба тренувати відносно прості моделі на ноутбуці може зайняти години або навіть дні. Через це доцільно використовувати графічні процесори для завдань глибокого навчання.

Найдоступніший спосіб отримати доступ до графічних процесорів для глибокого навчання – через Colab Pro від Google. Colab – це хмарна служба для виконання проектів з обробки даних.

Блокноти Colab дозволяють поєднувати виконуваний код і форматований текст в одному документі разом із зображеннями, HTML, LaTeX тощо. Colab дозволяє використовувати всю потужність популярних бібліотек Python для аналізу та візуалізації даних.

Висновки до розділу 1

Було описано предметну сферу, доведено актуальність дослідження, досліджено та проаналізовано існуючі рішення та публікації для розпізнавання об'єктів на супутникових знімках, серед яких особливий вплив на дослідження мали «Виявлення літаків, транспортних засобів і кораблів на аерофотознімках і супутникових знімках за допомогою еволюційного глибокого навчання» за

авторством Акшай Кумар Тудоджу [7] та «Автоматизоване виявлення військової техніки за аерофотознімками з низької висоти» [4] за авторством Фаррух Камран, Мухаммад Шахзад і Фейсал Шафаїт. Розглянуто ресурси, які були використані для проведення дослідження, а саме:

– набір даних для виявлення та розпізнавання рухомих і стаціонарних цілей (MSTAR);

– хмарна служба для виконання проектів з обробки даних Google Colab.

2 НЕЙРОННІ МЕРЕЖІ ДЛЯ РОЗПІЗНАВАННЯ ВІЙСЬКОВОЇ ТЕХНІКИ НА СУПУТНИКОВИХ ЗНІМКАХ

Для розпізнавання військової техніки на супутникових знімках було використано різні види нейронних мереж.

2.1 Багатошаровий перцептрон (MLP)

Повністю пов’язана багатошарова нейронна мережа називається багатошаровим перцептроном (MLP) [14].

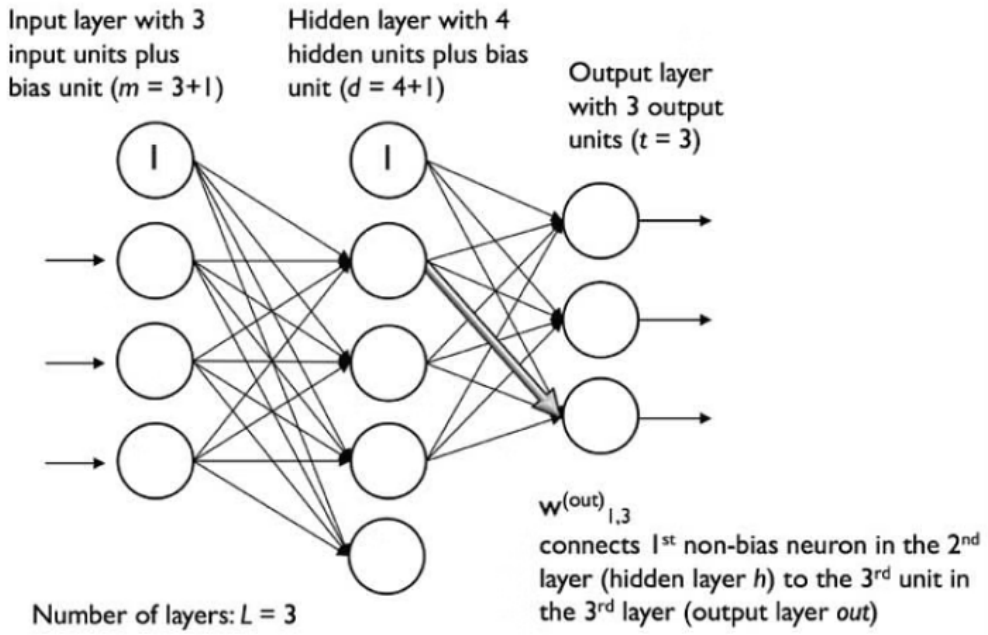


Рисунок 2.1 – Багатошаровий перцептрон (MLP)

Він має 3 шари, включаючи один прихований. Якщо він має більше ніж 1 прихований шар, він називається глибокою ШНМ. MLP є типовим прикладом прямої (feedforward) штучної нейронної мережі [15].

Кількість шарів і кількість нейронів називаються гіперпараметрами нейронної мережі, і вони потребують налаштування. Щоб знайти ідеальні значення для них, необхідно використовувати методи перехресної перевірки.

Тренування коригування ваги здійснюється за допомогою зворотного поширення. Більш глибокі нейронні мережі краще обробляють дані. Однак глибші шари можуть призвести до проблем із зникаючим градієнтом. Для вирішення цієї проблеми потрібні спеціальні алгоритми [16].

На рисунку 2.1 показана повністю зв’язана тришарова нейронна мережа з 3 вхідними нейронами та 3 вихідними нейронами. Член зміщення додається до вхідного вектора.

Процедура навчання MLP виглядає наступним чином:

1) починаючи з вхідного рівня, дані передаються на вихідний рівень. Цей крок є прямим поширенням;

2) на основі результату обчислюється похибку (різниця між прогнозованим і відомим результатом). Помилку потрібно звести до мінімуму;

3) зворотне поширення помилки. Необхідно знайти його похідну відносно кожної ваги в мережі та оновити модель;

4) перші три кроки, наведені вище, повторюються протягом кількох епох, щоб дізнатися ідеальну вагу.

Нарешті, вихідні дані беруться за допомогою порогової функції для отримання прогнозованих міток класу [17-20].

Пряме поширення в MLP відбувається за допомогою обчислення одиниці активації $a_1^{(h)}$ прихованого шару [20].

$$z_1^{(h)} = a_0^{(in)} w_{0,1}^{(h)} + a_1^{(in)} w_{1,1}^{(h)} + \ldots + a_m^{(in)} w_{m,1}^{(h)} \tag{2.1}$$

$$a_1^{(h)} = \varphi(z_1^{(h)}),$$

$$\varphi(z) = \frac{1}{1+e^{-z}},$$

де $a_i^{(in)}$ – відноситься до і-го значення у вхідному шарі;

$a_i^{(h)}$ – відноситься до і-го блоку в прихованому шарі;

$a_i^{(out)}$ – відноситься до і-го блоку вихідного рівня;

$a_0^{(in)}$ – член зміщення (bias) і дорівнює 1; він матиме відповідну вагу w_0;

$w_{i,1}^{(l)}$ – ваговий коефіцієнт від шару l до шару l+1;

$\varphi(z)$ – функція активації;

$z_i^{(h)}$ – сума зважених значень входів;

Одиниця активації є результатом застосування функції активації φ до значення z. Він повинен бути диференційованим, щоб мати можливість вивчати ваги за допомогою градієнтного спуску. Функція активації φ часто є сигмоїдною (логістичною) функцією [21].

Це дозволяє нелінійність, необхідну для вирішення складних проблем, таких як обробка зображень.

2.2 Згорткова нейронна мережа (CNN/ConvNet)

Згорткова нейронна мережа (CNN або ConvNet) — це мережева архітектура для глибокого навчання, яке навчається безпосередньо з даних.

CNN особливо корисні для пошуку шаблонів у зображеннях для розпізнавання об’єктів, класів і категорій. Вони також можуть бути досить ефективними для класифікації аудіо, часових рядів і даних сигналу [22].

Згорткова нейронна мережа може мати десятки або сотні шарів, кожен з яких навчається виявляти різні особливості зображення. Фільтри застосовуються до кожного тренувального зображення з різною роздільною здатністю, а вихідні дані кожного згорнутого зображення використовуються як вхідні дані для наступного шару. Фільтри можуть починатися з дуже простих функцій, таких як яскравість і межі, і збільшуватися в складності до функцій, які однозначно визначають об’єкт [23-25].

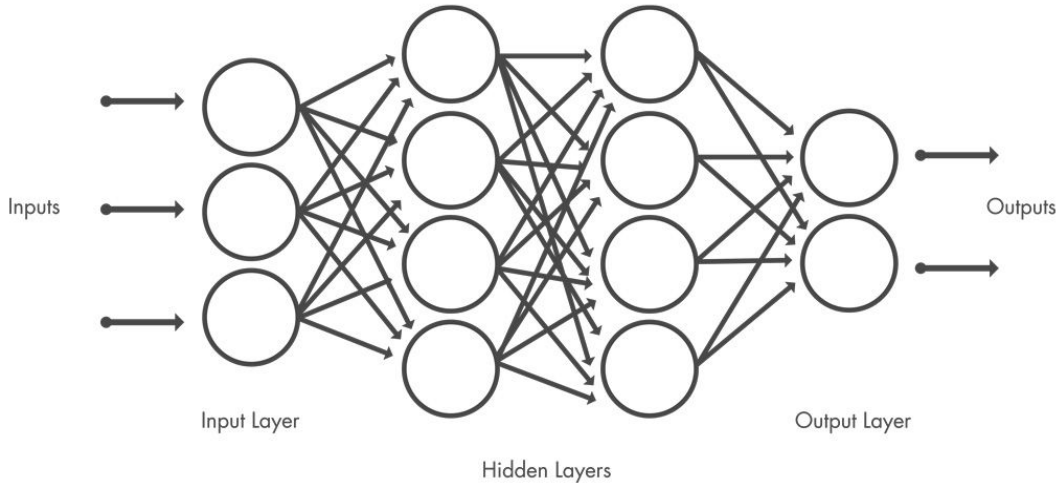


Рисунок 2.2 – Згорткова нейронна мережа

CNN складається з вхідного рівня, вихідного рівня та багатьох прихованих шарів між ними, як показано на рисунку 2.2.

Ці рівні виконують операції, які змінюють дані з метою вивчення особливостей даних. Три найпоширеніші рівні: згортка, активація або ReLU та об'єднання [26].

Згортка пропускає вхідні зображення через набір згорткових фільтрів, кожен із яких активує певні функції зображень.

Випрямлена лінійна одиниця (ReLU) дозволяє швидше та ефективніше навчатися, пов'язуючи від'ємні значення з нулем і зберігаючи додатні значення. Це іноді називають активацією, оскільки лише активовані функції переносяться на наступний рівень [27].

Об'єднання спрощує вихід, виконуючи нелінійне зменшення дискретизації, зменшуючи кількість параметрів, які мережа повинна вивчати.

Ці операції повторюються на десятках або сотнях шарів, причому кожен шар навчається визначати різні функції.

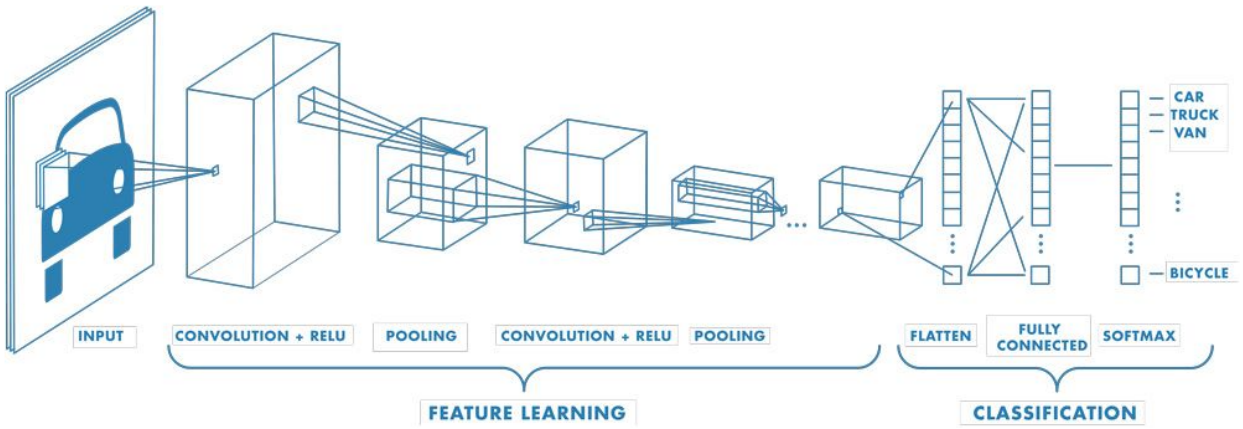


Рисунок 2.3 – Приклад мережі з багатьма згортковими шарами

Фільтри застосовуються до кожного тренувального зображення з різною роздільною здатністю, а вихідні дані кожного згорнутого зображення використовуються як вхідні дані для наступного шару [28].

На відміну від традиційної нейронної мережі, CNN має спільні ваги та значення зміщення, які однакові для всіх прихованих нейронів у даному шарі.

Це означає, що всі приховані нейрони виявляють ту саму функцію, таку як край або крапка, у різних областях зображення. Це робить мережу толерантною до перекладу об’єктів у зображенні. Наприклад, мережа, навчена розпізнавати автомобілі, зможе робити це, де б автомобіль не був на зображенні [29, 30].

Після вивчення функцій на багатьох рівнях архітектура CNN переходить до класифікації.

Передостанній шар — це повністю зв’язаний шар, який виводить вектор K-вимірів (де K — кількість класів, які можна передбачити) і містить ймовірності для кожного класу зображення, яке класифікується.

Останній рівень архітектури CNN використовує класифікаційний рівень для надання кінцевих результатів класифікації [31].

CNN забезпечують оптимальну архітектуру для виявлення та вивчення ключових характеристик зображень і даних часових рядів. CNN є ключовою технологією програмах, головна ціль яких є:

– медична візуалізація: CNN можуть вивчати тисячі звітів про патологію, щоб візуально виявити наявність або відсутність ракових клітин на зображеннях [32];

– обробка аудіо: виявлення ключових слів можна використовувати в будь-якому пристрої з мікрофоном, щоб виявити, коли вимовляється певне слово чи

фраза («Привіт, Сірі!»). CNN можуть точно вивчити та визначити ключове слово, ігноруючи всі інші фрази незалежно від середовища [33];

– виявлення об’єктів: автоматизоване водіння покладається на CNN для точного виявлення наявності знака чи іншого об’єкта та прийняття рішень на основі вихідних даних [33];

– генерація синтетичних даних: за допомогою генеративних суперницьких мереж (GAN) можна створювати нові зображення для використання в програмах глибокого навчання, включаючи розпізнавання облич і автоматизоване водіння [34].

2.3 XCeption

XCeption — це архітектура глибокої згорткової нейронної мережі, яка включає згортки, що розділяються по глибині. Його розробили дослідники Google. Google представив інтерпретацію модулів Inception у згорткових нейронних мережах як проміжний крок між звичайною згорткою та операцією поглибленої згортки (за поглибленою згорткою слідує поточкова згортка). У цьому світлі розділену по глибині згортку можна розуміти як модуль Inception з максимально великою кількістю башт. Дякуючи цьому спостереженню, вони запропонували нову архітектуру глибокої згорткової нейронної мережі, натхненну Inception, де модулі Inception були замінені згортками, які можна розділити по глибині [35].

Оригінальна вихідна згортка, що розділяється по глибині, є згорткою по глибині, за якою слідує поточкова згортка. Згортка по глибині – це просторова згортка n×n по каналу. Припустимо, що на рисунку 2.4 маємо 5 каналів, тоді

матимемо 5 n×n просторової згортки. Поточкова згортка насправді є згорткою 1×1 для зміни розміру [35, 36].

Модифікована роздільна згортка по глибині є поточковою згорткою, за якою слідує згортка по глибині. Ця модифікація мотивована початковим модулем у Inception-v3, згідно з яким спочатку виконується згортка 1×1 перед будь-якими просторовими згортками n×n. Таким чином, вона відрізняється від оригінального. (тут n=3, оскільки в Inception-v3 використовуються просторові згортки 3×3) [37].

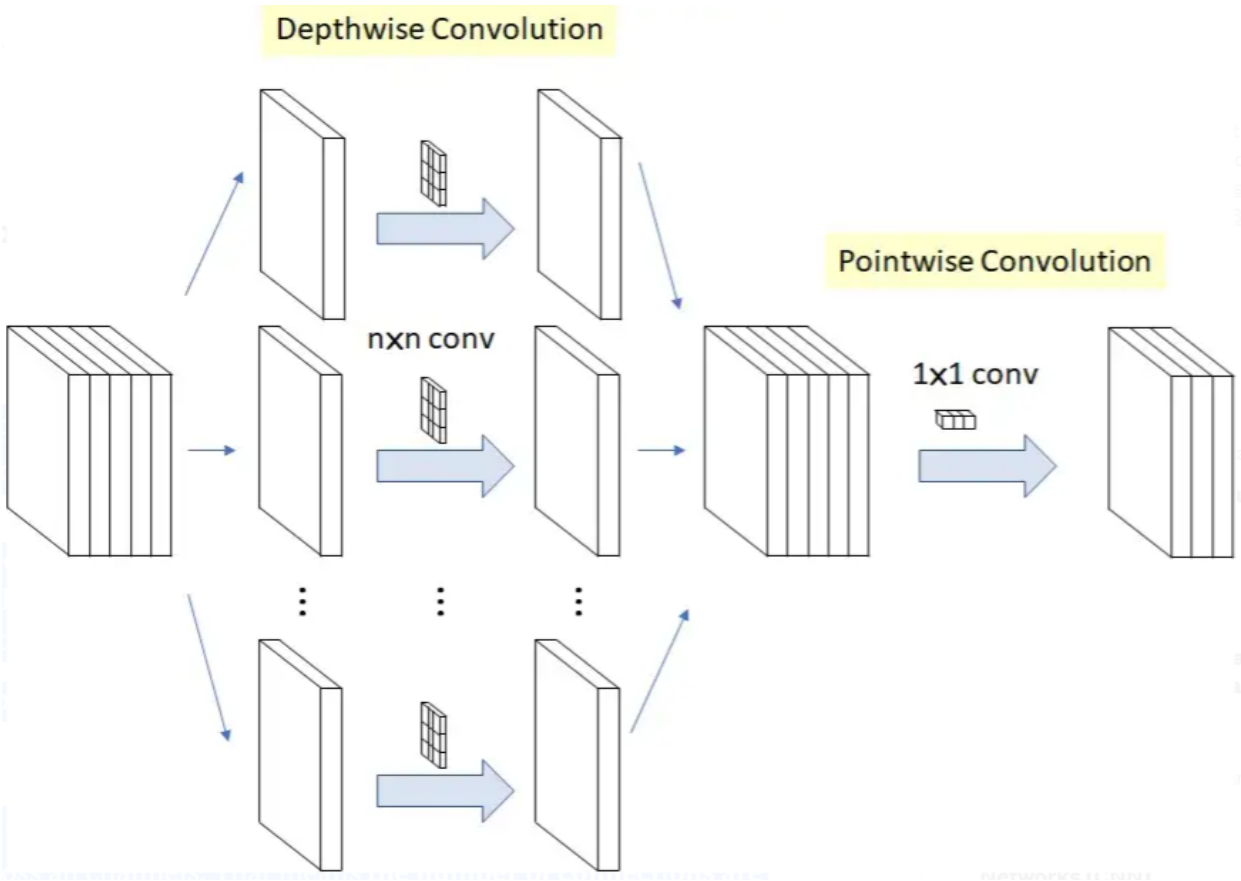


Рисунок 2.4 – Оригінальна роздільна згортка по глибині

Дві незначні відмінності:

– порядок операцій: оригінальні роздільні згортки по глибині, як це зазвичай реалізовано (наприклад, у TensorFlow), виконують спочатку просторову згортку по каналу, а потім виконують згортку 1 × 1, тоді як модифікована згортка з роздільною глибиною спочатку виконує згортку 1 × 1, а потім – поканальну просторову згортку. Вважається, що це неважливо, тому що, коли воно використовується в налаштуваннях стека, на початку та в кінці всіх зв’язаних початкових модулів з’являються лише невеликі відмінності [37, 38];

– наявність/відсутність нелінійності: в оригінальному початковому модулі є нелінійність після першої операції. У XCeption, модифікованій роздільній згортці по глибині, немає проміжної нелінійності ReLU [39].

У порівнянні зі звичайною згорткою не потрібно виконувати згортку по всіх каналах. Це означає, що кількість підключень менше, а модель легша.

В XCeption дані спочатку проходять через вхідний потік, потім через середній потік, який повторюється вісім разів, і, нарешті, через вихідний потік. Зверніть увагу, що на рисунку 2.5 після всіх шарів Convolution і SeparableConvolution виконується пакетна нормалізація [40].

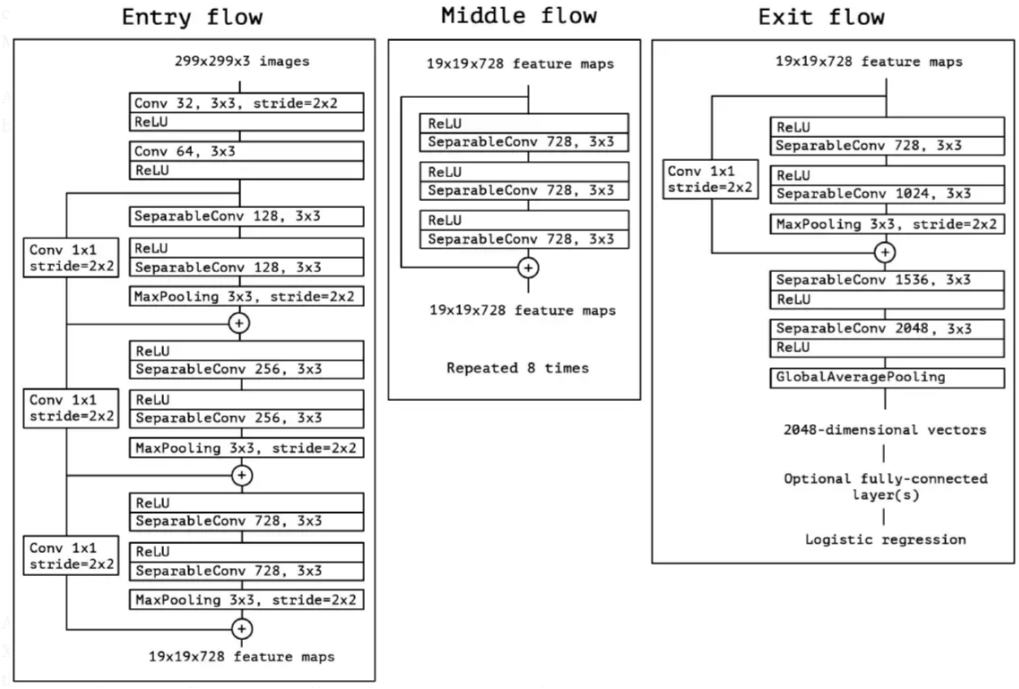


Рисунок 2.5 – Загальна архітектура XCeption

XCeption — це ефективна архітектура, яка базується на двох основних моментах:

– роздільна згортка по глибині;

– комбінації клавіш між блоками згортки, як у ResNet.

Згортки, що розділяються по глибині, є альтернативою класичним згорткам, які мають бути набагато ефективнішими з точки зору часу обчислення [41].

Специфіка XCeption полягає в тому, що за глибинною згорткою не слідує поточкова згортка, але порядок є зворотним, як на рисунку 2.6:

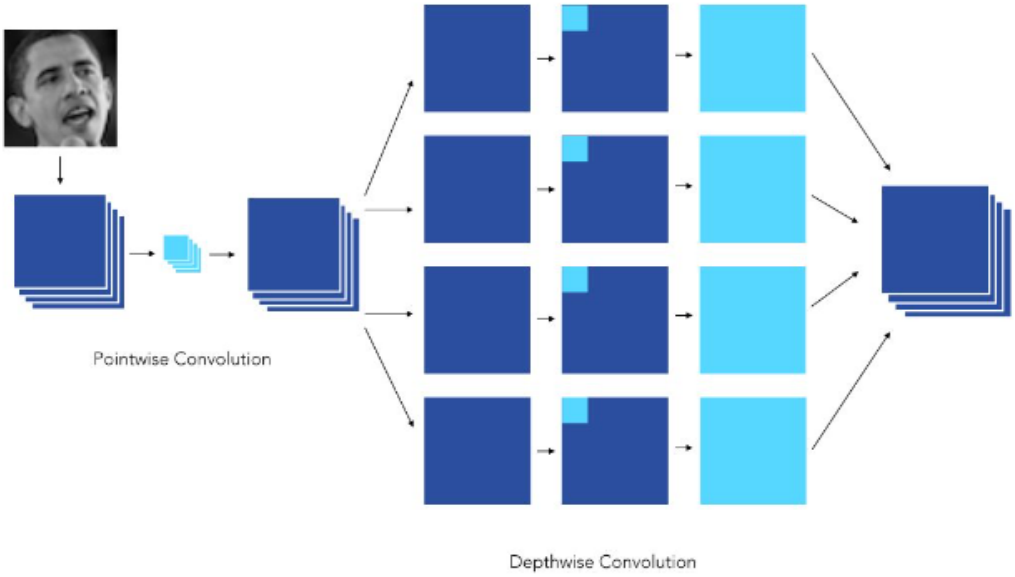


Рисунок 2.6 – Приклад архітектури XCeption

2.4 Visual Geometry Group (VGG)

VGG означає Visual Geometry Group; це стандартна глибока архітектура згорткової нейронної мережі (CNN) із кількома рівнями. «Глибокий» відноситься до кількості шарів з VGG-16 або VGG-19, що складаються з 16 і 19 згорткових шарів [42]. Архітектура VGG є основою новаторських моделей розпізнавання об’єктів. Розроблена як глибока нейронна мережа, VGGNet також перевершує базові показники для багатьох завдань і наборів даних за межами ImageNet. Крім того, зараз це одна з найпопулярніших архітектур розпізнавання зображень [43]. На рисунку 2.7 зображено архітектуру VGG.

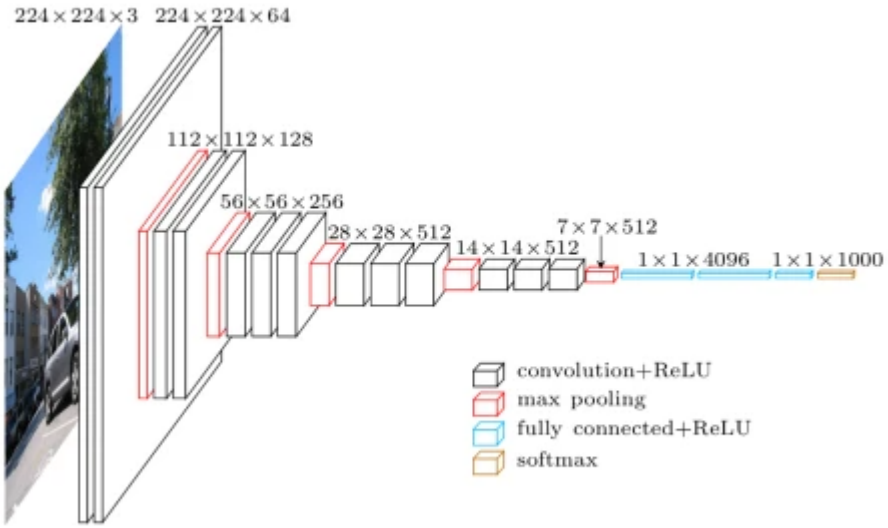


Рисунок 2.7 – Архітектура VGG

Модель VGG, або VGGNet, яка підтримує 16 рівнів, також називається VGG16, яка є моделлю згорткової нейронної мережі, запропонованою А. Зіссерманом і К. Симоньян з Оксфордського університету. Ці дослідники опублікували свою модель у дослідницькій статті під назвою «Дуже глибокі згорткові мережі для розпізнавання великомасштабних зображень» [44]. Модель VGG16 досягає майже 92,7% точності топ-5 тестів ImageNet. ImageNet — це набір даних, що складається з понад 14 мільйонів зображень, що належать до майже 1000 класів. Крім того, це була одна з найпопулярніших моделей, представлених на ILSVRC-2014. Він замінює великі фільтри розміром ядра кількома фільтрами розміром ядра 3 × 3 один за одним, таким чином значно покращуючи AlexNet [44, 45]. Модель VGG16 навчалася за допомогою графічних процесорів Nvidia Titan Black протягом кількох тижнів. VGGNet-16 підтримує 16 шарів і може класифікувати зображення за 1000 категоріями об’єктів,

включаючи клавіатуру, тварин, олівець, мишу тощо. Крім того, модель має вхідний розмір зображення 224 на 224 [44].

Концепція моделі VGG19 (також VGGNet-19) така ж, як і VGG16, за винятком того, що вона підтримує 19 рівнів. «16» і «19» означають кількість вагових шарів у моделі (згорткових шарів). Це означає, що VGG19 має три більше згорткових шарів, ніж VGG16 [45, 46].

Мережа VGG побудована з дуже маленькими згортковими фільтрами. VGG-16 складається з 13 згорткових шарів і трьох повністю зв'язаних шарів. VGGNet приймає вхідне зображення розміром 224×224. Для конкурсу ImageNet творці моделі вирізали центральну ділянку розміром 224 × 224 на кожному зображенні, щоб підтримувати вхідний розмір зображення. Згорткові шари VGG використовують мінімальне сприятливе поле, тобто 3×3, найменший можливий розмір, який усе ще захоплює вгору/вниз і вліво/вправо [47]. Крім того, існують також згорткові фільтри 1×1, які діють як лінійне перетворення вхідних даних. Далі йде блок ReLU, який є величезною інновацією від AlexNet, яка скорочує час навчання. ReLU означає функцію активації випрямленої лінійної установки; це кусково-лінійна функція, яка буде виводити вхідні дані, якщо вони позитивні; інакше вихід дорівнює нулю. Крок згортки фіксується на 1 пікселі, щоб зберегти просторову роздільну здатність після згортки (крок — це кількість піксельних зрушень над вхідною матрицею). Усі приховані шари в мережі VGG використовують ReLU. VGG зазвичай не використовує нормалізацію локальної реакції (LRN), оскільки це збільшує споживання пам’яті та час навчання. Крім того, це не покращує загальну точність. VGGNet має три повністю підключені рівні. З трьох рівнів перші два мають по 4096 каналів кожен, а третій має 1000 каналів, по 1 для кожного класу [48].

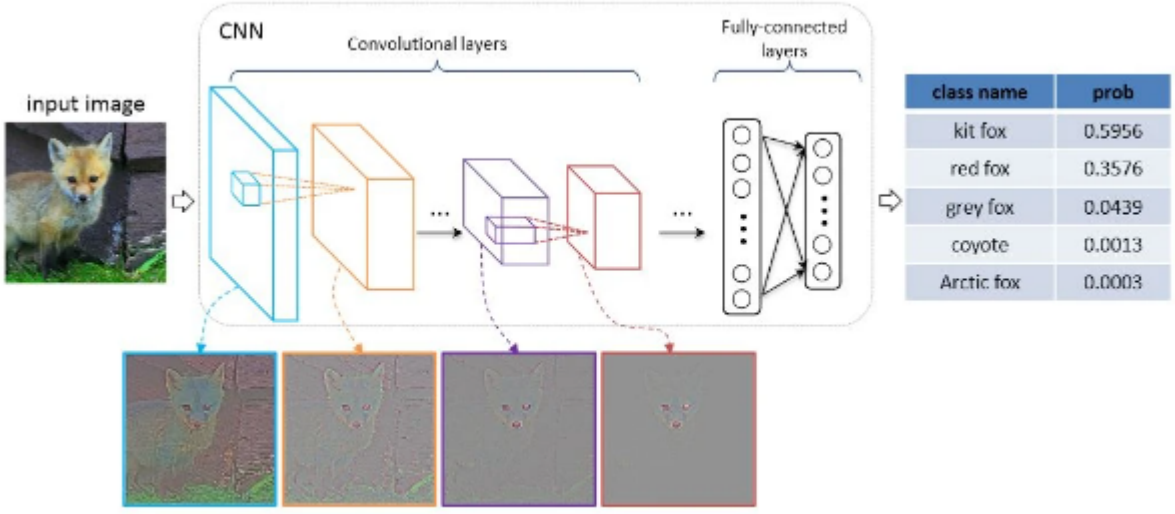


Рисунок 2.8 – Приклад роботи та архітектура VGG

Висновки до розділу 2

Було розглянуто різні архітектури нейронних мереж, такі як:

– багатошаровий перцептрон;

– згорткова нейронна мережа;

– XCeption;

– VGG;

Описано процес навчання багатошарового перцептрону та процес прямого поширення в ньому.

Проаналізовано переваги згорткової нейронної мережі та процесу згортки, пояснено значення фільтрів та розміру ядра.

Розглянуто архітектуру нейронної мережі XCeption, яка була розроблена компанією Google як інтерпретацією модулів Inception у згорткових нейронних

мережах як проміжний крок між звичайною згорткою та операцією поглибленої згортки (за поглибленою згорткою слідує поточкова згортка). Пояснено різницю між звичайною згорткою та згорткою по глибині.

Описано архітектуру нейронної мережі Visual Geometry Group (VGG) та пояснено принцип її роботи.

3 МОДЕЛЮВАННЯ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ ДЛЯ РОЗПІЗНАВАННЯ ВІЙСЬКОВОЇ ТЕХНІКИ НА СУПУТНИКОВИХ ЗНІМКАХ

Проекти ШІ відрізняються від традиційних програмних проектів. Відмінності полягають у наборі технологій, навичках, необхідних для проекту на основі ШІ, і необхідності глибоких досліджень. Щоб реалізувати прагнення щодо штучного інтелекту, слід використовувати стабільну, гнучку мову програмування та доступні інструменти. Python пропонує все це, тому сьогодні існує багато проектів Python AI.

Від розробки до розгортання та обслуговування, Python допомагає розробникам бути продуктивними та впевненими щодо програмного забезпечення, яке вони створюють. Переваги, завдяки яким Python найкраще підходить для машинного навчання та проектів на основі ШІ, включають простоту та узгодженість, доступ до чудових бібліотек і фреймворків для ШІ та машинного навчання (ML), гнучкість, незалежність від платформи та широке співтовариство. Це додає загальної популярності мови [49].

Впровадження алгоритмів AI і ML може бути складним і вимагає багато часу. Дуже важливо мати добре структуроване та перевірене середовище, щоб мати можливість створювати найкращі рішення для програмування.

Щоб скоротити час розробки, Python має ряд фреймворків і бібліотек. Бібліотека програмного забезпечення — це попередньо написаний код, який розробники використовують для вирішення типових завдань програмування. Python зі своїм багатим набором технологій має великий набір бібліотек для штучного інтелекту та машинного навчання, такі як:

– Keras, TensorFlow і Scikit-learn для машинного навчання;

– NumPy для високопродуктивних наукових обчислень і аналізу даних;
– SciPy для передових обчислень;
– Pandas для аналізу даних загального призначення;
– Seaborn для візуалізації даних;
– Scikit-learn містить різноманітні алгоритми класифікації, регресії та кластеризації, включаючи машини опорних векторів, випадкові ліси, посилення градієнта, k-середні та DBSCAN, і розроблено для роботи з числовими та науковими бібліотеками Python NumPy та SciPy.

Існує також широкий вибір Python IDE, які надають повний набір інструментів для тестування, налагодження, рефакторингу та локальної автоматизації збірки в одному інтерфейсі.

Colaboratory, або скорочено Colab, — це дослідницький продукт Google, який дозволяє розробникам писати та виконувати код Python через свій браузер. Google Colab є чудовим інструментом для глибокого навчання. Це хостинг-ноутбук Jupyter, який не потребує налаштування та має чудову безкоштовну версію, яка надає безкоштовний доступ до обчислювальних ресурсів Google, таких як графічні процесори та процесори TPU.

Є кілька причин, щоб вибрати використання Google Colab:

– попередньо встановлені бібліотеки;
– збережено в хмарі;
– співпраця;
– безкоштовне використання GPU та TPU.

Дистрибутив Anaconda Jupyter Notebook постачається з кількома попередньо встановленими бібліотеками даних, такими як Pandas, NumPy, Matplotlib, що чудово.

Google Colab, з іншого боку, надає ще більше попередньо встановлених бібліотек машинного навчання, таких як Keras, TensorFlow і PyTorch.

При використанні звичайного блокнот Jupyter як середовища розробки, усе зберігається на локальній машині. Якщо бути обережним щодо конфіденційності, ця функція може бути кращою. Однак набагато краще, щоб блокноти були доступні з будь-якого пристрою за допомогою простого входу в Google. Усі ваші блокноти Google Colab зберігаються у обліковому записі Диска Google, як і файли Google Документів і Google Таблиць.

Ще одна чудова функція Google Colab – це функція співпраці. При співпраці з кількома розробниками над проектом, чудово використовувати блокнот Google Colab. Подібно до спільної роботи над документом Google Docs, можна спільно кодувати з кількома розробниками за допомогою блокнота Google Colab. Крім того, також можливо поділитися своєю завершеною роботою з іншими розробниками.

Google Research дозволяє використовувати їх виділені графічні процесори та TPU для особистих проектів машинного навчання. Для деяких проектів прискорення GPU і TPU має величезне значення навіть для невеликих проектів. Це одна з головних причин для кодування освітніх проектів у Google Colab. Крім того, оскільки він використовує ресурси Google, операції оптимізації нейронної мережі не заважають іншим процесорам.

Перед початком виконання роботи необхідно підключити всі бібліотеки на фреймворки які можуть допомогти при програмній реалізації моделей. На рисунку 3.1 зображені необхідні бібліотеки та параметри Matplotlib, такі як:

– Tensorflow;
– Keras;

– Matplotlib;
– Seaborn;

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.preprocessing import image_dataset_from_directory
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import *
from tensorflow.keras.applications.vgg19 import VGG19
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix
import tensorflow_addons as tfa

params = {"ytick.color" : "w",
          "xtick.color" : "w",
          "axes.labelcolor" : "w",
          "axes.edgecolor" : "w",
          "figure.figsize" : (10,10),
          "axes.titlecolor" : 'w',
          "axes.facecolor" : 'w',
          "figure.facecolor" : 'k'}

colors = plt.rcParams['axes.prop_cycle'].by_key()['color']

%matplotlib inline
```

Рисунок 3.1 – Необхідні бібліотеки та параметри Matplotlib

3.1 Попередня обробка даних

Tensorflow має зручну утиліту, відому як image_dataset_from_directory, яка дозволяє вам працювати з набором даних, який уже є на хмарі. На рисунку 3.2

показано як було розділено дані навчання та перевірки, а саме 20% даних були відведені для перевірки і 80% для навчання. В сумі, датасет MSTAR пропонує 8 класів військової техніки з 9466 зображень. 7573 зображень було відведено для навчання, а перевірка буде виконуватись на 1893 зображень. Розмір зображення становить 128x128, з розміром партії (batch size) 32. Так як всі зображення були чорнобілими, необхідно було виставити колірний режим в grayscale. Варто зазначити, структуру каталогів, де кожен клас був папкою в батьківському класі.

```
train_ds = image_dataset_from_directory('drive/MyDrive/MSTAR',
                                        subset='training',
                                        image_size=image_size,
                                        labels='inferred',
                                        validation_split=.2,
                                        seed=10,
                                        label_mode='categorical',
                                        color_mode='grayscale',
                                      batch_size=batch_size)
val_ds = image_dataset_from_directory('drive/MyDrive/MSTAR',
                                        subset='validation',
                                        image_size=image_size,
                                        labels='inferred',
                                        validation_split=.2,
                                        seed=10,
                                        label_mode='categorical',
                                        color_mode='grayscale',
                                        batch_size=batch_size)
```

```
Found 7459 files belonging to 8 classes.
Using 5968 files for training.
Found 7459 files belonging to 8 classes.
Using 1491 files for validation.
```

Рисунок 3.2 – Попередня обробка даних

Для перевірки правильності даних було побудовано перші 8 зображень навчальних даних і відповідні мітки, як зображено на рисунку 3.3.

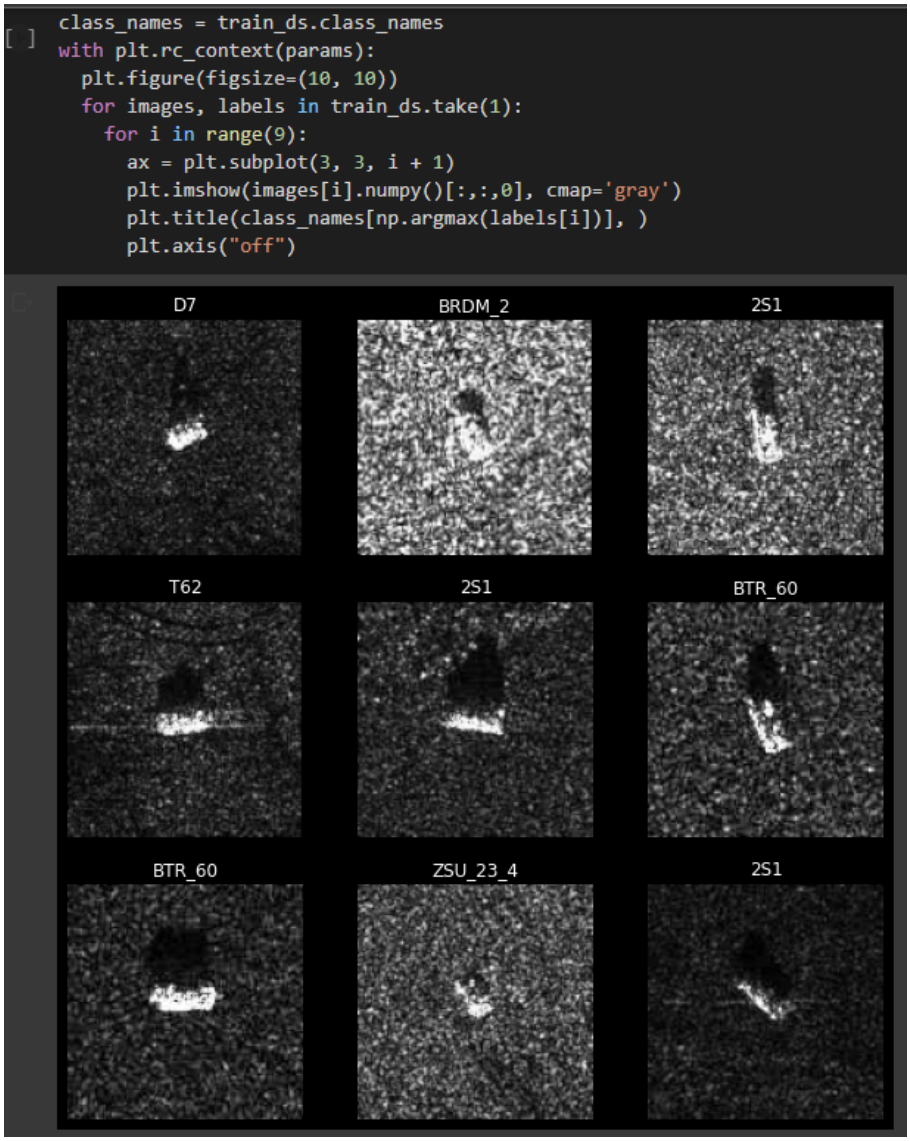


Рисунок 3.3 – Навчальні дані та їх відповідні мітки

На рисунку 3.4 зображено список існуючих класів військової техніки.

```
[ ]  class_names

     ['2S1', 'BRDM_2', 'BTR_60', 'D7', 'SLICY', 'T62', 'ZIL131', 'ZSU_23_4']
```

Рисунок 3.4 – Існуючі класи військової техніки

Дані містять такі 8 класів:

– 2C1 Гвоздика — Самохідна артилерійська установка;

– ЗСУ-23–4 Шилка — зенітна самохідна установка;

– БРДМ-2 — броньована машина-розвідник-амфібія;

– БТР-60 — бронетранспортер;

– D7 — Бульдозер Caterpillar;

– ЗІЛ-131 — Військово-вантажний автомобіль;

– Т-62 — Основний бойовий танк;

– SLICY — структура, що діє як укриття (а не транспортний засіб).

Крім того, найкращою практикою є отримання остаточного тестового набору даних із даних перевірки, як зображено на рисунку 3.5.

```
val_batches = tf.data.experimental.cardinality(val_ds)
test_ds = val_ds.take(val_batches // 5)
val_ds = val_ds.skip(val_batches // 5)

print('Number of validation batches: %d' % tf.data.experimental.cardinality(val_ds))
print('Number of test batches: %d' % tf.data.experimental.cardinality(test_ds))
```

```
Number of validation batches: 38
Number of test batches: 9
```

Рисунок 3.5 – Розділення тестового набору і набору для перевірки

На останньому етапі нашої попередньої обробки було використано можливості автоналаштування Tensorflow, щоб максимізувати продуктивність даних відповідно до апаратного забезпечення та мінімізувати кількість часу, необхідного для навчання моделей, як зображено на рисунку 3.6.

```
[ ]  AUTOTUNE = tf.data.AUTOTUNE

     train_ds = train_ds.cache().shuffle(1000).prefetch(buffer_size=AUTOTUNE)
     val_ds = val_ds.cache().prefetch(buffer_size=AUTOTUNE)
     test_ds = test_ds.cache().prefetch(buffer_size=AUTOTUNE)
```

Рисунок 3.6 – Автоналаштування Tensorflow

3.2 Визначення показників та функції оцінювання

Перш ніж розпочати імплементацію першої моделі, необхідно визначити показники та функції, які будуть використовуватись для оцінки кожної ітерації. На рисунку 3.7 зображено метрики для оцінки ітерацій нейронних мереж.

```
[ ]  num_classes=8

     metrics = [
           keras.metrics.CategoricalAccuracy(name='categorical_accuracy'),
           tfa.metrics.MatthewsCorrelationCoefficient(num_classes=8, name='MCC'),
           tfa.metrics.FBetaScore(num_classes=8, average='weighted', beta=2.0, name='F2'),
           keras.metrics.AUC(name='auc'),
           keras.metrics.AUC(name='prc', curve='PR'),
     ]
```

Рисунок 3.7 – Метрики для оцінки нейронних мереж

– CategoricalAccuracy - Підраховує, як часто прогнози збігаються з мітками;

– MatthewsCorelationCoefficient - Статистичний показник також відомий як коефіцієнт фі. Коефіцієнт кореляції Метьюза (MCC) використовується в машинному навчанні як міра якості бінарних і мультикласових класифікацій. Він враховує істинні та хибні позитивні та негативні результати і, як правило, розглядається як збалансований показник, який можна використовувати, навіть якщо класи дуже різного розміру. Значення коефіцієнта кореляції MCC знаходиться в межах від -1 до +1. Коефіцієнт +1 означає ідеальний прогноз, 0 – середній випадковий прогноз, а -1 – зворотний прогноз;

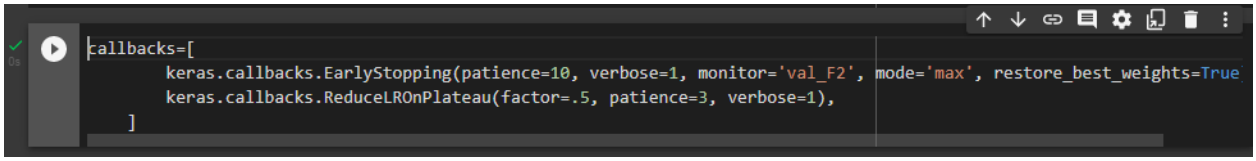
– FBetaScore - Це зважене гармонічне середнє значення точності та згадування. Вихідний діапазон [0, 1]. Працює як для класифікації з кількома класами, так і з кількома мітками;

– AUC - AUC (площа під кривою) кривих ROC (робоча характеристика приймача) або PR (точне запам’ятовування) є показниками якості двійкових класифікаторів. На відміну від точності та подібно до перехресних ентропійних втрат, ROC-AUC і PR-AUC оцінюють усі робочі точки моделі. Цей клас апроксимує AUC за допомогою суми Рімана. Під час фази накопичення показників прогнози накопичуються в межах попередньо визначених сегментів за значенням. Потім AUC обчислюється шляхом інтерполяції середніх значень для групи. Ці сегменти визначають оцінені робочі точки.

Було створено зворотні виклики щоб запобігти перенавчанню та надати можливість нейронній мережі зменшувати швидкість навчання для її оптимізації, як показано на рисунку 3.8:

– EarlyStopping - припиняє навчання, коли відстежуваний показник перестав покращуватися. Відстежуваним показником було обрано FBetaScore (F2);

– ReduceLROnPlateau - зменшує швидкість навчання, якщо показник перестав покращуватися. За замовчуванням, відстежуваний показник це втрата при навчанні для перевірки.



```
callbacks=[
        keras.callbacks.EarlyStopping(patience=10, verbose=1, monitor='val_F2', mode='max', restore_best_weights=True
        keras.callbacks.ReduceLROnPlateau(factor=.5, patience=3, verbose=1),
    ]
```

Рисунок 3.8 – Зворотні виклики щоб запобігти перенавчанню

Для порівняння результатів нейронних мереж були використані наступні метрики: втрата, крива точності запам’ятовування, F2 і коефіцієнт кореляції Метьюза. На рисунку 3.9 зображено функцію для виведення графіків з зазначеники метриками.

```
def plot_metrics(history):
  with plt.rc_context(params):
    metrics = ['loss', 'prc', 'F2', 'MCC']
    plt.figure(figsize=(10,10))
    for n, metric in enumerate(metrics):
      name = metric.replace("_"," ").capitalize()
      plt.subplot(2,2,n+1)
      plt.plot(history.epoch, history.history[metric], color=colors[0], label='Train')
      plt.plot(history.epoch, history.history['val_'+metric],
               color=colors[0], linestyle="--", label='Val')
      plt.xlabel('Epoch')
      plt.ylabel(name)
      if metric == 'loss':
        plt.ylim([0, plt.ylim()[1]])
      elif metric == 'auc':
        plt.ylim([0.8,1])
      elif metric == 'MCC':
        plt.ylim([-1,1])
      else:
        plt.ylim([0,1])

      plt.legend();
```

Рисунок 3.9 – Функція для для виведення графіків з метриками втрати

Також, важливо мати матриці плутанини (confusion matrices) для кожної моделі. На рисунку 3.10 зображено функцію виведення матриці плутанини, яка була використана для порівняння результатів нейронних мереж.

```
[ ]  def plot_cm(model, data):
        with plt.rc_context(params):
            y_true = []
            y_pred = []
            for x,y in data:
                y= tf.argmax(y,axis=1)
                y_true.append(y)
                y_pred.append(tf.argmax(model.predict(x),axis = 1))

            y_pred = tf.concat(y_pred, axis=0)
            y_true = tf.concat(y_true, axis=0)

            cm = confusion_matrix(y_true, y_pred)
            fig = plt.figure(figsize = (10,10))
            ax1 = fig.add_subplot(1,1,1)
            sns.set(font_scale=1.4) #for label size
            sns.heatmap(cm,cmap='binary', annot=True, fmt='d', xticklabels=class_names, yticklabels=class_names, annot_kws={"size": 10},
                cbar = False);
            ax1.set_ylabel('True Values',fontsize=14)
            ax1.set_xlabel('Predicted Values',fontsize=14)
            plt.show()
```

Рисунок 3.10 – Функція для виведення матриці плутанини

На цьому попередня обробка даних та визначення показників та метрик для порівняння результатів нейронних мереж було завершено та підготовлено необхідні функції для виведення результатів.

3.3 Аугменатція даних

Аугментація даних — це набір методів для штучного збільшення обсягу даних шляхом створення нових точок даних із наявних даних. Це включає внесення невеликих змін до даних або використання моделей глибокого навчання для створення нових точок даних.

Програми машинного навчання, особливо в області глибокого навчання, продовжують урізноманітнюватись і швидко збільшуватися. Використання аугментації даних може покращити підходи до розробки моделей, які орієнтовані на дані, та можуть бути хорошим інструментом проти проблем, з якими стикається світ штучного інтелекту.

Аугментація даних корисна для покращення продуктивності та результатів моделей машинного навчання шляхом формування нових і різних прикладів для навчання наборів даних. Якщо набір даних у моделі машинного навчання насичений і достатній, модель працює краще й точніше.

Для моделей машинного навчання збір і маркування даних може бути виснажливим і дорогим процесом. Трансформації в наборах даних за допомогою методів аугментації даних дозволяють зменшити ці операційні витрати.

Методи розширення даних можуть зробити моделі машинного навчання надійними, створюючи варіації, які модель може побачити в реальному світі.

Кожна створена модель нейронних мереж мала аугментацію даних, таку як:

- випадкове перевертання зображення горизонтально та вертикально;
- випадкове повернення зображення в діапазоні від 0 до 360 градусів;
- випадкове збільшення зображення на 20 відсотків;
- зміна масштабу зображення.

3.4 Модель 1. Багатошаровий перцептрон

Було створено MLP з вхідним шаром, трьома прихованими шарами з функцією активації ReLU та одним вихідним шаром, як показано на рисунку 3.11.

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 random_flip_36 (RandomFlip)  (None, 128, 128, 1)      0

 random_rotation_36 (RandomR  (None, 128, 128, 1)      0
 otation)

 random_zoom_36 (RandomZoom)  (None, 128, 128, 1)      0

 rescaling_26 (Rescaling)    (None, 128, 128, 1)       0

 flatten_23 (Flatten)        (None, 16384)             0

 dense_73 (Dense)            (None, 100)               1638500

 dense_74 (Dense)            (None, 100)               10100

 dense_75 (Dense)            (None, 100)               10100

 dense_76 (Dense)            (None, 8)                 808

=================================================================
Total params: 1,659,508
Trainable params: 1,659,508
Non-trainable params: 0
_________________________________________________________________
```

Рисунок 3.11 – Архітектура багатошарового перцептрону

Початковою швидкістю навчання було обрано 0.001 та функцією втрати категоріальну кроссентропію (categorical crossentropy), яка обчислює втрату крос-ентропії між мітками та прогнозами. Кількість епох для навчання було вказано 100, проте завдяки зворотним викликам, навчання багатошарового перцептрону зайняло 61 епоху. Метрики протягом 61 епохи навчання зображено на рисунку 3.12. На рисунку 3.13 зображено фінальні метрики після 61 епохи навчання.

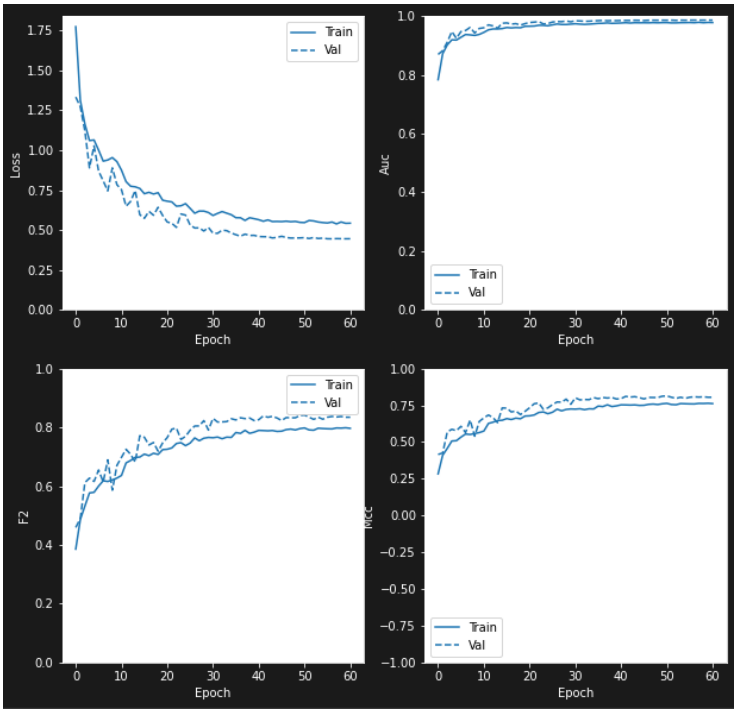


Рисунок 3.12 – Метрики багатошарового перцептрону протягом 61 епохи навчання

```
'loss': 0.4300192892551422,
'categorical_accuracy': 0.84375,
'MCC': 0.8154382705688477,
'F2': 0.8434131741523743,
'auc': 0.9873831868171692,
'prc': 0.9203866720199585}
```

Рисунок 3.13 – Фінальні метрики багатошарового перцептрону після 61 епохи навчання

Втрата багатошарового перцептрону залишилась доволі високою і становила 0.43, однак точність становила 0.98 та F2 становило 0.84, що є надзвичайно високими показниками для звичайного багатошарового перцептрону.

На рисунку 3.14 зображено матрицю плутанини багатошарового перцептрону

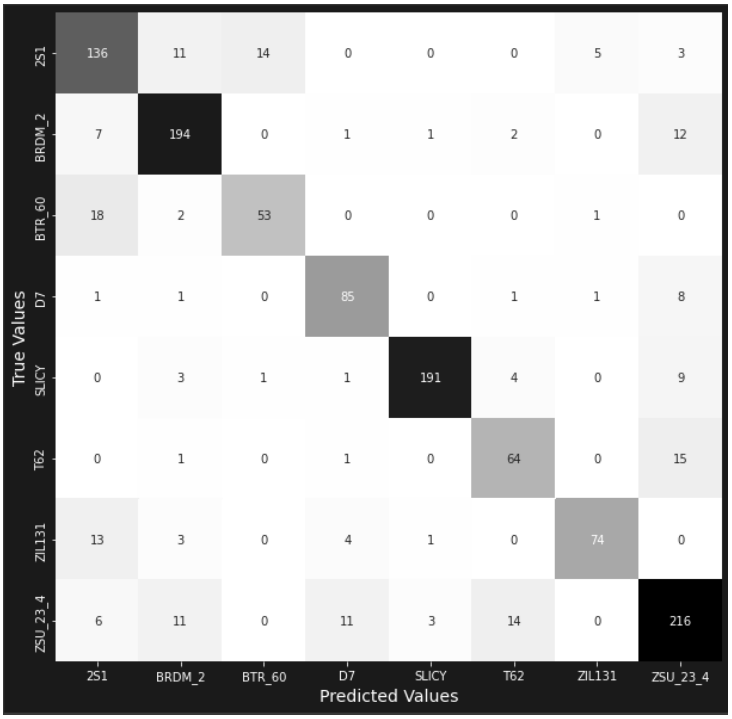


Рисунок 3.14 – Матриця плутанини багатошарового перцептрону

Для незалежної перевірки моделі були використано зображення танка Т62, яке було повернуте на 45 градусів та в кути якого був доданий додатковий шум, яке зображено на рисунку 3.15

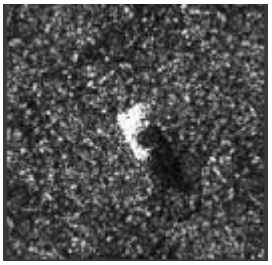


Рисунок 3.15 – Зображення танка Т62 яке було редаговане для незалежної оцінки моделі

Не дивлячись на вражаючі результати навчання багатошарового перцептрону, модель не змогла розпізнати танк T62. Результатом передбачення був клас BRDM_2 з впевненістю 48.38% як зображено на рисунку 3.16

```
1/1 [==============================] - 0s 16ms/step
'This image most likely belongs to BRDM_2 with a 48.38 percent confidence'
```

Рисунок 3.16 – Результат передбачення танка T62 багатошаровим перцептроном

Такі результати метрик і незмога розпізнати зображення поза датасетом можна пояснити тим, що перцептрон виявив окремі зони чи специфічні пікселі, притаманні зображенням об’єктів, а не навчився розпізнавати самі об’єкти.

3.5 Модель 2. Згорткова нейронна мережа №1

Для другої моделі було використано згорткову нейронну мережу з трьома згортковими шарами з розмірністю вихідного простору (тобто кількістю вихідних фільтрів у згортці) 32 з функціями активації ReLU та одним прихованим шаром розміром 128 з функцією активації ReLU, як показано на рисунку 3.17

Layer (type)	Output Shape	Param #
random_flip_37 (RandomFlip)	(None, 128, 128, 1)	0
random_rotation_37 (RandomRotation)	(None, 128, 128, 1)	0
random_zoom_37 (RandomZoom)	(None, 128, 128, 1)	0
rescaling_27 (Rescaling)	(None, 128, 128, 1)	0
conv2d_57 (Conv2D)	(None, 126, 126, 32)	320
max_pooling2d_50 (MaxPooling2D)	(None, 63, 63, 32)	0
conv2d_58 (Conv2D)	(None, 61, 61, 32)	9248
max_pooling2d_51 (MaxPooling2D)	(None, 30, 30, 32)	0
conv2d_59 (Conv2D)	(None, 28, 28, 32)	9248
max_pooling2d_52 (MaxPooling2D)	(None, 14, 14, 32)	0
flatten_24 (Flatten)	(None, 6272)	0
dense_77 (Dense)	(None, 128)	802944
dense_78 (Dense)	(None, 8)	1032

Total params: 822,792
Trainable params: 822,792
Non-trainable params: 0

Рисунок 3.17 – Архітектура першої моделі згорткової нейронної мережі

Початковою швидкістю навчання було обрано 0.001 та функцією втрати категоріальну кроссентропію (categorical crossentropy). Кількість епох для навчання було вказано 100. Завдяки зворотним викликам, навчання цієї моделі згорткової нейронної мережі зайняло 15 епох. Метрики протягом 15 епохи навчання зображено на рисунку 3.18. На рисунку 3.19 зображено фінальні метрики після 15 епохи навчання.

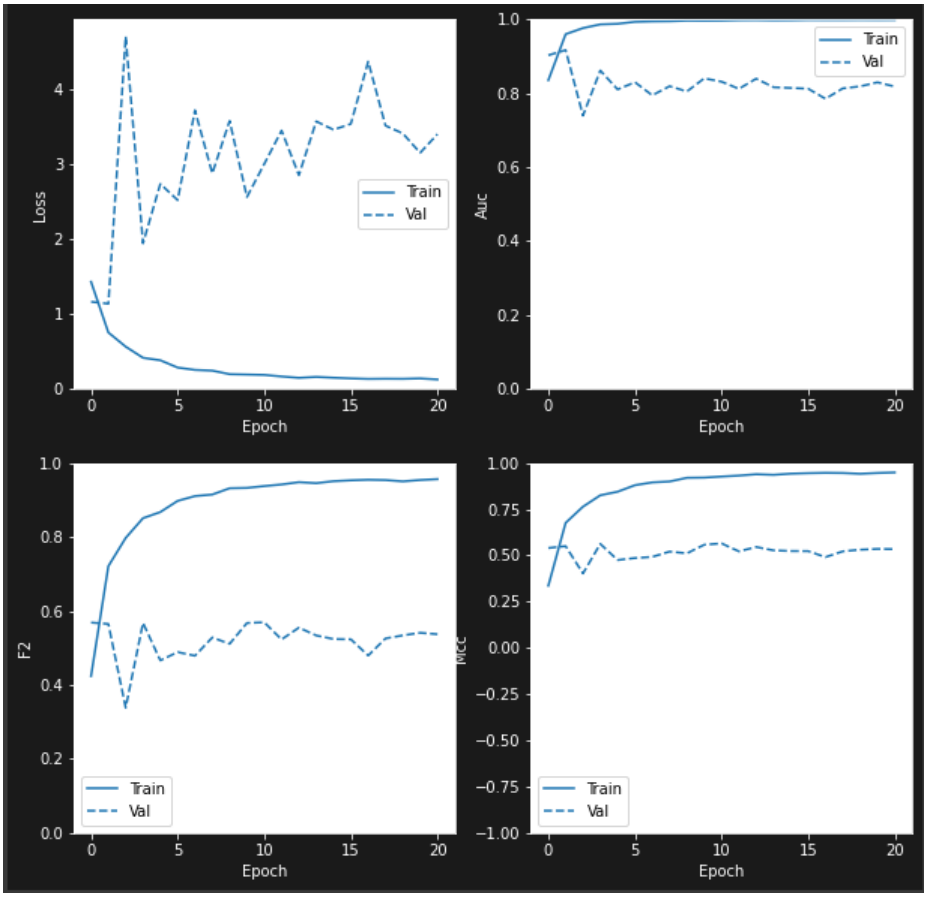


Рисунок 3.18 – Метрики першої моделі згорткової нейронної мережі протягом 15 епох навчання

```
'loss': 3.1524710655212402,
'categorical_accuracy': 0.5833333134651184,
'MCC': 0.5659202337265015,
'F2': 0.5699931383132935,
'auc': 0.8252659440040588,
'prc': 0.5151249766349792}
```

Рисунок 3.19 – Фінальні метрики першої моделі згорткової нейронної після 15 епох навчання

Як показано на рисунках 3.18 та 3.19, навчання першої моделі згорткової нейронної мережі дало доволі посередні результати. Втрата залишилась високою і становила 3.15, а точність становила 0.56, F2 становило 0.56, що значить, що лише в половині випадків мережа дасть правильне передбачення.

На рисунку 3.20 зображено матрицю плутанини першої моделі згорткової нейронної мережі.

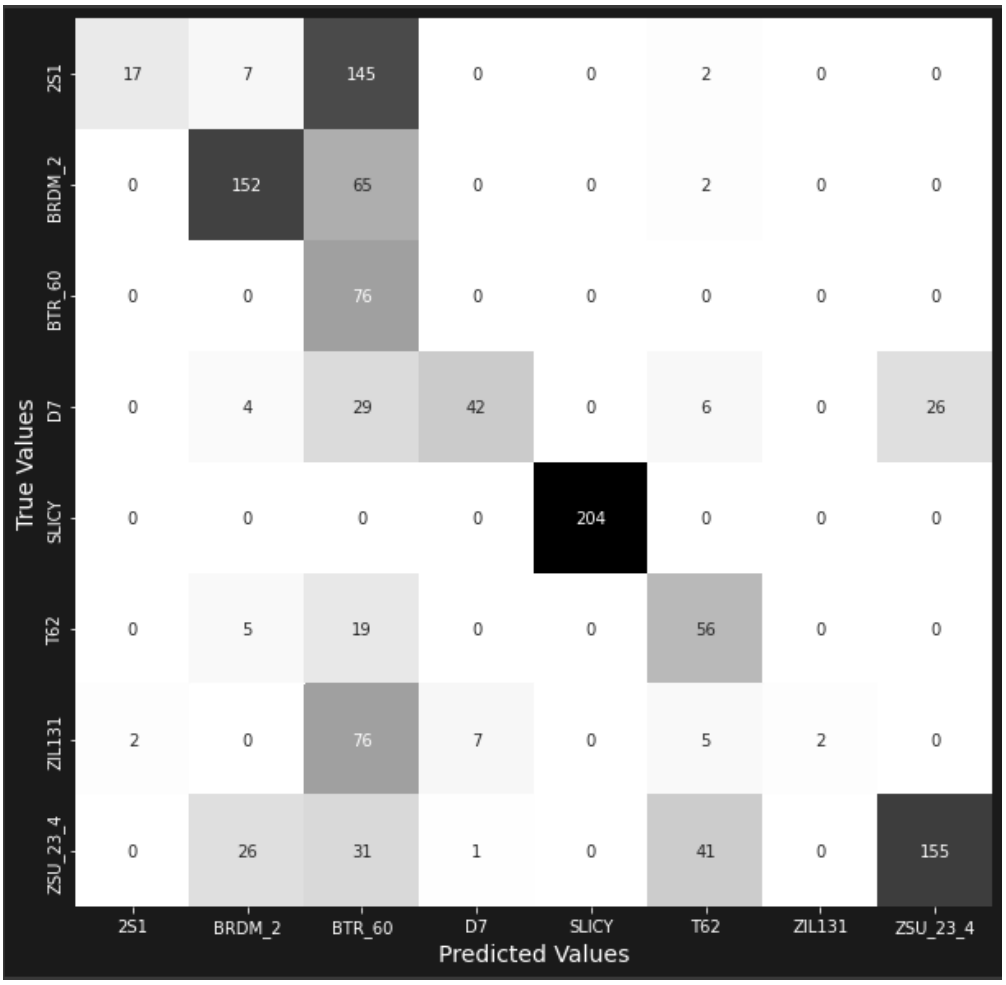


Рисунок 3.20 – Матриця плутанини першої моделі CNN

Ця модель згорткової нейронної мережі змогла розпізнати танк Т62 який був присутній в датасеті з впевненістю 44.38 відсотків, проте для незалежної перевірки моделі були використано зображення танка Т62, яке було видозмінене (рисунок 3.16). На рисунку 3.21 зображено результат передбачення. Модель не змогла розпізнати видозмінене зображення танка Т62, та з 99.53% впевненістю передбачила його як BTR_60. Приймаючи до уваги метрики та матрицю плутанини, можна зробити висновок, що ця модель не змогла добре навчитись на датасеті з аугментацією даних та дійсно розпізнавати саме об’єкти зображені в датсеті, а як і багатошаровий перцептрон виявила окремі зони чи специфічні пікселі, притаманні зображенням об’єктів. Згідно з результатами, згорткова нейронна мережа з однаковою кількістю фільтрів та однаковим розміром ядра на кожному шарі згортки не є підходящою для розпізнавання об’єктів на використаному датасеті.

```
1/1 [==============================] - 0s 18ms/step
'This image most likely belongs to BTR_60 with a 99.53 percent confidence'
```

Рисунок 3.21 – Результат передбачення танка Т62 першою моделлю згорткової нейронної мережі

3.6 Модель 3. Згорткова нейронна мережа №2

Для третьої моделі було використано згорткову нейронну мережу з трьома згортковими шарами з розмірністю вихідного простору (тобто кількістю вихідних фільтрів у згортці) 10, 20 та 30 з функціями активації ReLU та вихідним шаром з функцією активації Softmax, як показано на рисунку 3.22

Layer (type)	Output Shape	Param #
random_flip_38 (RandomFlip)	(None, 128, 128, 1)	0
random_rotation_38 (RandomRotation)	(None, 128, 128, 1)	0
random_zoom_38 (RandomZoom)	(None, 128, 128, 1)	0
rescaling_28 (Rescaling)	(None, 128, 128, 1)	0
conv2d_60 (Conv2D)	(None, 128, 128, 10)	100
max_pooling2d_53 (MaxPooling2D)	(None, 64, 64, 10)	0
conv2d_61 (Conv2D)	(None, 64, 64, 20)	1820
max_pooling2d_54 (MaxPooling2D)	(None, 32, 32, 20)	0
conv2d_62 (Conv2D)	(None, 32, 32, 30)	5430
global_average_pooling2d_5 (GlobalAveragePooling2D)	(None, 30)	0
dense_79 (Dense)	(None, 20)	620
dense_80 (Dense)	(None, 8)	168

Total params: 8,138
Trainable params: 8,138
Non-trainable params: 0

Рисунок 3.22 – Архітектура другої моделі згорткової нейронної мережі

Початковою швидкістю навчання було обрано 0.001 та функцією втрати категоріальну кроссентропію (categorical crossentropy). Кількість епох для навчання було вказано 100. Завдяки зворотним викликам, навчання цієї моделі згорткової нейронної мережі зайняло 16 епох. Метрики протягом 16 епохи навчання зображено на рисунку 3.23. На рисунку 3.24 зображено фінальні метрики після 16 епох навчання.

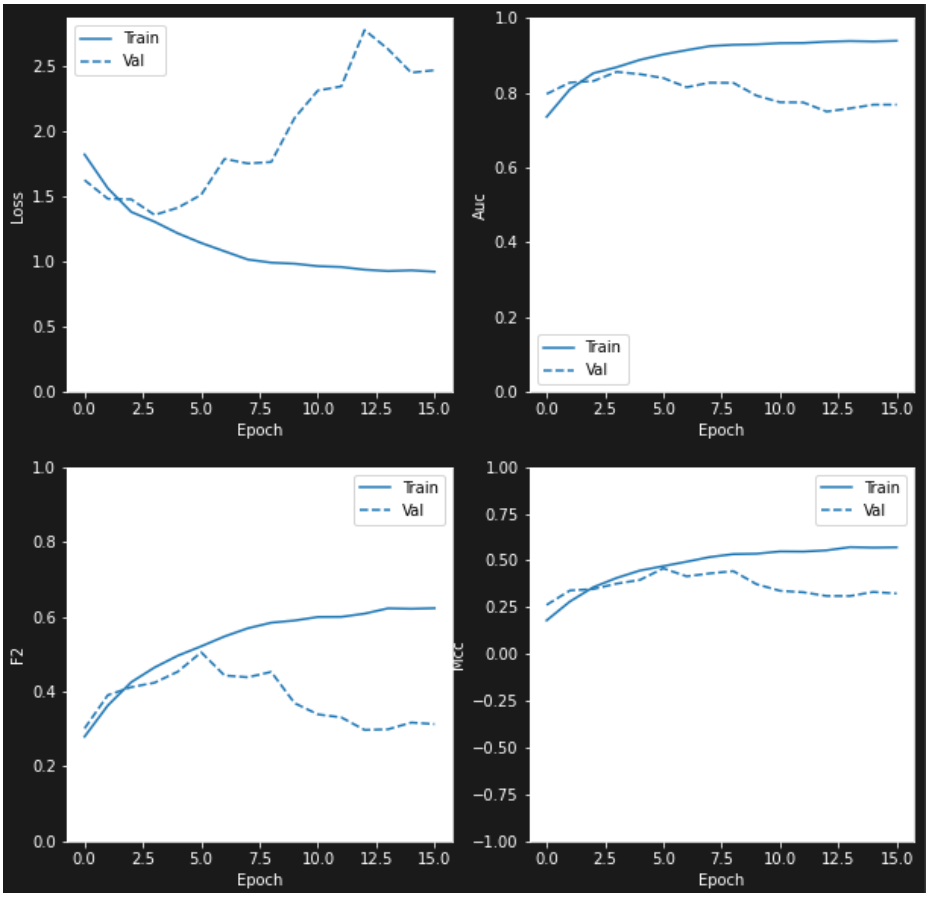


Рисунок 3.23 – Метрики другої моделі згорткової нейронної мережі протягом 16 епох навчання

```
'loss': 1.444333553314209,
'categorical_accuracy': 0.5069444179534912,
'MCC': 0.44653868675231934,
'F2': 0.5056707859039307,
'auc': 0.8532546162605286,
'prc': 0.5754493474960327}
```

Рисунок 3.24 – Фінальні метрики другої моделі згорткової нейронної після 16 епох навчання

Як показано на рисунках 3.23 та 3.24, навчання другої моделі згорткової нейронної мережі мало непогані результати на перших епох, але поступово стало гіршим, і після 16 епох, втрата залишилась високою і становила 1.44, що є кращим результатом, ніж модель згорткової нейронної мережі №1, а точність становила 0.50, F2 становило 0.50, що значить, що, як і модель згорткової нейронної мережі №1, лише в половині випадків мережа дасть правильне передбачення, що можна порівняти зі звичайним вгадуванням.

На рисунку 3.25 зображено матрицю плутанини другої моделі згорткової нейронної мережі.

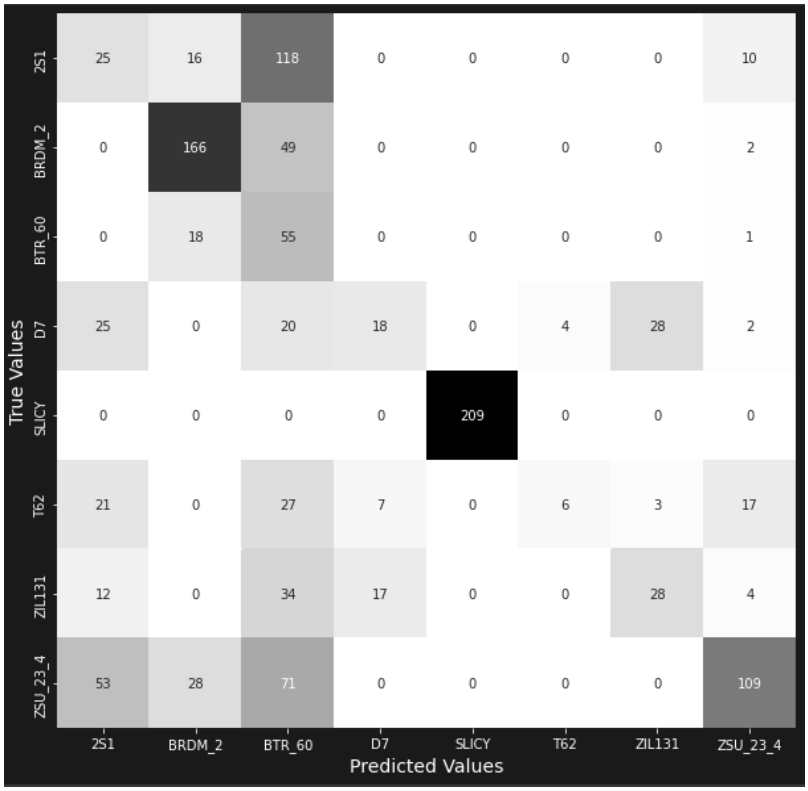


Рисунок 3.25 – Матриця плутанини другої моделі CNN

Ця модель згорткової нейронної мережі не змогла розпізнати танк Т62 який був присутній в датасеті. Для незалежної перевірки моделі були використано зображення танка Т62, яке було видозмінене (рисунок 3.16). На рисунку 3.26 зображено результат передбачення. Модель не змогла розпізнати видозмінене зображення танка Т62, та з 47.96% впевненістю передбачила його як BTR_60. Приймаючи до уваги метрики та матрицю плутанини, можна зробити висновок, що ця модель не змогла добре навчитись на датасеті з аугментацією даних. Згідно з результатами, ця архітектура згорткової нейронної мережі з поступово збільшеною кількістю фільтрів та однаковим розміром ядра на кожному шарі згортки не є підходящою для розпізнавання об’єктів на використаному датасеті.

```
1/1 [==============================] - 0s 17ms/step
'This image most likely belongs to BTR_60 with a 47.96 percent confidence'
```

Рисунок 3.26 – Результат передбачення танка Т62 другою моделлю згорткової нейронної мережі

3.7 Модель 4. Нейронна мережа з архітектурою XCeption

Для четвертої моделі було використано нейронну мережу з архітектурою XCeption. Загальна архітектура XCeption зображена на рисунку 2.5. На початку цієї моделі було додано вхідний шар розміром 128 та аугментацію даних та додано вихідний шар розміром 8 та функцією активації Softmax.

Початковою швидкістю навчання було обрано 0.001 та функцією втрати категоріальну кроссентропію (categorical crossentropy). Кількість епох для навчання

було вказано 100. Завдяки зворотним викликам, навчання цієї моделі згорткової нейронної мережі зайняло 17 епох. Метрики протягом 17 епох навчання зображено на рисунку 3.27. На рисунку 3.28 зображено фінальні метрики після 17 епох навчання.

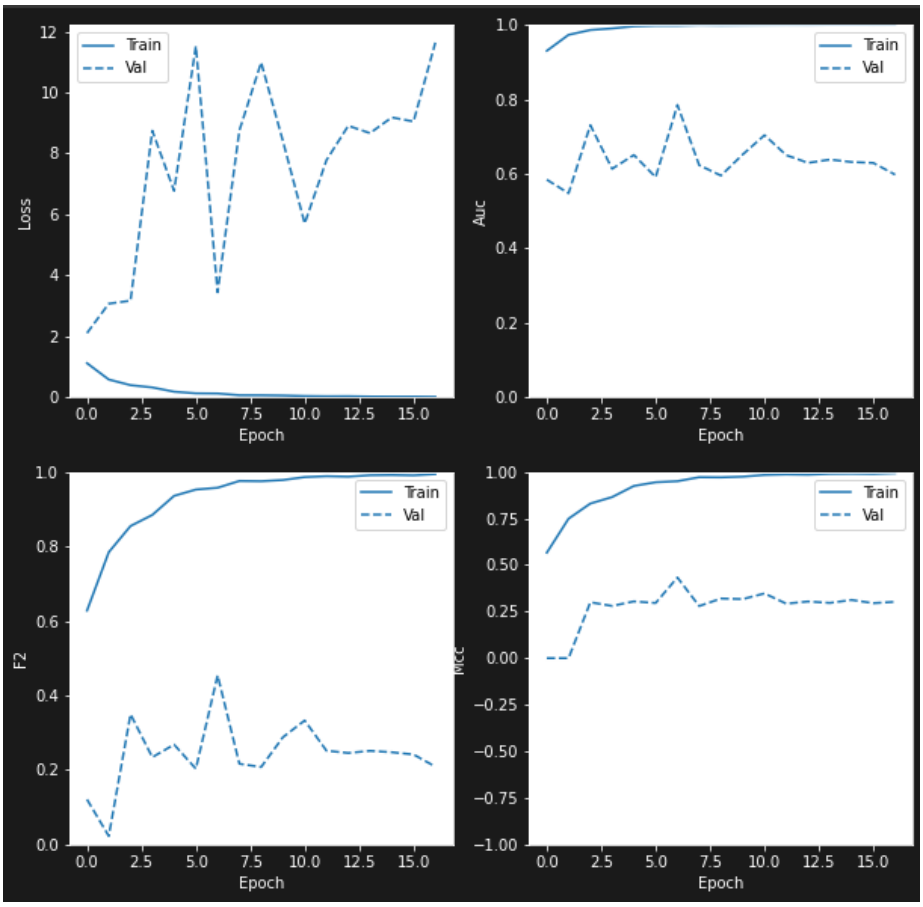


Рисунок 3.27 – Метрики нейронної мережі з архітектурою XCeption протягом 17 епох навчання

```
'loss': 3.3422811031341553,
'categorical_accuracy': 0.4652777910232544,
'MCC': 0.42147356271743774,
'F2': 0.4347888231277466,
'auc': 0.7974045872688293,
'prc': 0.4710308313369751}
```

Рисунок 3.28 – Фінальні метрики нейронної мережі з архітектурою XCeption після 17 епох навчання

Як показано на рисунках 3.27 та 3.28, навчання нейронної мережі з архітектурою XCeption дало незадовільні результати. Втрата залишилась дуже високою і становила 3.34, а точність становила 0.46, F2 становило 0.43, що значить, що ця архітектура нейронної мережі не підходить для поставленої задачі з використаним датасетом.

На рисунку 3.29 зображено матрицю плутанини другої моделі згорткової нейронної мережі.

Ця модель нейронної мережі не змогла розпізнати танк Т62 який був присутній в датасеті. Для незалежної перевірки моделі були використано зображення танка Т62, яке було видозмінене (рисунок 3.16). На рисунку 3.30 зображено результат передбачення. Модель не змогла розпізнати видозмінене зображення танка Т62, та з 84.45% впевненістю передбачила його як BTR_60. Приймаючи до уваги метрики та матрицю плутанини, можна зробити висновок, що ця модель не змогла добре навчитись на датасеті з аугментацією даних. Згідно з результатами, архітектура нейронної мережі XCeption не є підходящою для розпізнавання об’єктів на використаному датасеті.

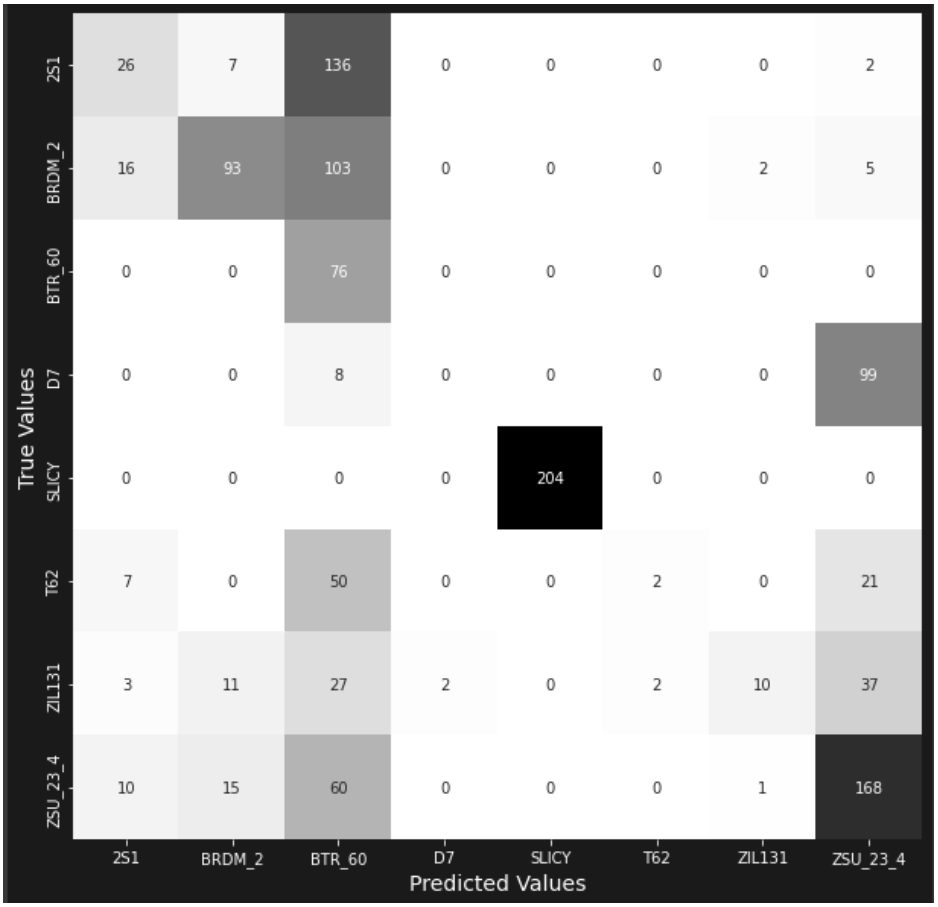


Рисунок 3.29 – Матриця плутанини нейронної мережі з архітектурою XCeption

```
1/1 [==============================] - 0s 20ms/step
'This image most likely belongs to BTR_60 with a 84.45 percent confidence'
```

Рисунок 3.30 – Результат передбачення танка T62 моделлю нейронної мережі з архітектурою XCeption

3.8 Модель 5. Нейронна мережа з архітектурою VGG

Для четвертої моделі було використано нейронну мережу з архітектурою VGG19. Використовуючи функціонал Keras, ваги для цієї моделі були попередньо натреновані для ImageNet. Загальна архітектура VGG зображена на рисунку 2.7. На початку цієї моделі було додано вхідний шар розміром 128 та аугментацію даних та додано вихідний шар розміром 8 та функцією активації Softmax.

Початковою швидкістю навчання було обрано 0.001 та функцією втрати категоріальну кроссентропію (categorical crossentropy). Кількість епох для навчання було вказано 100. Завдяки зворотним викликам, навчання цієї моделі згорткової нейронної мережі зайняло 27 епох. Метрики протягом 27 епох навчання зображено на рисунку 3.31. На рисунку 3.32 зображено фінальні метрики після 27 епох навчання.

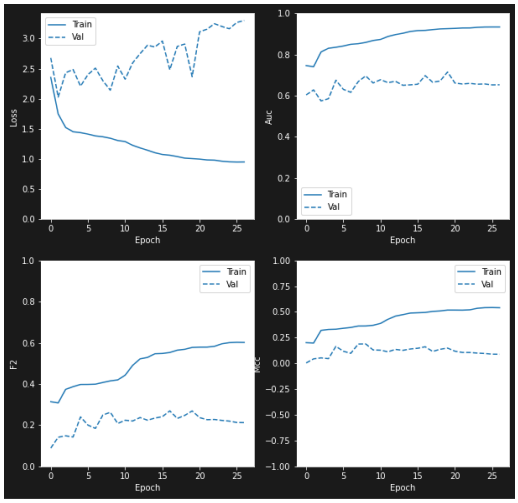


Рисунок 3.31 – Метрики нейронної мережі з архітектурою VGG протягом 27 епох навчання

```
'loss': 2.44622802734375,
'categorical_accuracy': 0.3333333432674408,
'MCC': 0.19377602636814117,
'F2': 0.29696810245513916,
'auc': 0.6985099911689758,
'prc': 0.2483445554971695}
```

Рисунок 3.32 – Фінальні метрики нейронної мережі з архітектурою VGG після 27 епох навчання

Як показано на рисунках 3.31 та 3.32, навчання нейронної мережі з архітектурою VGG дало доволі погані результати. Втрата залишилась дуже високою і становила 2.44, а точність становила 0.33, F2 становило 0.29, що значить, що ця архітектура нейронної мережі з вагами на базі ImageNet не підходить для поставленої задачі з використаним датасетом.

На рисунку 3.33 зображено матрицю плутанини другої моделі згорткової нейронної мережі.

Ця модель нейронної мережі не змогла розпізнати танк Т62 який був присутній в датасеті. Для незалежної перевірки моделі були використано зображення танка Т62, яке було видозмінене (рисунок 3.16). На рисунку 3.34 зображено результат передбачення. Модель не змогла розпізнати видозмінене зображення танка Т62, та з 43.47% впевненістю передбачила його як 2S1. Приймаючи до уваги метрики та матрицю плутанини, можна зробити висновок, що ця модель не змогла добре навчитись на датасеті з аугментацією даних. Згідно з результатами, архітектура нейронної мережі VGG19 з вагами ImageNet не є підходящою для розпізнавання об’єктів на використаному датасеті.

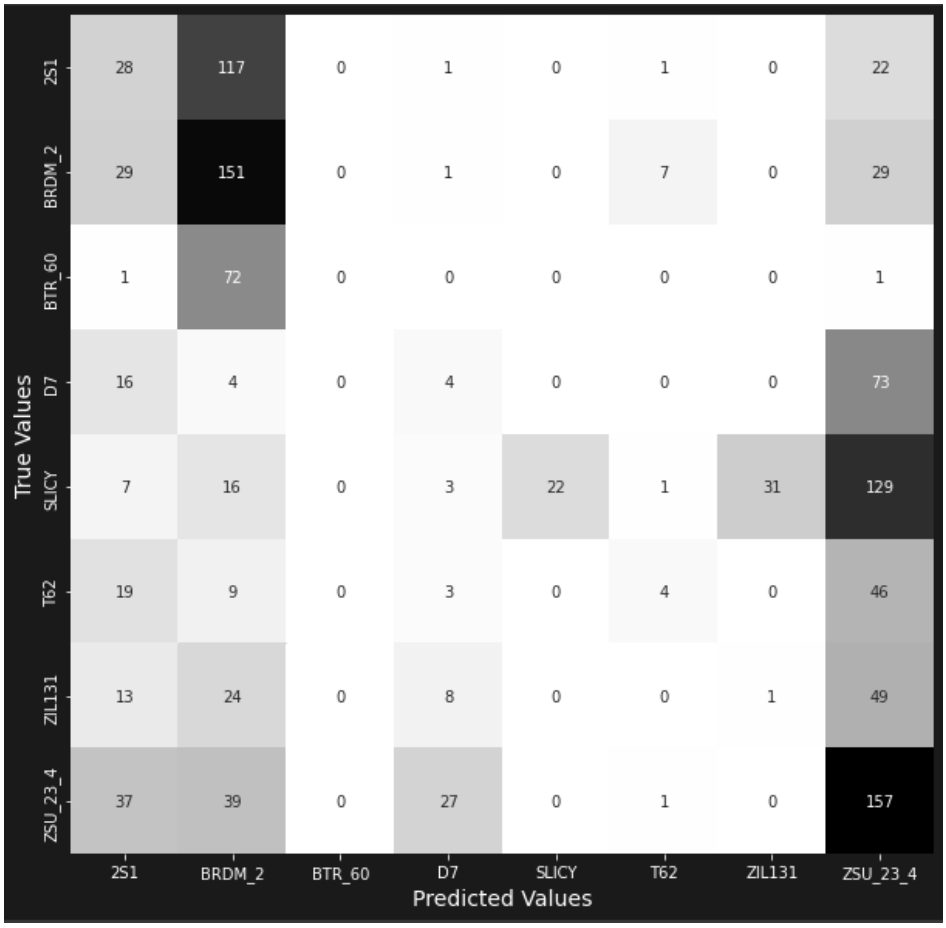


Рисунок 3.33 – Матриця плутанини нейронної мережі з архітектурою VGG19

```
1/1 [==============================] - 0s 18ms/step
'This image most likely belongs to 2S1 with a 43.47 percent confidence'
```

Рисунок 3.34 – Результат передбачення танка T62 моделлю нейронної мережі з архітектурою VGG19 з вагами ImageNet

3.9 Модель 6. Згорткова нейронна мережа №3

Попередні нейронні мережі не дали бажаних результатів, тому маючи досвід попередніх згорткових нейронних мереж з однаковою та поступово збільшеною кількістю фільтрів та однаковим розміром ядра, моделей нейронних мереж з архітектурою XCeption та VGG та використаних нейронних мереж в проаналізованих існуючих дослідженнях про розпізнавання об’єктів на супутникових знімках, включаючи до уваги особливості використаного датасету, було створено згорткову нейронну мережу саме для виявлення об’єктів на зображеннях у використаному датасеті від MSTAR. Було створено згорткову нейронну мережу з аугментацією даних, чотирма згортковими шарами з розмірністю вихідного простору (тобто кількістю вихідних фільтрів у згортці) 128, 64, 32 та 16 з функціями активації ReLU та розміром ядра 5, 4, 3 та 2 відповідно, двома повністю зв’язаними шарами розміром 1024 та 128 з функціями активації ReLU та вихідним шаром розміром 8 з функцією активації Softmax, як показано на рисунку 3.35.

Layer (type)	Output Shape	Param #
rescaling_31 (Rescaling)	(None, 128, 128, 3)	0
random_flip_41 (RandomFlip)	(None, 128, 128, 3)	0
random_rotation_41 (RandomRotation)	(None, 128, 128, 3)	0
random_zoom_41 (RandomZoom)	(None, 128, 128, 3)	0
conv2d_69 (Conv2D)	(None, 124, 124, 128)	9728
max_pooling2d_59 (MaxPooling2D)	(None, 62, 62, 128)	0
conv2d_70 (Conv2D)	(None, 59, 59, 64)	131136
max_pooling2d_60 (MaxPooling2D)	(None, 29, 29, 64)	0
conv2d_71 (Conv2D)	(None, 27, 27, 32)	18464
max_pooling2d_61 (MaxPooling2D)	(None, 13, 13, 32)	0
conv2d_72 (Conv2D)	(None, 12, 12, 16)	2064
max_pooling2d_62 (MaxPooling2D)	(None, 6, 6, 16)	0
flatten_26 (Flatten)	(None, 576)	0
dense_84 (Dense)	(None, 1024)	590848
dense_85 (Dense)	(None, 128)	131200
dropout_11 (Dropout)	(None, 128)	0
dense_86 (Dense)	(None, 8)	1032

Total params: 884,472
Trainable params: 884,472
Non-trainable params: 0

Рисунок 3.35 – Архітектура фінальної моделі CNN

Початковою швидкістю навчання було обрано 0.001 та функцією втрати категоріальну кроссентропію (categorical crossentropy). Кількість епох для навчання було вказано 100. Завдяки зворотним викликам, навчання цієї моделі згорткової нейронної мережі зайняло 37 епох. Метрики протягом 37 епох навчання зображено на рисунку 3.36. На рисунку 3.37 зображено фінальні метрики після 37 епох навчання.

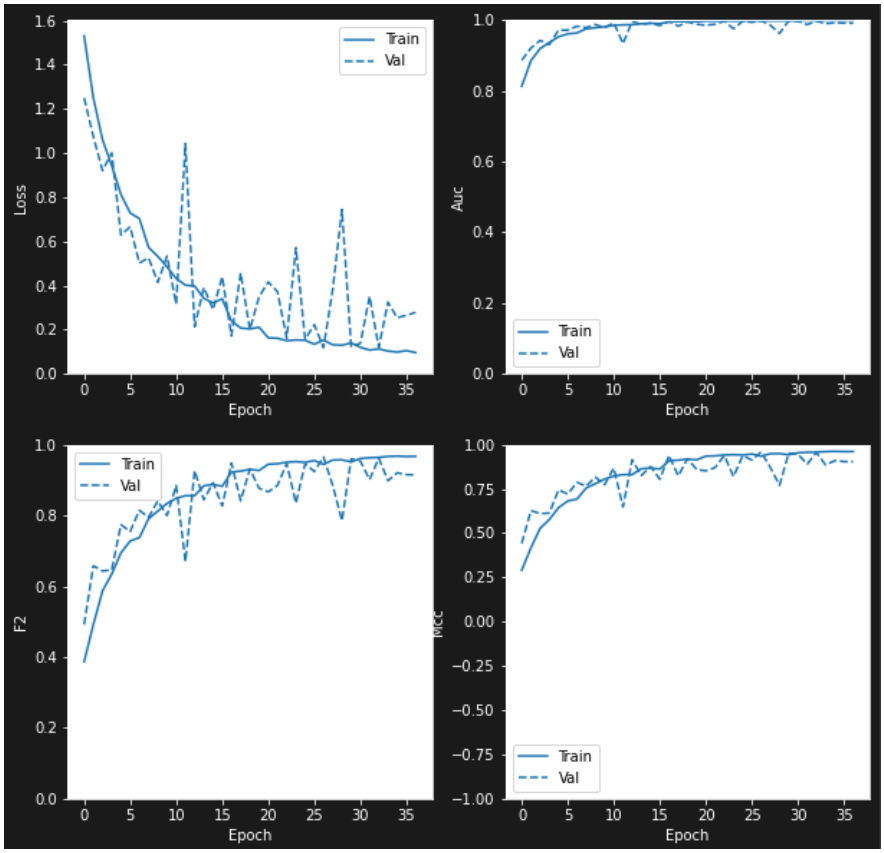


Рисунок 3.36 – Метрики фінальної згорткової нейронної мережі протягом 37 епох навчання

```
'loss': 0.12086480110883713,
'categorical_accuracy': 0.9375,
'MCC': 0.9264187812805176,
'F2': 0.9374281167984009,
'auc': 0.9988735318183899,
'prc': 0.9925348162651062}
```

Рисунок 3.37 – Фінальні метрики фінальної згорткової нейронної мережі після 37 епох навчання

Як показано на рисунках 3.36 та 3.37, навчання фінальної моделі згорткової нейронної мережі дало задовільні результати. Втрата залишилась низькою і становила 0.12, а точність становила 0.93, F2 становило 0.93, що значить, що ця архітектура нейронної мережі підходить для поставленої задачі з використаним датасетом.

На рисунку 3.38 зображено матрицю плутанини другої моделі згорткової нейронної мережі.

Ця модель згорткової нейронної мережі не тільки змогла розпізнати танк Т62 який був присутній в датасеті, а і пройшла незалежну перевірки, для якої було використано зображення танка Т62, яке було видозмінене (рисунок 3.16). На рисунку 3.39 зображено результат передбачення. Модель змогла розпізнати видозмінене зображення танка Т62 з 94.36%. Приймаючи до уваги метрики та матрицю плутанини, можна зробити висновок, що ця модель змогла добре навчитись на датасеті з аугментацією даних. Згідно з результатами, ця архітектура згорткової нейронної мережі з поступово зменшуючуюся кількістю фільтрів та розміром ядра на кожному шарі згортки є підходящою для розпізнавання об’єктів на використаному датасеті.

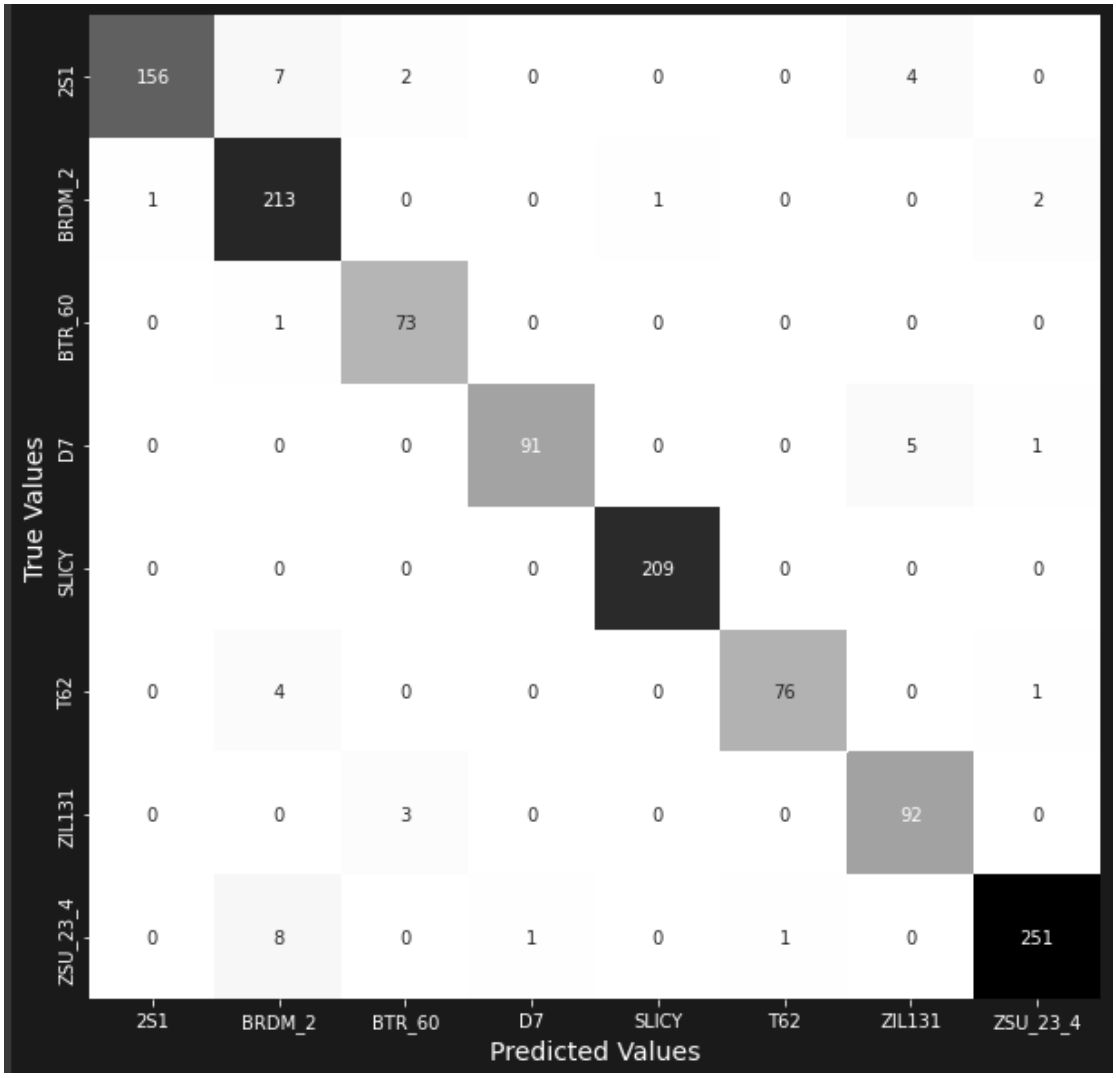


Рисунок 3.38 – Матриця плутанини фінальної CNN

```
1/1 [==============================] - 0s 17ms/step
'This image most likely belongs to T62 with a 94.36 percent confidence'
```

Рисунок 3.39 – Результат передбачення танка T62 фінальною моделлю згорткової нейронної мережі

3.10 Порівняння результатів навчання та передбачення досліджених моделей нейронних мереж

Проаналізувавши всі попередньо обрані нейронні мережі, зібравши їх матриці плутанини та фінальні метрики, необхідно порівняти їх результати.

Високі показники багатошарового перцептрону і незмога розпізнати зображення поза датасетом можна пояснити тим, що перцептрон виявив окремі зони чи специфічні пікселі, а не навчився розпізнавати самі об’єкти.

Моделі 1 та 2 згорткових нейронних мереж показали незадовільні результати, з чого можна зробити висновок, що вони не змогли навчитися на використаному датасеті. Використані архітектури моделей згорткових нейронних мереж 1 та 2 не підходять для поставленої задачі. Найгірші результати навчання показали нейронні мережі з архітектурою XCeption та VGG. Ці нейронні мережі не змогли навчитися на використаному датасеті та їх архітектури не підходять для поставленої задачі. Модель 3 згорткової нейронної мережі змогла не тільки навчитися на використаному датасеті, а й розпізнати зображення, якого не було в датасеті. Це свідчить про те, що архітектура третьої моделі згорткової нейронної мережі найбільше підходить для розпізнавання військової техніки на супутникових знімках з використаним датасетом. На таблиці 3.1 зображені фінальні метрики досліджених нейронних мереж та кількість епох їх навчання. У додатку А представлено повну програмну реалізацію застосунку

Таблиця 3.1 – Метрики досліджених нейронних мереж

НМ	Втрата	Кат. точн.	К. к. М.	F2	AUC	Епохи
Б. перц.	0.43	0.84	0.81	0.84	0.98	61
CNN 1	3.15	0.58	0.56	0.56	0.82	15
CNN 2	1.44	0.50	0.44	0.50	0.85	16
XCeption	3.34	0.46	0.42	0.43	0.79	17
VGG	2.44	0.33	0.19	0.29	0.69	27
CNN 3	0.12	0.93	0.92	0.93	0.99	37

Висновки до розділу 3

Було проаналізовано програмну реалізацію, яка включала попередню обробку даних для завантаження датасету для навчання та валідації, визначення метрик для подальшого порівняння результатів навчання нейронних мереж та доведено важливість аугментації даних. Було імплементовано різні види нейронних мереж, такі як: багатошаровий перцептрон, модель згорткової нейронної мережі 1 з однаковою кількістю фільтрів, модель згорткової нейронної мережі 2 з поступово збільшеною кількістю фільтрів, нейронну мережу з архітектурою XCeption, нейронну мережу з архітектурою VGG та модель згорткової нейронної мережі 3, архітектура якої була розроблена спеціально для поставленої задачі. Було проведено порівняння результатів навчання нейронних мереж. Серед проаналізованих нейронних мереж найкращі результати показала модель 3 згорткової нейронної

мережі, яка змогла не тільки навчитися на використаному датасеті, а й розпізнати зображення, якого не було в датасеті.

ВИСНОВКИ

В ході виконання кваліфікаційної роботи було досліджено та проаналізовано нейронні мережі для розпізнавання військової техніки на супутникових знімках. Проаналізовано існуючі роботи по розпізнаванню об’єктів на супутникових знімках. Програмну реалізацію було створено за допомогою хмарного сервісу Google Colab, в якій було імплементовано багатошаровий перцептрон, три моделі згорткових нейронних мереж, нейронну мережу з архітектурою XCeption та нейронну мережу з архітектурою VGG. Було проведено порівняння результатів навчання нейронних мереж на наборі даних для виявлення та розпізнавання рухомих і стаціонарних цілей (MSTAR). Найкращі результати показала модель 3 згорткової нейронної мережі, головною відмінністю якої було поступове зменшення кількості фільтрів та розміру ядра на шарах згортки. Ця модель змогла не тільки навчитися на використаному датасеті, а й розпізнати зображення, якого не було в датасеті.

У методичній частині було створено лабораторну роботу на тему розпізнавання різних пейзажів з супутникових знімків із застосуванням багатошарового перцептрону та CNN та дослідження впливу аугментації даних на результати навчання нейронних мереж.

У спеціальній частині з охорони праці та безпеки у надзвичайних ситуаціях було виконано аналіз умов праці в серверному приміщенні закладу ресторанного типу. Перевірено забезпечення вимог охорони праці. Виявлено, що оцінка умов праці на робочому місці відноситься до IV категорії, коли спостерігається робота у несприятливих умовах праці.

Завдання кваліфікаційної роботи було виконано в повній мірі, зокрема:

1) проаналізовано сучасний стан задачі розпізнавання об’єктів, зокрема військової техніки на супутникових знімках;

2) досліджено існуючі технології та засоби для розпізнавання військової техніки на супутникових знімках; реалізовано моделі нейронних мереж для виявлення військової техніки;

3) досліджено показники навчання моделей нейронних мереж для розпізнавання та виявлення об’єктів на супутникових і аерофотознімках.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1) У. Алганчі, М. Сойдас та Е. Сертел, «Порівняльні дослідження підходів глибокого навчання для виявлення літаків із супутникових зображень із дуже високою роздільною здатністю», 2020 р. 332 с.

2) Дж. Альгазо, А. Башар, Г. Латіф і М. Зікрія, «Виявлення морських суден за допомогою згорткових нейронних мереж із супутникових зображень», 2021 р. 123 с.

3) Г. Алібрагім і С. А. Людвіг, «Оптимізація за гіперпараметрами: порівняння генетичного алгоритму з пошуком по сітці та байєсівською оптимізацією», 2021 р. 65 с.

4) Ф. Камран, М. Шахзад і Ф. Шафаїт, «Автоматизоване виявлення військової техніки за аерофотознімками з низької висоти», 2019 р. 8 с.

5) Дж. Алзубі, А. Найяр та А. Кумар, «Машинне навчання від теорії до алгоритмів: огляд», 2018 р. 43 с.

6) А. Аммар, А. Коубаа, М. Ахмед і А. Саад, «Обробка аерофотознімків для виявлення автомобілів за допомогою згорткових нейронних мереж: порівняння швидшого r-cnn і yolov3», 2019 р. 6 с.

7) А. К. Тудоджу, «Виявлення літаків, транспортних засобів і кораблів на аерофотознімках і супутникових знімках за допомогою еволюційного глибокого навчання», 2021 р. 321 с.

8) Ю. Лю, Н. Лю, Х. Хуо та Т. Фанг, «Виявлення транспортних засобів на супутникових зображеннях високої роздільної здатності за допомогою згорткових нейронних мереж з з’єднаним шаром», 2015 р. 54 с.

9) Дж. Ма, З. Чжоу, Б. Ван, Х. Цзун і Ф. Ву, «Виявлення кораблів на оптичних супутникових зображеннях за допомогою спрямованих обмежувальних прямокутників на основі передбачення центру корабля та орієнтації», 2019 р. 98 с.

10) В. Вей і Дж. Чжан, «Технологія дистанційного зондування зображення літака на основі глибокого навчання», 2015 р., 48 с.

11) Ю. Лі, К. Фу, Х. Сун і З. Сун, «Структура виявлення повітряних суден на основі підкріпленого навчання та згорткових нейронних мереж у зображеннях дистанційного зондування», 2017 р. 431 с.

12) Д. Кумар, Х. Чжан, Х. Су та С. Вей, «Точне виявлення об'єктів на основі швидкого R-CNN у зображеннях дистанційного зондування», 2019 р. 75 с.

13) Г.-Х. Ні, П. Чжан, Х. Ніу, Ю. Доу та Ф. Ся, «Виявлення судна з використанням мультиблокового детектора з однократним пострілом, навченого трансфером», 2020 р., 123 с.

14) А. П. Антоніо-Жав’єр Галлего та П. Гіл, «Автоматична класифікація кораблів на основі оптичних аерофотознімків за допомогою згорткових нейронних мереж», 2018 р. 87 с.

15) Н. м. Асземі та П. Домінік, «Оптимізація гіперпараметрів у згортковій нейронній мережі з використанням генетичних алгоритмів», 2019 р. 656 с.

16) М. Бахі та М. Батуш, «Глибоке навчання для віртуального скринінгу на основі лігандів у відкритті ліків», 2018 р. 876 с.

17) Е. Бочінскі, Т. Сенст і Т. Сікора, «Гіперпараметрична оптимізація для комітетів згорткової нейронної мережі на основі еволюційних алгоритмів», 2017 р. 86 с.

18) Дж. Карлет і Б. Абайова, «Швидке виявлення транспортних засобів на аерофотознімках», 2017 р. 423 с.

19) Х. Чень, С. Сян, К.-Л. Лю та Ч.-Х. Пан, «Виявлення літаків мережами глибокого навчання», 2013 р. 134 с.

20) У. Алганчі, М. Сойдас та Є. Сертел, «Порівняльні дослідження підходів глибокого навчання для виявлення літаків із супутникових зображень із дуже високою роздільною здатністю», 2020 р. 458 с.

21) Дж. Альгазо, А. Башар, Г. Латіф і М. Зікрія, «Виявлення морських суден за допомогою згорткових нейронних мереж із супутникових зображень», 2021 р. 437 с.

22) Х. Алібрагім і С. А. Людвіг, «Гіперпараметрична оптимізація: порівняння генетичного алгоритму з пошуком у сітці та байєсівською оптимізацією», 2021 р. 1559 с.

23) А. Аммар, А. Коубаа, М. Ахмед і А. Саад, «Обробка аерофотознімків для виявлення автомобілів за допомогою згорткових нейронних мереж: порівняння швидшої r-cnn і yolov3», 2019 р. 543 с.

24) А. П. Антоніо-Джав'єр Галегго та Ф. Гіл, «Автоматична класифікація кораблів на основі оптичних аерофотознімків за допомогою згорткових нейронних мереж», 2018 р. 54 с.

25) Н. М. Асземі та Ф. Домінік, «Оптимізація гіперпараметрів у згортковій нейронній мережі з використанням генетичних алгоритмів», 2019 р. 278 с.

26) М. Бахі та М. Батуш, «Глибоке навчання для віртуального скринінгу на основі лігандів у відкритті ліків», 2018 р. 5 с.

27) Е. Бочінскі, Т. Сенст і Т. Сікора, «Гіперпараметрична оптимізація для комітетів згорткової нейронної мережі на основі еволюційних алгоритмів», 2017 р. 3928 с.

28) Дж. Карлет і Б. Абайова, «Швидке виявлення транспортних засобів на аерофотознімках», 2017 р. 1709 с.

29) Х. Чень, С. Сян, К.-Л. Лю та Ч.-Х. Пан, «Виявлення літаків мережами глибокої віри», 2013 р. 58 с.

30) М. Клейсен і Б. Де Мур, «Пошук гіперпараметрів у машинному навчанні», 2015 р. 1502 с.

31) А. В. А. Сордова, В. К. Квісп, Р. Дж. К. Інка, В. Н. Чок'юхуайта та І. К. Гутіррез, «Нові підходи та інструменти для виявлення кораблів на оптичних супутникових зображеннях», 2020 р. 123 с.

32) З. Денг, Х. Сун, С. Жу, Дж. Жао, Л. Лей і Х. Зу, «Багатомасштабне виявлення об'єктів у зображеннях дистанційного зондування за допомогою згорткових нейронних мереж», 2018 р. 22 с.

33) Т. Домхан, Дж. Т. Спрінгенберг і Ф. Хаттер, «Прискорення автоматичної оптимізації гіперпараметрів глибоких нейронних мереж шляхом екстраполяції кривих навчання», 2015 р. 54 с.

34) Б. Еванс, Х. Аль-Сахаф, Б. Сюе та М. Чжан, «Еволюційне глибоке навчання: підхід генетичного програмування до класифікації зображень», 2018 р. 6 с.

35) С. Фуджіно, Т. Хатанака, Н. Морі та К. Мацумото, «Еволюційне глибоке навчання на основі глибокої згорткової нейронної мережі для розпізнавання розкадровок аніме», 2019 р. 398 с.

36) К. Сян, Л. Као, М. Ченг, К. Ванг та Дж. Лі, «Виявлення транспортних засобів на основі глибоких нейронних мереж у супутникових зображеннях», 2015 р. 187 с.

37) А. Мансур, А. Хассан, В. М. Хуссейн та Е. Саїд, «Автоматизоване виявлення транспортних засобів на супутникових зображеннях із використанням глибокого навчання», 2019 р. 127 с.

38) К.-К. Мао, Х.-М. Сун, Ю.-Б. Лю та П.-К. Джіа, «Міні-yolov3: детектор об’єктів у реальному часі для вбудованих програм», 2019 р. 538 с.

39) А. Мартін і Д. Камачо, «Розвиток архітектури та гіперпараметрів dnns для виявлення зловмисних програм», 2020 р. 377 с.

40) В. МакКінні та ін., «pandas: фундаментальна бібліотека Python для аналізу даних і статистики», 2011 р. 9 с.

41) П. Нірі, «Автоматичне налаштування гіперпараметрів у глибоких згорткових нейронних мережах з використанням асинхронного навчання з підкріпленням», 2018 р. 77 с.

42) Г.-С. Ні, П. Жанг, С. Ніу, І. Доу та Ф. Ся, «Виявлення кораблів за допомогою детектора з багаторазовими блоками, навченими перенесенням», 2017 р. 106 с.

43) С. Нікбахт, К. Анітеску, та Т. Рабжук, «Оптимізація гіперпараметрів нейронної мережі за допомогою генетичного алгоритму», 2021 р. 426 с.

44) А. Зіссерманом і К. Симоньян, «Дуже глибокі згорткові мережі для розпізнавання великомасштабних зображень», 2017 р. 127 с.

45) Т. Оффофф, С. Путтеманс, В. Калогіру, Ж.-П. Робін і Т. Гедеме, «Виявлення транспортних засобів і суден на супутникових зображеннях: порівняльне дослідження одноразових детекторів», 2020 р. 1217 с.

46) А. Павлік, Дж. Сігал, Х. Шарп і М. Петре, «Краудсорсинг документації наукового програмного забезпечення: приклад проекту документації numpy», 2014 р. 36 с.

47) Б. Прабха, К. Рамеш і А. Гіта, «Процедура планування завдань на основі генетичного алгоритму для економічно ефективних хмарних центрів обробки даних», 2021 р. 543 с.

48) М. Прітт і Г. Черн, «Класифікація супутникових зображень із глибоким навчанням», 2017 р. 7 с.

49) С. Чіфанг, І. Гочінг і Л. Пін, «Виявлення повітряним судном зображення дистанційного зондування високої роздільної здатності на основі швидшої моделі r-cnn і моделі SSD», 2018 р. 137 с.

50) К. Раза та С. Хонг, «Швидкий і точний дизайн виявлення риби з покращеною моделлю yolo-v3 і навчанням передачі», 2020 р. 16 с.

51) Дж. Редмон і А. Фархаді, «Yolov3: поступове вдосконалення», 2018 р. 21 с.

52) Дж. Ріжсдійк, Л. Ву, Г. Перін та С. Пікек, «Навчання з підкріпленням для налаштування гіперпараметрів у аналізі бічних каналів на основі глибокого навчання». 2021 р. 71 с.

53) А. Рохан, М. Рабах і С.-Х. Кім, «Виявлення та відстеження об’єктів реального часу на основі згорткової нейронної мережі для parrot ar drone 2», 2019 р. 584 с.

54) Р. Сараванан і П. Суджата, «Сучасні методи алгоритмів машинного навчання: перспектива підходів до навчання під наглядом у класифікації даних», 2018 р. 949 с.

55) Л. Ши, З. Тян , Т. Ванг, С. Су, Дж. Ліу та Дж. Жанг, «Виявлення літаків на зображеннях дистанційного зондування на основі деконволюції та позиціонування уваги», 2021 р. 4260 с.

ДОДАТОК А

Програмна реалізація додатку

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.preprocessing import image_dataset_from_directory
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import *
from tensorflow.keras.applications.vgg19 import VGG19
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix
import tensorflow_addons as tfa

params = {"ytick.color" : "w",
          "xtick.color" : "w",
          "axes.labelcolor" : "w",
          "axes.edgecolor" : "w",
          "figure.figsize" : (10,10),
          "axes.titlecolor" : 'w',
          "axes.facecolor" : 'w',
          "figure.facecolor" : 'k'}

colors = plt.rcParams['axes.prop_cycle'].by_key()['color']

%matplotlib inline

image_size = (128, 128)
batch_size = 32

from google.colab import drive
drive.mount('/content/drive')
```

```
train_ds = image_dataset_from_directory('drive/MyDrive/MSTAR',
                                        subset='training',
                                        image_size=image_size,
                                        labels='inferred',
                                        validation_split=.2,
                                        seed=10,
                                        label_mode='categorical',
                                        color_mode='grayscale',
                                      batch_size=batch_size)
val_ds = image_dataset_from_directory('drive/MyDrive/MSTAR',
                                        subset='validation',
                                        image_size=image_size,
                                        labels='inferred',
                                        validation_split=.2,
                                        seed=10,
                                        label_mode='categorical',
                                        color_mode='grayscale',
                                        batch_size=batch_size)
class_names = train_ds.class_names
val_batches = tf.data.experimental.cardinality(val_ds)
test_ds = val_ds.take(val_batches // 5)
val_ds = val_ds.skip(val_batches // 5)

print('Number of validation batches: %d' % tf.data.experimental.cardinality(val_ds))
print('Number of test batches: %d' % tf.data.experimental.cardinality(test_ds))

AUTOTUNE = tf.data.AUTOTUNE

train_ds = train_ds.cache().shuffle(1000).prefetch(buffer_size=AUTOTUNE)
val_ds = val_ds.cache().prefetch(buffer_size=AUTOTUNE)
test_ds = test_ds.cache().prefetch(buffer_size=AUTOTUNE)

num_classes=8
```

```
metrics = [
      keras.metrics.CategoricalAccuracy(name='categorical_accuracy'),
      tfa.metrics.MatthewsCorrelationCoefficient(num_classes=8, name='MCC'),
      tfa.metrics.FBetaScore(num_classes=8, average='weighted', beta=2.0, name='F2'),
      keras.metrics.AUC(name='auc'),
      keras.metrics.AUC(name='prc', curve='PR'),
]

callbacks=[
               keras.callbacks.EarlyStopping(patience=10,  verbose=1,  monitor='val_F2',
mode='max', restore_best_weights=True),
        keras.callbacks.ReduceLROnPlateau(factor=.5, patience=3, verbose=1),
    ]

model1 = keras.Sequential([
    InputLayer(input_shape=(image_size + (1,))),
    tf.keras.layers.RandomFlip("horizontal_and_vertical"),
    tf.keras.layers.RandomRotation(1),
    tf.keras.layers.RandomZoom(-0.2),
    tf.keras.layers.Rescaling(1./128),
    Flatten(),
    Dense(100, activation='relu'),
    Dense(100, activation='relu'),
    Dense(100, activation='relu'),
    Dense(num_classes, activation='softmax'),
])

model1.summary()

model1.compile(
  optimizer=keras.optimizers.Adam(0.001),
  loss='categorical_crossentropy',
  metrics=metrics,
```

```
)

history1 = model1.fit(train_ds, epochs=100, callbacks=callbacks,
validation_data=val_ds)

def holdout_results(model):
    result = model.evaluate(test_ds)
    return dict(zip(model.metrics_names, result))

def plot_metrics(history):
  with plt.rc_context(params):
    metrics = ['loss', 'auc', 'F2', 'MCC']
    plt.figure(figsize=(10,10))
    for n, metric in enumerate(metrics):
      name = metric.replace("_"," ").capitalize()
      plt.subplot(2,2,n+1)
      plt.plot(history.epoch, history.history[metric], color=colors[0], label='Train')
      plt.plot(history.epoch, history.history['val_'+metric],
               color=colors[0], linestyle="--", label='Val')
      plt.xlabel('Epoch')
      plt.ylabel(name)
      if metric == 'loss':
        plt.ylim([0, plt.ylim()[1]])
      elif metric == 'auc':
        plt.ylim([0,1])
      elif metric == 'MCC':
        plt.ylim([-1,1])
      else:
        plt.ylim([0,1])

      plt.legend();

def plot_cm(model, data):
    with plt.rc_context(params):
```

```
        y_true = []
        y_pred = []
        for x,y in data:
            y= tf.argmax(y,axis=1)
            y_true.append(y)
            y_pred.append(tf.argmax(model.predict(x),axis = 1))

        y_pred = tf.concat(y_pred, axis=0)
        y_true = tf.concat(y_true, axis=0)

        cm = confusion_matrix(y_true, y_pred)
        fig = plt.figure(figsize = (10,10))
        ax1 = fig.add_subplot(1,1,1)
        sns.set(font_scale=1.4) #for label size
           sns.heatmap(cm,cmap='binary', annot=True, fmt='d', xticklabels=class_names,
yticklabels=class_names, annot_kws={"size": 10},
            cbar = False);
        ax1.set_ylabel('True Values',fontsize=14)
        ax1.set_xlabel('Predicted Values',fontsize=14)
        plt.show()

image = keras.utils.load_img(
    path="drive/MyDrive/HB14938_rotated_45.png",
    color_mode='grayscale',
    target_size=(128,128)
)

image_array = keras.utils.img_to_array(image)
image_array = tf.expand_dims(image_array, 0)

image

def predict_t72(model):
    predictions = model.predict(image_array)
```

```
      return f'This image most likely belongs to {class_names[np.argmax(predictions)]}
with a {100 * np.max(predictions):.2f} percent confidence'

predict_t72(model1)

plot_metrics(history1)

plot_cm(model1, val_ds)

holdout_results(model1)

plot_cm(model1, test_ds)

model2 = tf.keras.Sequential([
  tf.keras.layers.InputLayer(input_shape=(image_size + (1,))),
  tf.keras.layers.RandomFlip('horizontal_and_vertical'),
  tf.keras.layers.RandomRotation(1),
  tf.keras.layers.RandomZoom(-0.2),
  tf.keras.layers.Rescaling(1./128),
  tf.keras.layers.Conv2D(32, 3, activation='relu'),
  tf.keras.layers.MaxPooling2D(),
  tf.keras.layers.Conv2D(32, 3, activation='relu'),
  tf.keras.layers.MaxPooling2D(),
  tf.keras.layers.Conv2D(32, 3, activation='relu'),
  tf.keras.layers.MaxPooling2D(),
  tf.keras.layers.Flatten(),
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dense(num_classes, activation='softmax')
])
model2.summary()

model2.compile(
  optimizer=keras.optimizers.Adam(0.001),
  loss='categorical_crossentropy',
```

```
  metrics=metrics)

history2 = model2.fit(train_ds, epochs=100, callbacks=callbacks,
validation_data=val_ds)

plot_metrics(history2)

plot_cm(model2, val_ds)

holdout_results(model2)

predict_t72(model2)

plot_cm(model2, test_ds)

CNN = Sequential([
  InputLayer(input_shape=(image_size + (1,))),
  tf.keras.layers.RandomFlip('horizontal_and_vertical'),
  tf.keras.layers.RandomRotation(1),
  tf.keras.layers.RandomZoom(height_factor=(0.2, 0.5), width_factor=(0.2, 0.5)),
  tf.keras.layers.Rescaling(1./128),
  Conv2D(filters=10, kernel_size=3, activation='relu', padding='same'),
  MaxPooling2D(),
  Conv2D(filters=20, kernel_size=3, activation='relu', padding='same'),
  MaxPooling2D(),
  Conv2D(filters=30, kernel_size=3, activation='relu', padding='same'),
  GlobalAveragePooling2D(),
  Dense(20, activation='relu'),
  Dense(num_classes, activation='softmax'),
])
CNN.summary()

CNN.compile(optimizer=keras.optimizers.Adam(0.001),
              loss='categorical_crossentropy',
```

```
              metrics=metrics)

cnn_hist = CNN.fit(train_ds, epochs=50, callbacks=callbacks, validation_data=val_ds)

plot_metrics(cnn_hist)

plot_cm(CNN, val_ds)

holdout_results(CNN)

predict_t72(CNN)

plot_cm(CNN, test_ds)

def make_model(input_shape, num_classes):
    inputs = keras.Input(shape=input_shape)
    # Image augmentation block
    x = resize_and_rescale(inputs)
    x = data_augmentation(x)

    # Entry block
    x = keras.layers.Conv2D(32, 3, strides=2, padding="same")(x)
    x = keras.layers.BatchNormalization()(x)
    x = keras.layers.Activation("relu")(x)

    x = keras.layers.Conv2D(64, 3, padding="same")(x)
    x = keras.layers.BatchNormalization()(x)
    x = keras.layers.Activation("relu")(x)

    previous_block_activation = x  # Set aside residual

    for size in [128, 256, 512, 728]:
        x = keras.layers.Activation("relu")(x)
        x = keras.layers.SeparableConv2D(size, 3, padding="same")(x)
```

```
        x = keras.layers.BatchNormalization()(x)

        x = keras.layers.Activation("relu")(x)
        x = keras.layers.SeparableConv2D(size, 3, padding="same")(x)
        x = keras.layers.BatchNormalization()(x)

        x = keras.layers.MaxPooling2D(3, strides=2, padding="same")(x)

        # Project residual
        residual = keras.layers.Conv2D(size, 1, strides=2, padding="same")(
            previous_block_activation
        )
        x = keras.layers.add([x, residual])  # Add back residual
        previous_block_activation = x  # Set aside next residual

    x = keras.layers.SeparableConv2D(1024, 3, padding="same")(x)
    x = keras.layers.BatchNormalization()(x)
    x = keras.layers.Activation("relu")(x)

    x = keras.layers.GlobalAveragePooling2D()(x)
    if num_classes == 2:
        activation = "sigmoid"
        units = 1
    else:
        activation = "softmax"
        units = num_classes

    x = keras.layers.Dropout(0.5)(x)
    outputs = keras.layers.Dense(units, activation=activation)(x)
    return keras.Model(inputs, outputs)


xception = make_model(input_shape=image_size + (1,), num_classes=8)
xception.summary()
```

```
def entry_flow(inputs) :

    x = keras.layers.RandomFlip('horizontal_and_vertical')(inputs)
    x = keras.layers.RandomRotation(1)(x)
    x = keras.layers.RandomZoom(height_factor=(0.2, 0.5), width_factor=(0.2, 0.5))(x)
    x = keras.layers.Rescaling(1./128)(x)

    x = keras.layers.Conv2D(32, 3, strides = 2, padding='same')(x)
    x = keras.layers.BatchNormalization()(x)
    x = keras.layers.Activation('relu')(x)

    x = keras.layers.Conv2D(64,3,padding='same')(x)
    x = keras.layers.BatchNormalization()(x)
    x = keras.layers.Activation('relu')(x)

    previous_block_activation = x

    for size in [128, 256, 728] :

        x = keras.layers.Activation('relu')(x)
        x = keras.layers.SeparableConv2D(size, 3, padding='same')(x)
        x = keras.layers.BatchNormalization()(x)

        x = keras.layers.Activation('relu')(x)
        x = keras.layers.SeparableConv2D(size, 3, padding='same')(x)
        x = keras.layers.BatchNormalization()(x)

        x = keras.layers.MaxPooling2D(3, strides=2, padding='same')(x)

                            residual    =    keras.layers.Conv2D(size,    1,    strides=2,
padding='same')(previous_block_activation)

        x = keras.layers.Add()([x, residual])
```

```
        previous_block_activation = x

    return x

def middle_flow(x, num_blocks=8) :

    previous_block_activation = x

    for _ in range(num_blocks) :

        x = keras.layers.Activation('relu')(x)
        x = keras.layers.SeparableConv2D(728, 3, padding='same')(x)
        x = keras.layers.BatchNormalization()(x)

        x = keras.layers.Activation('relu')(x)
        x = keras.layers.SeparableConv2D(728, 3, padding='same')(x)
        x = keras.layers.BatchNormalization()(x)

        x = keras.layers.Activation('relu')(x)
        x = keras.layers.SeparableConv2D(728, 3, padding='same')(x)
        x = keras.layers.BatchNormalization()(x)

        x = keras.layers.Add()([x, previous_block_activation])
        previous_block_activation = x

    return x

def exit_flow(x) :

    previous_block_activation = x

    x = keras.layers.Activation('relu')(x)
    x = keras.layers.SeparableConv2D(728, 3, padding='same')(x)
    x = keras.layers.BatchNormalization()(x)
```

```
    x = keras.layers.Activation('relu')(x)
    x = keras.layers.SeparableConv2D(1024, 3, padding='same')(x)
    x = keras.layers.BatchNormalization()(x)

    x = keras.layers.MaxPooling2D(3, strides=2, padding='same')(x)

                    residual     =     keras.layers.Conv2D(1024,     1,     strides=2,
padding='same')(previous_block_activation)
    x = keras.layers.Add()([x, residual])

    x = keras.layers.Activation('relu')(x)
    x = keras.layers.SeparableConv2D(728, 3, padding='same')(x)
    x = keras.layers.BatchNormalization()(x)

    x = keras.layers.Activation('relu')(x)
    x = keras.layers.SeparableConv2D(1024, 3, padding='same')(x)
    x = keras.layers.BatchNormalization()(x)

    x = keras.layers.GlobalAveragePooling2D()(x)

    activation = "softmax"
    units = 8

    x = keras.layers.Dropout(0.5)(x)
    x = keras.layers.Dense(units, activation=activation)(x)

    return x

inputs = keras.Input(shape=image_size + (1,))
outputs = exit_flow(middle_flow(entry_flow(inputs)))
xception = keras.Model(inputs, outputs)

xception.summary()
```

```
xception.compile(
    optimizer=keras.optimizers.Adam(1e-3),
    loss="categorical_crossentropy",
    metrics=metrics,
)
xception_hist = xception.fit(
    train_ds, epochs=100, callbacks=callbacks, validation_data=val_ds,
)

plot_metrics(xception_hist)

holdout_results(xception)

predict_t72(xception)

plot_cm(xception, val_ds)

train_ds = image_dataset_from_directory('drive/MyDrive/MSTAR',
                                        subset='training',
                                        image_size=image_size,
                                        labels='inferred',
                                        validation_split=.2,
                                        seed=10,
                                        label_mode='categorical',
                                        color_mode='rgb',
                                        batch_size=batch_size)

val_ds = image_dataset_from_directory('drive/MyDrive/MSTAR',
                                        subset='validation',
                                        image_size=image_size,
                                        labels='inferred',
                                        validation_split=.2,
                                        seed=10,
```

```
                                        label_mode='categorical',
                                        color_mode='rgb',
                                        batch_size=batch_size)

val_batches = tf.data.experimental.cardinality(val_ds)
test_ds = val_ds.take(val_batches // 5)
val_ds = val_ds.skip(val_batches // 5)

print('Number of validation batches: %d' % tf.data.experimental.cardinality(val_ds))
print('Number of test batches: %d' % tf.data.experimental.cardinality(test_ds))

AUTOTUNE = tf.data.AUTOTUNE

train_ds = train_ds.cache().shuffle(1000).prefetch(buffer_size=AUTOTUNE)
val_ds = val_ds.cache().prefetch(buffer_size=AUTOTUNE)
test_ds = test_ds.cache().prefetch(buffer_size=AUTOTUNE)

vgg_base = VGG19(include_top=False, weights= 'imagenet', input_shape=(128, 128, 3))
keras.utils.plot_model(vgg_base)

vgg_model = keras.models.Sequential([
  Input(shape=image_size + (3,)),
  tf.keras.layers.RandomFlip('horizontal_and_vertical'),
  tf.keras.layers.RandomRotation(1),
  tf.keras.layers.RandomZoom(height_factor=(-0.2, -0.5), width_factor=(-0.2, -0.5)),
  tf.keras.layers.Rescaling(1./128),
  vgg_base,
  Flatten(),
  Dense(1024, activation='relu'),
  Dropout(0.5),
  Dense(num_classes, activation='softmax')
])

keras.utils.plot_model(vgg_model)
```

```
vgg_model.compile(
    optimizer=keras.optimizers.Adam(1e-3),
    loss="categorical_crossentropy",
    metrics=metrics,)

vgg_hist = vgg_model.fit(
    train_ds, epochs=100, callbacks=callbacks, validation_data=val_ds,
)

plot_metrics(vgg_hist)

plot_cm(vgg_model, val_ds)

holdout_results(vgg_model)

image = keras.utils.load_img(
    path="drive/MyDrive/HB14938_rotated_45.png",
    color_mode='rgb',
    target_size=(128,128)
)

image

image_array = keras.utils.img_to_array(image)
image_array = tf.expand_dims(image_array, 0)

predict_t72(vgg_model)

plot_cm(vgg_model,test_ds)

last_model = tf.keras.Sequential([
  tf.keras.layers.InputLayer(input_shape=(image_size + (3,))),
  tf.keras.layers.Rescaling(1./128),
```

```
  tf.keras.layers.RandomFlip('horizontal_and_vertical'),
  tf.keras.layers.RandomRotation(1),
  tf.keras.layers.RandomZoom(-0.2),
  tf.keras.layers.Conv2D(128, 5, activation='relu'),
  tf.keras.layers.MaxPooling2D(),
  tf.keras.layers.Conv2D(64, 4, activation='relu'),
  tf.keras.layers.MaxPooling2D(),
  tf.keras.layers.Conv2D(32, 3, activation='relu'),
  tf.keras.layers.MaxPooling2D(),
  tf.keras.layers.Conv2D(16, 2, activation='relu'),
  tf.keras.layers.MaxPooling2D(),
  tf.keras.layers.Flatten(),
  tf.keras.layers.Dense(1024, activation='relu'),
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dropout(0.5),
  tf.keras.layers.Dense(num_classes, activation='softmax')
])
last_model.summary()

last_model.compile(
    optimizer=keras.optimizers.Adam(1e-3),
    loss="categorical_crossentropy",
    metrics=metrics,
)

last_model_hist = last_model.fit(
    train_ds, epochs=100, callbacks=callbacks, validation_data=val_ds,
)

plot_metrics(last_model_hist)

plot_cm(last_model, val_ds)

holdout_results(last_model)
```

```
image = keras.utils.load_img(
    path="drive/MyDrive/HB14938_rotated_45.png",
    color_mode='rgb',
    target_size=(128,128)
)

image_array = keras.utils.img_to_array(image)
image_array = tf.expand_dims(image_array, 0)

image

predict_t72(last_model)

plot_cm(last_model, test_ds)
```